ADPG

Arenadata Postgres

Arenadata Postgres (ADPG) — это коммерческий дистрибутив базы данных PostgreSQL с дополнительным Enterprise-функционалом, а также с корпоративной поддержкой и документацией.

Версия CURRENT Язык: RU

Содержание

1. Концепции
   - Сценарии использования
   - Архитектура ADPG
   - Балансировка нагрузки
   - Мониторинг
   - Управление пулом подключений

2. Подготовка окружения
   - Требования к оборудованию
   - Требования к сети
   - Программные требования

3. Начало работы
   - Установка
     - Online-установка
       - Установка ADCM
       - Подготовка хостов
       - Установка кластера ADPG
         - Создание кластера
         - Добавление сервисов
         - Добавление хостов в кластер
         - Добавление компонентов
         - Настройка сервисов
         - Настройка кластера
         - Установка кластера
       - Установка мониторинга
     - Offline-установка
       - Установка ADCM
       - Подготовка хостов
       - Установка кластера Enterprise Tools
         - Создание кластера
         - Добавление сервисов
         - Добавление хостов в кластер
         - Добавление компонентов
         - Установка кластера
       - Установка кластера ADPG
         - Создание кластера
         - Добавление сервисов
         - Добавление хостов в кластер
         - Добавление компонентов
         - Настройка сервисов
         - Настройка кластера
         - Импорт настроек ET
         - Установка кластера
       - Установка мониторинга
   - Подключение к ADPG
     - psql
     - Сторонние клиентские инструменты
       - pgAdmin
       - DBeaver

4. Управление доступом
   - Аутентификация
     - Конфигурация PG_HBA
     - Аутентификация по паролю
     - Kerberos
     - LDAP
     - Включение SSL-шифрования
   - Роли и привилегии базы данных
   - Управление доступом на уровне строк и столбцов
     - Политики защиты строк
     - Политики защиты столбцов

5. Администрирование СУБД
   - Управление кластером через ADCM
     - Кластерные действия
       - Обновление кластера
     - Сервисные действия
       - ADPG
       - ADPG Control Agents
       - Balancer
       - Chrony
       - Etcd
       - Monitoring
   - Метрики мониторинга
   - Настройка отказоустойчивости
     - Обзор
     - Логическая репликация
     - Настройка балансировки нагрузки
     - Настройка PgBouncer
     - Переключение с лидера на реплику
   - Резервное копирование и восстановление
     - SQL-дамп
     - Резервное копирование на уровне файловой системы
     - PITR
     - Резервное копирование и восстановление с использованием pgBackRest
   - Оптимизация данных
     - Vacuuming
     - Сбор статистики с помощью ANALYZE
   - Логирование
   - Оптимизация производительности
   - Настройка huge pages в Linux

6. Работа с данными
   - Структура каталога данных
   - Обзор объектов базы данных
     - Базы данных
     - Схемы
     - Таблицы
       - Обзор таблиц
       - Наследование
       - Секционирование
       - Внешние таблицы
       - TOAST
     - Табличные пространства
     - Представления и материализованные представления
     - Последовательности
     - Индексы
     - Триггеры
     - Пользовательские функции и процедуры
   - Управление параллелизмом
     - Уровни изоляции транзакций
     - Блокировки
   - Запросы к базе данных
     - Пример работы с psql
     - Использование псевдонимов таблиц и столбцов
     - Типы запросов
       - Оператор JOIN
       - Подзапросы
       - Комбинирование запросов
       - Общие табличные выражения (CTE)
       - Агрегатные функции
       - Оконные функции
     - Использование комплексных типов данных
       - JSON/JSONB
       - JSONPath
       - XML
       - hstore
       - bytea
     - Полнотекстовый поиск
     - Анализ запросов
   - Работа с расширениями
     - timescaledb
   - Серверное программирование
     - Использование PL/pgSQL
     - Использование PL/Python
     - Использование функций на языке C
     - Использование PL/Perl
     - Использование PL/Tcl
   - Шифрование данных с помощью pgcrypto

7. Справочные материалы
   - Конфигурационные параметры
   - Поддерживаемые расширения

8. Релизы
   - Релизы ADPG
   - Известные проблемы

Сценарии использования

Евгения Кузина

Назад к содержанию

ADPG — мощная реляционная система управления базами данных (РСУБД). ADPG основана на объектно-реляционной СУБД с открытым исходным кодом PostgreSQL. ADPG можно использовать в критически важных и вспомогательных приложениях для бизнеса в различных отраслях.

ADPG является хорошим решением в следующих сценариях:

- **Финансовая индустрия**. ADPG полностью совместима с ACID и подходит для рабочих нагрузок обработки транзакций в реальном времени (Online Transaction Processing, OLTP). Это также аналитическая база данных, которую можно интегрировать с математическим ПО.
- **Промышленное производство**. ADPG представляет собой долгосрочное хранилище данных, предлагающее надежное хранение по низкой себестоимости. ADPG — идеальный выбор, поскольку поддерживает автоматическое переключение при сбое, полное дублирование данных и почти нулевое время простоя при обновлении серверов.
- **Данные ГИС**. ADPG включает мощное расширение GIS — PostGIS, которое предоставляет сотни функций для обработки географических данных в различных форматах.
- **Веб-технологии и рабочие нагрузки NoSQL**. Веб-сайтам может потребоваться обрабатывать тысячи или сотни тысяч запросов в секунду для обслуживания клиентов. ADPG отвечает этому требованию и может использоваться в качестве хранилища в масштабируемых приложениях. ADPG работает с современными веб-фреймворками, включая Django (Python), node.js (JavaScript), Hibernate (Java), Ruby on rails, PHP и другие.
  Благодаря возможностям репликации веб-сайты можно масштабировать до любого количества серверов баз данных.

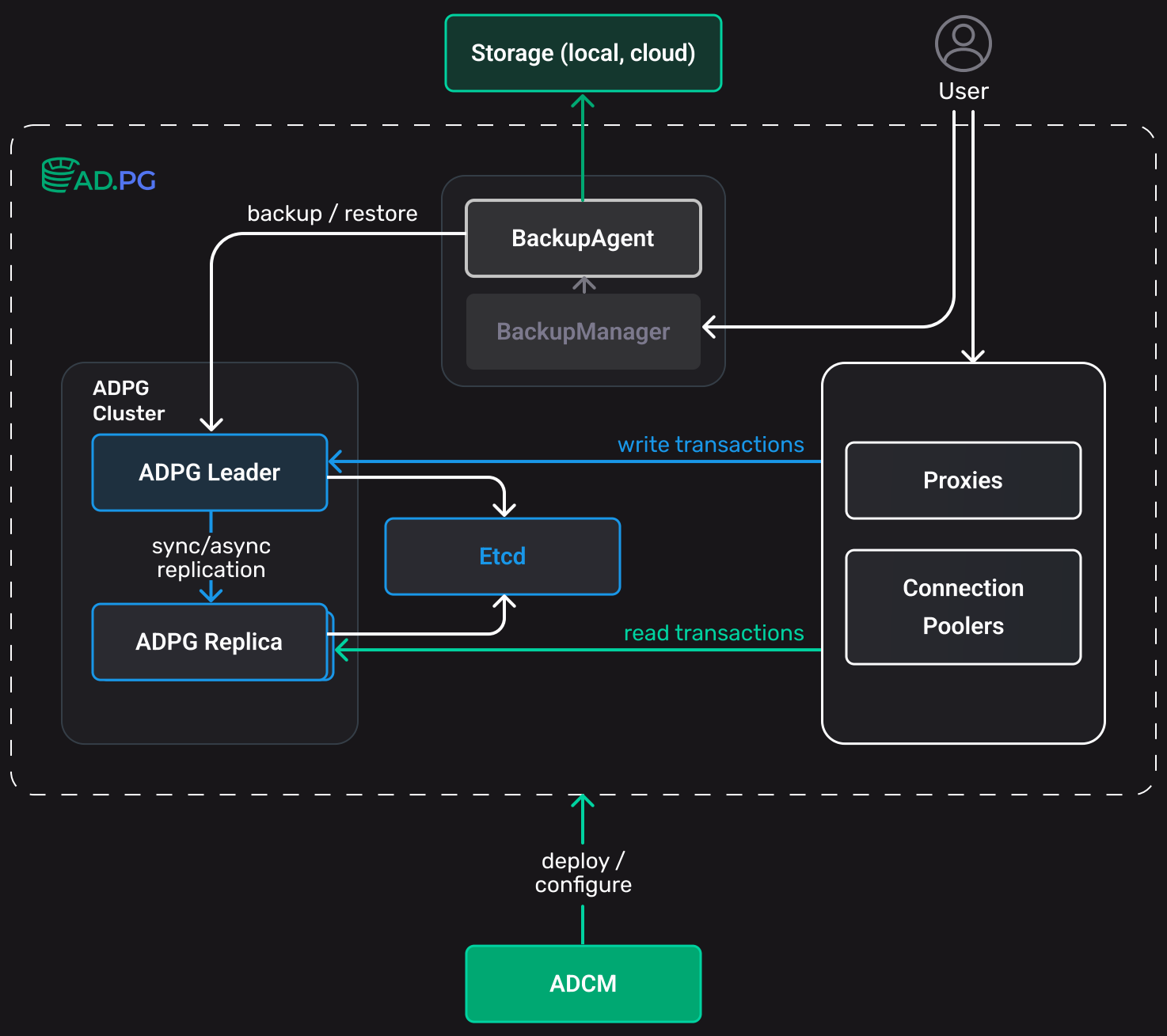
  ADPG также можно использовать в качестве хранилища данных в формате NoSQL.
- **Научные данные**. Исследовательские и научные проекты могут генерировать терабайты данных. ADPG обладает аналитическими возможностями и предлагает мощный механизм SQL, который может легко обрабатывать большие объемы данных. Функциональность ADPG также может быть расширена — можно написать функции агрегации и реализовать собственную бизнес-логику в базе данных.
- **Хранение метрик**. Вы можете установить расширение TimescaleDB и использовать ADPG для хранения данных временных рядов. Например, курсы валют, телеметрия движения транспортных средств, статистика доступа к серверу или нагрузка на ЦП, изменяющиеся во времени.
- **Хранилище для продуктов Arenadata**. Ряд продуктов Arenadata используют ADPG в качестве внутреннего хранилища. Например, ADH хранит метаинформацию в кластере ADPG.

Архитектура ADPG

Евгения Кузина

Назад к содержанию

Arenadata Postgres (ADPG) — СУБД для эффективной работы с данными, имеющими разный профиль нагрузки (преимущественно OLTP — Online Transaction Processing). ADPG позволяет работать с различными объемами данных и поддерживает широкий спектр типов данных, включая JSON и пользовательские типы, а также предоставляет инструментарий для создания механизмов обработки данных.



Архитектура ADPG

Основные компоненты кластера на схеме:

- **BackupAgent** — создание резервных копий и загрузка их в хранилище. В основе BackupAgent лежит утилита по управлению резервным копированием pgBackRest.
- **BackupManager** — управление созданием и хранением резервных копий.
- **ADPG leader** — основной хост, имеющий право выполнять write-транзакции.
- **ADPG replica** — дополнительные хосты, являющиеся репликами leader-хоста.
- **etcd** — распределенное хранилище конфигураций.
- **Proxies** — распределение read/write транзакций между лидером и репликами (HAProxy).
- **Connection Poolers** — экземпляры PgBouncer, установленные на каждой ноде ADPG, которые управляют количеством соединений и их переиспользованием.

ADPG использует Patroni для создания кластера на основе потоковой репликации PostgreSQL и реализации отказоустойчивости (High Availability,HA). Кластер ADPG, поддерживающий HA, должен содержать несколько нод ADPG. Каждая нода содержит сервер PostgreSQL и агент Patroni, обслуживающий локальный экземпляр PostgreSQL. Одна нода ADPG является лидером, остальные — репликами. Лидер обслуживает транзакции на чтение и запись (read-write), если иное не указано в настройках сервиса балансировки, а реплики — только запросы на чтение (read-only). В случае отказа лидера Patroni выбирает нового лидера из реплик, и кластер ADPG продолжает работу.

Patroni сохраняет конфигурацию кластера в распределенном хранилище конфигурации (Distributed Configuration Storage, DCS). ADPG требует наличия кластера etcd, чтобы использовать его в качестве DCS для Patroni.

Для того чтобы реализовать балансировку нагрузки, в ADPG используется HAProxy (High Availability Proxy). HAProxy является программным балансировщиком нагрузки TCP/HTTP. HAProxy слушает порты попарно: подключения к одному из них передаются лидеру, запросы ко второму распределяются по ADPG-нодам. За дополнительной информацией обратитесь к статье Балансировка нагрузки.

PgBouncer управляет пулом соединений ADPG. Приложение может подключаться к PgBouncer как к серверу ADPG. PgBouncer создает соединение с сервером ADPG или повторно использует одно из существующих подключений. PgBouncer предназначен для снижения издержек, связанных с установкой новых соединений. Чтобы не нарушать семантику транзакций, PgBouncer поддерживает несколько типов пула при ротации соединений:

- Пул сессий — при подключении клиента ему присваивается соединение с сервером на все время подключения клиента. Когда клиент отключается, соединение с сервером возвращается в пул. Это метод по умолчанию.
- Пул транзакций — соединение с сервером назначается клиенту только во время транзакции. Когда PgBouncer замечает, что транзакция завершена, соединение с сервером возвращается в пул.
- Пул операторов — соединение с сервером возвращается в пул сразу после завершения запроса. В этом режиме запрещены транзакции с несколькими операторами, поскольку они могут быть прерваны.

Также частью ADPG является хост для хранения резервных копий — **Storage**. Вы можете использовать любой тип хранилища, который поддерживается утилитой pgBackRest.

Управление кластером ADPG выполняется через ADCM. Следующие действия доступны в пользовательском интерфейсе ADCM:

- Кластерные действия, которые позволяют управлять кластером со всеми сервисами.
- Сервисные действия, управляющие определенным сервисом.

Балансировка нагрузки

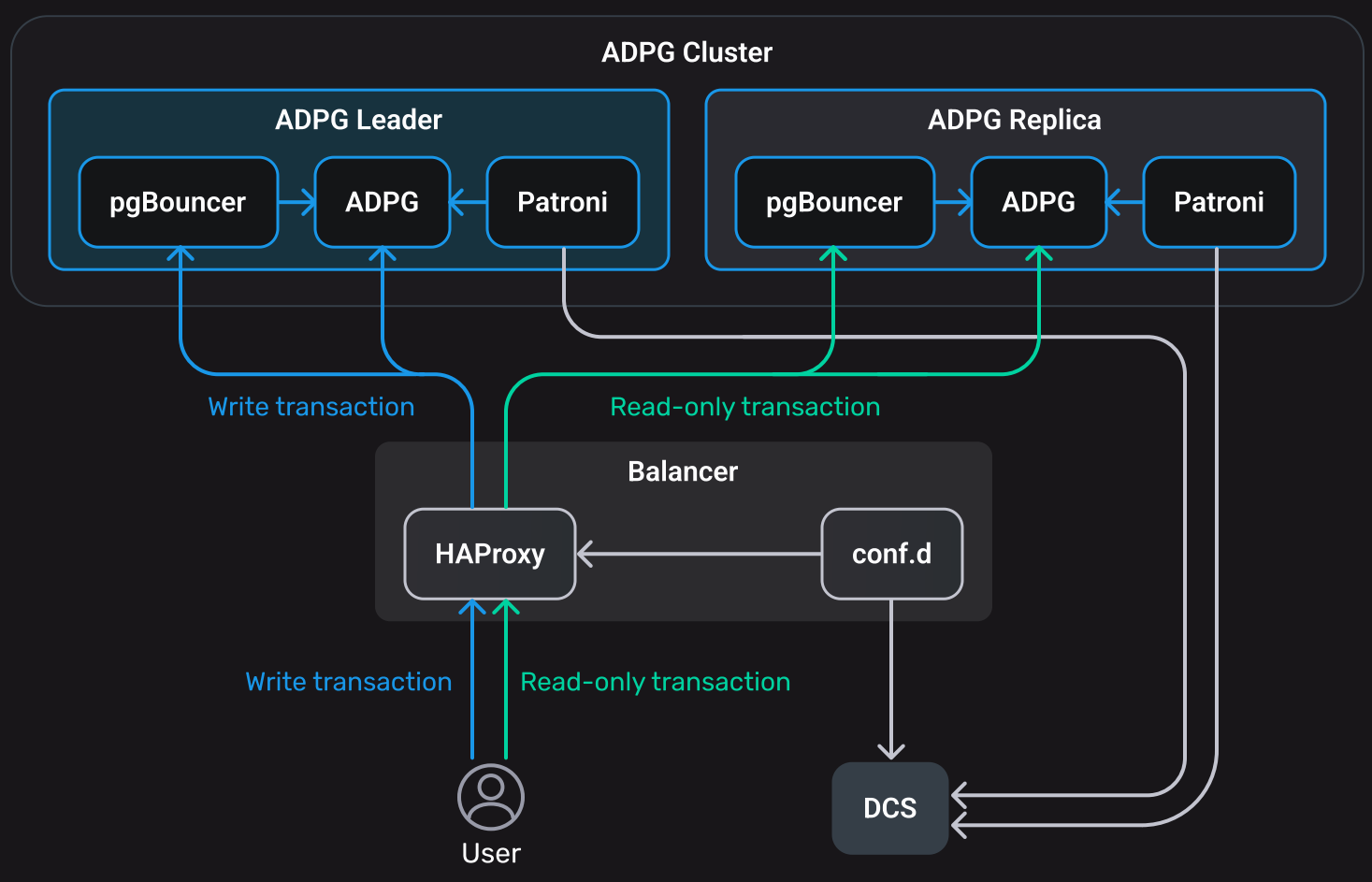
Евгения Кузина

Назад к содержанию

ПРИМЕЧАНИЕ

Балансировка нагрузки доступна в версии ADPG Enterprise.

В кластере ADPG реализована балансировка нагрузки, которая заключается в распределении запросов от пользователей по разным репликам базы данных. Это позволяет выполнять большее количество запросов одновременно.



Балансировка нагрузки в ADPG

Для того чтобы реализовать балансировку нагрузки, в ADPG используется HAProxy (High Availability Proxy). HAProxy является программным балансировщиком нагрузки TCP/HTTP. HAProxy слушает два порта: подключения к одному из них передаются лидеру, запросы ко второму распределяются по ADPG-нодам. Транзакции на запись должны приходить на первый порт, а на чтение на второй. Если транзакция на запись передается на порт для запросов на чтение, ADPG её отклонит.

ПРИМЕЧАНИЕ

HAProxy не определяет, является ли текущая транзакция запросом на запись или только на чтение. Выбирать корректный порт для подключений должны клиентские приложения.

В соответствии с заданными правилами HAProxy выбирает какой ноде передать клиентский запрос. Запросы на запись обрабатываются только нодой-лидером, а на чтение передаются репликам. Также можно настроить HAProxy так, чтобы лидер мог обрабатывать запросы на чтение. Настройка сервиса балансировки описана в статье Настройка балансировки нагрузки.

Если в кластере включена утилита PgBouncer, которая управляет пулом соединений, запросы на запись будут передаваться в экземпляр PgBouncer ноды-лидера ADPG, а запросы на чтение — экземплярам PgBouncer нод-реплик. См. Настройка PgBouncer.

В качестве распределенного хранилища конфигурации (Distributed Configuration Store, DCS) ADPG использует etcd-кластер. Сервис Patroni, реализующий потоковую репликацию, сохраняет информацию о конфигурации ADPG-кластера в Etcd. За дополнительной информацией об организации репликации в ADPG обратитесь к статье Обзор отказоустойчивости ADPG.

Сервис conf.d отвечает за обновление конфигурационных файлов HAProxy при изменении конфигурации кластера (добавлении новой ноды, смене лидера). Сервис conf.d устанавливается на один хост с HAProxy и подписывается на обновления конфигурации кластера ADPG, сохраняемые в Etcd. Когда конфигурация изменяется, conf.d генерирует новые конфигурационные файлы для HAProxy и перезагружает HAProxy, чтобы применить изменения.

HAProxy также использует Patroni REST API для дополнительной проверки — какая нода является лидером, а какая репликой.

РЕКОМЕНДАЦИЯ

Для повышения отказоустойчивости HAProxy можно использовать технологии, не имеющие прямого отношения к ADPG. Например, хосты HAProxy могут быть добавлены как записи A/AAAA с одним FQDN в таблицу соответствия DNS-сервера. Кроме того, утилиту keepalived можно использовать для управления несколькими хостами HAProxy.

Мониторинг

Евгения Кузина

Содержание

Назад к содержанию

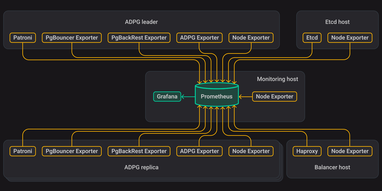
Обзор

Настройки сервиса Monitoring

Обзор

Мониторинг в кластере ADPG реализуется сервисом Monitoring, который разворачивает сервер Prometheus, осуществляющий сбор и хранение метрик. Также сервис Monitoring поддерживает использование Grafana для визуализации данных.

Prometheus извлекает метрики через HTTP-вызовы к определенным конечным точкам. Передача метрик на эти точки осуществляется экспортерами, которые должны быть установлены на хосты кластера.



Мониторинг в ADPG

Таблица ниже описывает компоненты сервиса мониторинга и на какие хосты они должны быть установлены.

Компонент	Описание	Хост
Prometheus	Сервер Prometheus, осуществляющий сбор и хранение метрик	Отдельный хост для мониторинга
Grafana	Веб-интерфейс Grafana, позволяющий создавать дашборды для визуализации метрик	Хост для Grafana. Рекомендуется использовать хост с сервером Prometheus
ADPG Exporter	Передает метрики PostgreSQL	Все хосты с сервисом ADPG
Pgbouncer Exporter	Передает метрики Pgbouncer	Все хосты с сервисом ADPG
PgBackRest Exporter	Передает статистику бэкапов	Все хосты с сервисом ADPG
Node Exporter	Передает системные метрики	Все хосты, с которых необходимо собирать системные метрики

Patroni и Etcd по умолчанию предоставляют конечную точку `/metrics`. Для них нет необходимости настраивать отдельный экспортер, поддерживающий парсинг (scraping) Prometheus. Сервис Balancer (HAProxy) не требует установки отдельного компонента для экспорта метрик, но вы можете настроить параметры встроенного HAProxy-экспортера в настройках сервиса Monitoring.

Настройки сервиса Monitoring

Для указания параметров мониторинга откройте вкладку ***Services*** кластера ADPG и выберите сервис ***Monitoring***. Параметры доступны на вкладке ***Primary configuration***.

HAProxy Exporter settings

Параметр	Описание	Значение по умолчанию
Listen port	Порт на хосте с сервисом Balancer для получения метрик HAProxy в формате Prometheus	8405
Metrics endpoint	Конечная точка на хосте с сервисом Balancer для получения метрик HAProxy в формате Prometheus	/metrics

Pgbouncer Exporter settings

Параметр	Описание	Значение по умолчанию
Listen port	Порт на хосте с сервисом ADPG для получения метрик пула соединений в формате Prometheus	9188

PgBackRest Exporter settings

Параметр	Описание	Значение по умолчанию
Listen port	Порт на хосте с сервисом ADPG для получения метрик PgBackRest в формате Prometheus	9854
Metrics endpoint	Конечная точка на хосте с сервисом ADPG для получения метрик PgBackRest в формате Prometheus	/metrics

ADPG Exporter settings

Параметр	Описание	Значение по умолчанию
Listen port	Порт на хосте с сервисом ADPG для получения метрик PostgreSQL в формате Prometheus	9187

Prometheus settings

Группа	Параметр	Описание	Значение по умолчанию
—	scrape_interval	Интервал сбора метрик	1m
—	scrape_timeout	Время ожидания данных при сборе метрик	10s
—	Password for Grafana connection	Пароль пользователя Grafana (`admprom_grafana`), который используется для подключения к Prometheus. Пароль должен содержать комбинацию заглавных [A-Z] и строчных [a-z] английских букв, цифр [0-9], специальных символов и иметь минимальную длину 8	—
—	Prometheus users to login/logout to Prometheus	Учетные данные пользователей для входа в веб-интерфейс Prometheus	—
Service parameters	config.file	Путь к конфигурационному файлу сервера Prometheus	/etc/admprom/prometheus/prometheus.yml
	storage.tsdb.path	Путь к базе данных сервера Prometheus	/var/lib/admprom/prometheus
	web.console.libraries	Путь к библиотекам управления консолями	/usr/share/admprom/prometheus/console_libraries
	web.console.templates	Путь к шаблонам консолей сервера Prometheus	/usr/share/admprom/prometheus/consoles
	web.config.file	Указывает, какой файл веб-конфигурации следует загрузить. Файл должен иметь YAML-формат	/etc/admprom/prometheus/prometheus-auth.yml
	storage.tsdb.retention.time	Время хранения данных. Поддерживаемые единицы: `y`, `w`, `d`, `h`, `m`, `s`, `ms`	15d
	web.listen-address	Адрес для доступа к веб-интерфейсу Prometheus	0.0.0.0:11200

Grafana settings

Параметр	Описание	Значение по умолчанию
Grafana administrator's password	Пароль пользователя Grafana `admin`. Пароль должен включать заглавные [A-Z] и строчные [a-z] английские буквы, цифры [0-9], специальные символы и иметь минимальную длину 8	—
Grafana listen port	Порт для доступа к web-интерфейсу Grafana	11210

Node Exporter settings

Параметр	Описание	Значение по умолчанию
Listen port	Порт на хосте кластера для получения системных метрик в формате Prometheus	11203
Metrics endpoint	Конечная точка на хосте кластера для получения системных метрик в формате Prometheus	/metrics

Параметры ***Prometheus settings → Password for Grafana connection*** и ***Grafana settings → Grafana administrator's password*** являются обязательными, их необходимо указать на этапе установки мониторинга.

Получить дополнительную информацию по установке сервиса Monitoring можно в статье Установка мониторинга.

Метрики сервиса Monitoring описаны в статье Метрики мониторинга ADPG-кластера.

Управление пулом подключений

Евгения Кузина

Содержание

Назад к содержанию

Как ADPG/PostgreSQL устанавливает соединения

Как работает PgBouncer

Режимы PgBouncer

- Session
- Transaction
- Statement

PostgreSQL имеет ресурсоемкую архитектуру обработки соединений. Каждое входящее соединение создает новый серверный процесс. Это неэффективно, если речь идет о большом количестве подключений в производственном окружении. Когда клиент отправляет запрос к серверу PostgreSQL, создается соединение с базой данных. Пока сервер не ответит клиенту, соединение остается открытым. Под каждое из таких соединений выделяется память на сервере. Если соединения открываются достаточно часто, это может приводить к проблемам. Вот почему необходим PgBouncer. PgBouncer — это программа, управляющая пулом соединений PostgreSQL, которая позволяет увеличивать количество соединений без снижения производительности.

ПРИМЕЧАНИЕ

PgBouncer доступен в версии ADPG Enterprise. См. Настройка PgBouncer.

Как ADPG/PostgreSQL устанавливает соединения

ADPG/PostgreSQL реализует клиент-серверную модель по схеме "один процесс для одного пользователя". В этой модели каждый клиентский процесс подключается к одному серверному процессу. Поскольку изначально не определено, сколько соединений будет установлено, сервер PostgreSQL должен использовать процесс-супервизор, который запускает новый процесс каждый раз, когда запрашивается соединение. Этот процесс-супервизор называется **postmaster** и прослушивает указанный порт TCP/IP на предмет входящих соединений. Всякий раз, когда он обнаруживает запрос на соединение, он запускает новый backend-процесс. Эти серверные процессы взаимодействуют друг с другом и с другими процессами экземпляра сервера (a server instance), используя семафоры и общую память (shared memory) для обеспечения целостности данных при конкурентном доступе к ним. Каждый из этих серверных процессов может расширяться в зависимости от данных, к которым он обращается.

Несмотря на то, что под каждый новый backend-процесс выделяется память, количество доступной памяти не является основным ограничивающим фактором. Более тонкий и важный момент заключается в том, что postmaster и его внутренние процессы совместно используют память для коммуникации, и некоторые части этого общего пространства являются глобальными узкими местами. Например, в PostgreSQL существует структура (`PROC_HDR`), которая отслеживает каждый текущий процесс или транзакцию. Операции, которые происходят в любом backend-процессе, требуют прохождения всего списка процессов или транзакций, сохраненных в этой структуре. Для добавления нового процесса используется эксклюзивная блокировка. Совокупный эффект заключается в том, что производительность любого backend-процесса обратно пропорционально количеству всех активных backend-процессов в системе.

Конфигурационный параметр ADPG/PostgreSQL `max_connections` определяет максимальное количество одновременных подключений к серверу. Значение по умолчанию — `100` . С большой долей вероятности такого количества подключений может быть недостаточно в производственном окружении.

Как работает PgBouncer

PgBouncer — это утилита, которая управляет пулом соединений. Клиентское приложение может подключиться к PgBouncer, как если бы это был сервер ADPG/PostgreSQL.

Когда PgBouncer получает запрос на клиентское соединение, он выполняет аутентификацию от имени сервера PostgreSQL. PgBouncer поддерживает все механизмы аутентификации, которые доступны в PostgreSQL. Важным моментом является то, что PgBouncer кеширует соединения.

Если аутентификация прошла успешно, PgBouncer проверяет наличие кешированного соединения с комбинацией тех же имени пользователя и базы данных. Если кешированное соединение найдено, PgBouncer возвращает соединение клиенту. Если соединение не найдено, PgBouncer создает новое соединение. Эта операция не должна нарушать ограничения, установленные в следующих параметрах: `default_pool_size` , `max_client_conn` , `max_db_connections` и `max_user_connections` . Все эти значения можно определить в настройках PgBouncer. Если создание нового соединения нарушает какие-либо ограничения, PgBouncer ставит соединение в очередь до тех пор, пока не появится возможность открыть новое, если только не превышено значение `max_client_conn` . Превышение значения `max_client_conn` прерывает соединение.

В режиме `transaction` или `statement` операции, следующие за аутентификацией, выполняются только тогда, когда клиент начинает выполнять транзакцию или оператор соответственно. Режимы пула PgBouncer описаны ниже.

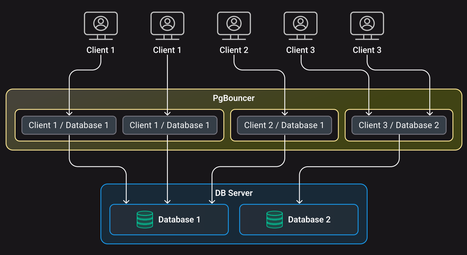


Схема работы PgBouncer

Прежде чем вернуть соединение в пул, PgBouncer выполняет запрос сброса, чтобы удалить всю информацию о сессии. Такой подход делает безопасным совместное использование соединений клиентами.

PgBouncer может помочь оптимизировать:

- сетевой трафик;
- накладные расходы на запуск нового серверного процесса и аутентификацию.

PgBouncer не поддерживает автоматическую балансировку нагрузки или режим высокой отказоустойчивости, но ADPG Enterprise Edition предоставляет эти функции. См. Балансировка нагрузки.

PgBouncer имеет следующие преимущества:

- Не требует изменения кода на стороне клиента.
- Предоставляет сквозную аутентификацию. PgBouncer может безопасно аутентифицировать пользователя, не имея доступа к его паролям. Это делает PgBouncer более безопасным и простым в обслуживании — не нужно обновлять PgBouncer каждый раз, когда пользователь обновляет свой пароль.
- Представляет собой одиночный процесс, и все команды клиента и ответы сервера проходят через него без какой-либо обработки. Следовательно, он занимает небольшой объем памяти.
- Может значительно улучшить количество транзакций в секунду, которые обрабатывает сервер, и хорошо масштабируется для многочисленных клиентов.

Режимы PgBouncer

Режимы PgBouncer позволяют пользователю решать, когда соединение должно быть возвращено в пул. Доступны следующие режимы:

- session
- transaction
- statement

Установить нужный режим можно на вкладке ***Primary Configuration*** сервиса ***ADPG***. См. Настройка PgBouncer.

Session

В режиме пула `session` соединение возвращается в пул, когда клиент завершает сессию. Это режим по умолчанию. Его использование сокращает время и ресурсы, необходимые для установления соединений и аутентификации пользователей.

Transaction

В режиме `transaction` соединение возвращается в пул, когда клиент завершает транзакцию. Нет никакой гарантии, что две транзакции, выполняемые на одном и том же клиентском соединении PgBouncer, будут выполняться на одном и том же серверном соединении PgBouncer. Соединение с сервером назначается клиентскому соединению всякий раз, когда клиент начинает новую транзакцию, и освобождается в пул, когда клиент завершает транзакцию, например, с использованием `ROLLBACK` или `COMMIT` , а не тогда, когда клиент отключается. Обратите внимание, что в PostgreSQL если транзакция не запускается явно, каждый запрос выполняется в виде индивидуальной транзакции. См. Transactions.

Режим `transaction` обеспечивает лучшую производительность и позволяет сократить количество простаивающих транзакций, в дополнение к сокращению времени, затрачиваемого на установление соединений и аутентификацию пользователей.

В этом режиме необходимо быть осторожным при использовании последовательностей, рекомендательных блокировок и `%_SHARED` в функциях PL/Perl. Кроме того, планы запросов, созданные операторами PREPARE, кешируются и существуют вне транзакции. Их использование в режиме `transaction` может привести к ошибкам или неожиданным результатам.

Полный список функционала PostgreSQL, основанного на специфике сессий, который не поддерживается в режиме `transaction` :

- SET / RESET;
- LISTEN;
- опция `WITH HOLD CURSOR` команды DECLARE;
- PREPARE / DEALLOCATE;
- опции `ON COMMIT PRESERVE ROWS` и `ON COMMIT DELETE ROWS` команды CREATE TABLE для временных таблиц;
- LOAD;
- рекомендательные блокировки уровня сессии.

За дополнительной информацией обратитесь к статье PgBouncer features.

Statement

В режиме `statement` соединение возвращается в пул сразу после выполнения оператора. Это самый агрессивный режим. Его можно рассматривать как пул транзакций с одним ограничением: транзакции с несколькими операторами не допускаются. Режим `statement` предназначен для обеспечения режима автоматической фиксации на клиенте, в первую очередь ориентированного на PL/Proxy.

Требования к оборудованию

Евгения Кузина

Содержание

Назад к содержанию

ADPG CE

ADPG EE

Arenadata Enterprise Tools

Arenadata Monitoring

ВАЖНО

Системные требования, приведенные ниже, являются **минимальными**. Целевой сайзинг необходимо рассчитывать исходя из требований вашей организации.

ADPG CE

Сервис ADPG (экземпляр PostgreSQL)

- CPU: 2 ядра
- RAM: 4 ГБ
- Диск: 30 ГБ

Сервис Monitoring

- CPU: 2 ядра
- RAM: 2 ГБ
- Диск: 20 ГБ

ADPG EE

Сервис ADPG (PostgreSQL с Patroni и PgBouncer)

- CPU: 4 ядра
- RAM: 8 ГБ
- Диск: 30 ГБ

Сервис Monitoring

- CPU: 2 ядра
- RAM: 2 ГБ
- Диск: 20 ГБ

Сервис Balancer

- CPU: 1 ядро
- RAM: 1 ГБ
- Диск: 5 ГБ

Сервис Etcd

- CPU: 2 ядра
- RAM: 2 ГБ
- Диск: 10 ГБ

Arenadata Enterprise Tools

Если вместе с вашим продуктом вы используете ET, примите во внимание следующие рекомендации:

- Виртуальный сервер.
- CPU: от 4 ядер.
- RAM: от 8 ГБ.
- Диск: от 50 ГБ.

Arenadata Monitoring

Если вместе с вашим продуктом вы используете ADM, примите во внимание следующие рекомендации:

- Виртуальный сервер.
- CPU: от 8 ядер.
- RAM: от 16 ГБ.
- Диск: от 10 ГБ на каждый отслеживаемый хост.

Требования к сети

Евгения Кузина

Назад к содержанию

- Требуется одна или несколько выделенных и обособленных одноранговых Interconnect-сетей под цели внутренней коммуникации кластера.
- К Interconnect-сетям должны быть подключены все серверы кластера.
- Все серверы должны иметь полные доменные имена (FQDN) и возможность обмениваться данными друг с другом на заданных скоростях.
- Должен быть известен IP-адрес каждого сервера.
- Должен быть обеспечен Ping-доступ к любому серверу (ICMP).
- В случае инсталляции ADPG с доступом в Интернет требуется доступ к развернутому на выделенном сервере Arenadata Cluster Manager. Установка Arenadata Cluster Manager описана в статье Установка ADCM.
- В случае инсталляции ADPG без доступа в Интернет (offline) требуется доступ к развернутому на выделенном сервере Arenadata Enterprise Tools. Установка Enterprise Tools описана в разделе Установка кластера Enterprise Tools.
- Хосты ADPG должны преобразовывать короткие имена хостов в IP-адреса для подключения друг к другу. Если в вашей сети нет службы DNS для разрешения имен хостов, вы можете добавить короткое и полное доменное имя для каждого IP-адреса в файл hosts на всех хостах ADPG.
- Перечисленные ниже порты должны быть открыты на хостах, на которых устанавливается указанный сервис.

ADPG

Сервис	Порт	Назначение
Общий	11203	Порт на хостах кластера для получения системных метрик в формате Prometheus
ADPG	5432	Порт по умолчанию, который прослушивает сервис ADPG (PostgreSQL)
	15432	Порт по умолчанию, который прослушивает PgBouncer
	8008	Порт Patroni REST API — порт на хосте с сервисом ADPG для получения метрик Patroni в формате Prometheus
	9187	Порт на хосте с сервисом ADPG для получения метрик PostgreSQL в формате Prometheus
	9188	Порт на хосте с сервисом ADPG для получения метрик пула соединений (PgBouncer) в формате Prometheus
	9854	Порт на хосте с сервисом ADPG для получения метрик PgBackRest в формате Prometheus
Balancer	6432	Порт на хосте с компонентом HAProxy, на который должны приходить транзакции на запись
	6433	Порт на хосте с компонентом HAProxy, на который должны приходить транзакции на чтение
	7000	Порт, на котором доступна веб-страница с отчетом статистики HAProxy
	16432	Порт, все подключения к которому передаются экземпляру PgBouncer ноды-лидера ADPG
	16433	Порт, все подключения к которому передаются экземплярам PgBouncer нод-реплик ADPG
	8405	Порт на хосте с сервисом Balancer для получения метрик HAProxy в формате Prometheus
Etcd	2380	Порт для связи между серверами Etcd
	2379	Порт для клиентских запросов — порт на хосте с сервисом Etcd для получения метрик PgBackRest в формате Prometheus
Monitoring	11200	Порт доступа к web-интерфейсу Prometheus
	11210	Порт доступа к web-интерфейсу Grafana

Enterprise Tools

Порт	Назначение
443	TCP-порт, используемый Arenadata Enterprise Tools (служба реестра Docker)
81	TCP-порт Arenadata Enterprise Tools (доступ к репозиториям в случае offline-установки)

ADCM

Порт	Назначение
8000	TCP-порт для отправки статусов компонентов кластера в ADCM

Программные требования

Евгения Кузина

Содержание

Назад к содержанию

Операционная система

ADCM

SELinux

Антивирус

Сетевой маршрут по умолчанию

На этой странице представлена информация о требованиях и рекомендациях перед установкой ADPG.

Операционная система

ВАЖНО

- **CentOS 7.9.** В связи с тем, что жизненный цикл CentOS 7 завершился 30 июня 2024, настоятельно рекомендуется использовать Ubuntu в качестве свободного программного обеспечения с открытым исходным кодом. Поддержка CentOS 7 в ADPG оставлена для совместимости с сервисом ADPG, который используется в других продуктах Arenadata, и будет прекращена в релизах, выходящих после 01 января 2026. ADPG v16.3.5 — последний релиз, поддерживающий CentOS 7, также ADPG v16.3.5 на CentOS 7 несовместим с ADPG Control, так как ADPG Control не может быть установлен на CentOS 7.
- **Ubuntu 22.04** заменяется на Ubuntu 24.04 в релизах после 1 апреля 2026 года. Рассмотрите возможность резервного копирования/восстановления кластера ADPG вместе с обновлением ОС, если это запланировано.
- **Alt Linux 8.4 SP** заменяется на Alt Linux 10 SP в релизах после 1 января 2026 года. Рассмотрите возможность резервного копирования/восстановления кластера ADPG вместе с обновлением ОС, если это запланировано.

ADPG поддерживает только архитектуру x86_64 и следующие операционные системы:

- CentOS 7.9 (x86_64)
- RHEL 7.9 (x86_64)
- Ubuntu 22.04 (x86_64)
- Alt Linux 8.4 SP (x86_64) в Enterprise-версии ADPG
- Astra Linux 1.7 SE "Орел", "Воронеж" (x86_64) в Enterprise-версии ADPG
- РЕД ОС 7.3 Сертифицированная редакция в Enterprise-версии ADPG

Вам необходимо подготовить все зависимые репозитории дополнительно к выбранной операционной системе.

ADCM

Необходимо использовать ADCM 2.6.0 или более позднюю версию. При попытке загрузить бандл ADPG в ADCM неподдерживаемой версии возникнет ошибка: `Map key "flag_autogeneration" is not allowed here (rule "cluster_object")`.

SELinux

SELinux должен быть выключен. Полное руководство по этой операции вы можете найти в разделе Установка ADCM.

Антивирус

Рекомендуется отключать антивирусные программы перед установкой и в течение процесса эксплуатации ADPG. Если антивирус все же используется, пожалуйста, уточните настройки, необходимые для использования ADPG, у поставщика вашего антивирусного ПО.

Сетевой маршрут по умолчанию

Убедитесь, что таблицы маршрутизации на каждом хосте содержат маршрут по умолчанию — `0.0.0.0/0`. За дополнительной информацией обратитесь к статье Подготовка хостов.

Установка

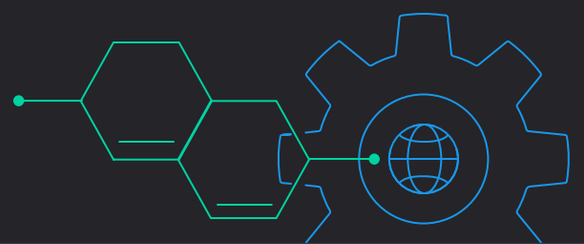
Назад к содержанию

Этот раздел содержит описание установки кластера Arenadata Postgres (ADPG). Для установки рекомендуется использовать Arenadata Cluster Manager (ADCM) — отдельный продукт Arenadata, предназначенный для простого, удобного и быстрого развертывания и дальнейшей эксплуатации программного обеспечения.

Возможны два способа установки ADPG через ADCM.

Online-установка

на хостах с доступом к Интернету



Offline-установка

на хостах без доступа к Интернету

Online-установка

Назад к содержанию

Этот раздел посвящен **online**-установке ADPG. Данный способ установки подходит для хостов с доступом к Интернету.

Процесс online-установки через ADCM включает следующие шаги.

Шаг 1.

Установка ADCM

Шаг 2.

Подготовка хостов

Шаг 3.

Установка кластера ADPG

Шаг 4.

Установка мониторинга

Шаг 5.

Установка ADPG Control

Установка ASCM

Содержание

Часть 1. Предварительные требования
Требования к аппаратному обеспечению
Требования к операционной системе
Установка Docker
Установка Docker Compose
Настройка доступа к Docker без прав root
Часть 2. Установка ASCM
Установка ASCM
Часть 3. Запуск и проверка установки
Проверка статуса контейнеров Docker
Проверка доступности портов
Часть 4. Обновление ASCM

Установка ASCM осуществляется с помощью Docker и Docker Compose. Ниже приведена пошаговая инструкция для развертывания.

Часть 1. Предварительные требования

Требования к аппаратному обеспечению

Минимальные требования к аппаратному обеспечению для установки ASCM:

- Архитектура: x86_64
- CPU: 4 ядра
- RAM: 8 ГБ
- Диск: 40 ГБ

Рекомендуемые требования к аппаратному обеспечению для установки ASCM:

- Архитектура: x86_64
- CPU: 8 ядер
- RAM: 16 ГБ
- Диск: 100 ГБ

Требования к операционной системе

Поддерживаемые операционные системы для установки ASCM:

Операционная система	Версия
CentOS	7.3 или выше
Red Hat Enterprise Linux	7.3 или выше
Ubuntu	16.04 или выше

Установка Docker

Описание

Docker — это платформа для разработки, доставки и запуска контейнерных приложений. Docker позволяет быстро доставлять ваш код, тестировать, развертывать и управлять инфраструктурой вашего приложения.

Команда Linux 1

```
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
```

Описание команды

- curl: утилита командной строки для передачи данных с использованием различных протоколов.
- -f: (--fail) указывает curl не выводить HTML-страницу при ошибке HTTP.
- -s: (--silent) режим тишины, curl не будет выводить прогресс и сообщения об ошибках.
- -S: (--show-error) показывать ошибки, даже если используется режим тишины (-s).
- -L: (--location) если сервер возвращает перенаправление, curl будет следовать за ним.
- https://get.docker.com: официальный скрипт установки Docker.
- | bash -s docker --mirror Aliyun: передает скрипт в bash и выполняет его с аргументами docker --mirror Aliyun, которые указывают использовать зеркало Aliyun для установки Docker.

Команда Linux 2

```
systemctl enable docker; systemctl start docker
```

Описание команды

- systemctl enable docker: включает автозапуск Docker при загрузке системы.
- systemctl start docker: запускает службу Docker.

Команда Linux 3

```
docker version
```

Описание команды

- docker version: выводит информацию о версии Docker, включая версию клиента и сервера.

Пример вывода

```
Client: Docker Engine - Community
 Version:           20.10.12
 API version:       1.41
 Go version:        go1.16.12
 Git commit:        e91ed57
 Built:             Mon Dec 13 11:45:41 2021
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.12
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.12
  Git commit:       459d0df
  Built:            Mon Dec 13 11:43:56 2021
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.4.12
  GitCommit:        7b11cfaabd73bb80907dd23182b9347b4245eb5d
 runc:
  Version:          1.0.2
  GitCommit:        v1.0.2-0-g52b36a2
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```

Установка Docker Compose

Описание

Docker Compose — это инструмент для определения и запуска многоконтейнерных приложений Docker. С помощью Compose вы используете файл YAML для настройки служб вашего приложения. Затем с помощью одной команды вы создаете и запускаете все службы из своей конфигурации.

Команда Linux 1

```
curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Описание команды

- curl: утилита командной строки для передачи данных с использованием различных протоколов.
- -L: (--location) если сервер возвращает перенаправление, curl будет следовать за ним.
- "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)": URL для загрузки Docker Compose. $(uname -s) и $(uname -m) динамически подставляют имя операционной системы и архитектуру.
- -o /usr/local/bin/docker-compose: сохраняет загруженный файл в /usr/local/bin/docker-compose.

Команда Linux 2

```
chmod +x /usr/local/bin/docker-compose
```

Описание команды

- chmod +x: делает файл исполняемым.
- /usr/local/bin/docker-compose: путь к файлу Docker Compose.

Команда Linux 3

```
docker-compose version
```

Описание команды

- docker-compose version: выводит информацию о версии Docker Compose.

Пример вывода

```
docker-compose version 1.29.2, build 5becea4c
docker-py version: 5.0.0
CPython version: 3.7.10
OpenSSL version: OpenSSL 1.1.0l  10 Sep 2019
```

Настройка доступа к Docker без прав root

Описание

По умолчанию команда docker может быть выполнена только пользователем root или пользователем, входящим в группу docker, которая автоматически создается Docker. Если вы хотите использовать Docker CLI без префикса sudo, добавьте своего пользователя в группу docker.

Команда Linux 1

```
sudo usermod -aG docker $USER
```

Описание команды

- sudo: выполняет команду с привилегиями суперпользователя.
- usermod: изменяет учетную запись пользователя.
- -aG docker: добавляет пользователя в группу docker.
- $USER: переменная окружения, содержащая имя текущего пользователя.

Команда Linux 2

```
newgrp docker
```

Описание команды

- newgrp docker: изменяет текущую группу пользователя на docker.

Часть 2. Установка ASCM

Установка ASCM

Описание

Установка ASCM осуществляется с помощью скрипта установки, который автоматически загружает и настраивает все необходимые компоненты.

Команда Linux 1

```
curl -sfL https://www.antrea.io/ascm/hack/install.sh | bash
```

Описание команды

- curl: утилита командной строки для передачи данных с использованием различных протоколов.
- -s: (--silent) режим тишины, curl не будет выводить прогресс и сообщения об ошибках.
- -f: (--fail) указывает curl не выводить HTML-страницу при ошибке HTTP.
- -L: (--location) если сервер возвращает перенаправление, curl будет следовать за ним.
- https://www.antrea.io/ascm/hack/install.sh: URL скрипта установки ASCM.
- | bash: передает скрипт в bash и выполняет его.

Команда Linux 2

```
kubectl get pods -n ascm-system
```

Описание команды

- kubectl: утилита командной строки для управления кластерами Kubernetes.
- get pods: команда для получения списка подов.
- -n ascm-system: указывает пространство имен ascm-system, в котором развернуты компоненты ASCM.

Пример вывода

```
NAME                                  READY   STATUS    RESTARTS   AGE
ascm-controller-manager-7d4b6f8c9-2xkqp   2/2     Running   0          5m
ascm-server-6c8f9d7b5-9wlmn               1/1     Running   0          5m
```

Часть 3. Запуск ASCM

Проверка статуса контейнеров Docker

Описание

Проверка статуса контейнеров Docker позволяет убедиться, что все компоненты ASCM запущены и работают корректно.

Команда Linux 1

```
docker ps -a
```

Описание команды

- docker ps: команда для отображения списка контейнеров.
- -a: (--all) показывать все контейнеры, включая остановленные.

Пример вывода

```
CONTAINER ID   IMAGE                                  COMMAND                  CREATED         STATUS         PORTS     NAMES
a1b2c3d4e5f6   ascm/ascm-controller:latest            "/manager"               5 minutes ago   Up 5 minutes             ascm-controller-manager
b2c3d4e5f6a1   ascm/ascm-server:latest                "/server"                5 minutes ago   Up 5 minutes             ascm-server
c3d4e5f6a1b2   mysql:5.7                              "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes             ascm-mysql
```

Проверка доступности портов

Описание

Проверка доступности портов позволяет убедиться, что сервисы ASCM доступны по сети.

Команда Linux 1

```
netstat -tlnp | grep -E '8080|8443|3306'
```

Описание команды

- netstat: утилита командной строки для отображения сетевых соединений.
- -t: (--tcp) показывать только TCP-соединения.
- -l: (--listening) показывать только прослушивающие порты.
- -n: (--numeric) показывать адреса и номера портов в числовом формате.
- -p: (--program) показывать PID и имя программы, использующей порт.
- | grep -E '8080|8443|3306': фильтрует вывод, оставляя только строки с портами 8080, 8443 и 3306.

Пример вывода

```
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      1234/ascm-server
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      1234/ascm-server
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      5678/mysqld
```

Часть 4. Проверка установки

Проверка статуса Docker-контейнеров

Описание

Проверка статуса Docker-контейнеров позволяет убедиться, что все компоненты ASCM запущены и работают корректно.

Команда Linux 1

```
docker ps -a
```

Описание команды

- docker ps: команда для отображения списка контейнеров.
- -a: (--all) показывать все контейнеры, включая остановленные.

Пример вывода

```
CONTAINER ID   IMAGE                                  COMMAND                  CREATED         STATUS         PORTS     NAMES
a1b2c3d4e5f6   ascm/ascm-controller:latest            "/manager"               5 minutes ago   Up 5 minutes             ascm-controller-manager
b2c3d4e5f6a1   ascm/ascm-server:latest                "/server"                5 minutes ago   Up 5 minutes             ascm-server
c3d4e5f6a1b2   mysql:5.7                              "docker-entrypoint.s…"   5 minutes ago   Up 5 minutes             ascm-mysql
```

Проверка доступности портов

Описание

Проверка доступности портов позволяет убедиться, что сервисы ASCM доступны по сети.

Команда Linux 1

```
netstat -tlnp | grep -E '8080|8443|3306'
```

Описание команды

- netstat: утилита командной строки для отображения сетевых соединений.
- -t: (--tcp) показывать только TCP-соединения.
- -l: (--listening) показывать только прослушивающие порты.
- -n: (--numeric) показывать адреса и номера портов в числовом формате.
- -p: (--program) показывать PID и имя программы, использующей порт.
- | grep -E '8080|8443|3306': фильтрует вывод, оставляя только строки с портами 8080, 8443 и 3306.

Пример вывода

```
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      1234/ascm-server
tcp        0      0 0.0.0.0:8443            0.0.0.0:*               LISTEN      1234/ascm-server
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      5678/mysqld
```

Установка ASCM

Описание

Установка ASCM осуществляется с помощью скрипта установки, который автоматически загружает и настраивает все необходимые компоненты.

Команда Linux 1

```
curl -sfL https://www.antrea.io/ascm/hack/install.sh | bash
```

Описание команды

- curl: утилита командной строки для передачи данных с использованием различных протоколов.
- -s: (--silent) режим тишины, curl не будет выводить прогресс и сообщения об ошибках.
- -f: (--fail) указывает curl не выводить HTML-страницу при ошибке HTTP.
- -L: (--location) если сервер возвращает перенаправление, curl будет следовать за ним.
- https://www.antrea.io/ascm/hack/install.sh: URL скрипта установки ASCM.
- | bash: передает скрипт в bash и выполняет его.

Команда Linux 2

```
kubectl get pods -n ascm-system
```

Описание команды

- kubectl: утилита командной строки для управления кластерами Kubernetes.
- get pods: команда для получения списка подов.
- -n ascm-system: указывает пространство имен ascm-system, в котором развернуты компоненты ASCM.

Пример вывода

```
NAME                                  READY   STATUS    RESTARTS   AGE
ascm-controller-manager-7d4b6f8c9-2xkqp   2/2     Running   0          5m
ascm-server-6c8f9d7b5-9wlmn               1/1     Running   0          5m
```

Часть 5. Обновление ASCM

Обновление ASCM

Описание

Обновление ASCM осуществляется с помощью скрипта установки, который автоматически загружает и настраивает все необходимые компоненты.

Команда Linux 1

```
curl -sfL https://www.antrea.io/ascm/hack/install.sh | bash
```

Описание команды

- curl: утилита командной строки для передачи данных с использованием различных протоколов.
- -s: (--silent) режим тишины, curl не будет выводить прогресс и сообщения об ошибках.
- -f: (--fail) указывает curl не выводить HTML-страницу при ошибке HTTP.
- -L: (--location) если сервер возвращает перенаправление, curl будет следовать за ним.
- https://www.antrea.io/ascm/hack/install.sh: URL скрипта установки ASCM.
- | bash: передает скрипт в bash и выполняет его.

Команда Linux 2

```
kubectl get pods -n ascm-system
```

Описание команды

- kubectl: утилита командной строки для управления кластерами Kubernetes.
- get pods: команда для получения списка подов.
- -n ascm-system: указывает пространство имен ascm-system, в котором развернуты компоненты ASCM.

Пример вывода

```
NAME                                  READY   STATUS    RESTARTS   AGE
ascm-controller-manager-7d4b6f8c9-2xkqp   2/2     Running   0          5m
ascm-server-6c8f9d7b5-9wlmn               1/1     Running   0          5m
```

Подготовка хостов

Назад к содержанию

Перед установкой нового кластера необходимо добавить информацию о его будущих хостах в ADCM. Это задача **хостпровайдеров** — специальных плагинов, отвечающих за взаимодействие между хостами и ADCM.

ADCM поддерживает несколько типов хостпровайдеров. Для получения подробной информации о том, как их настроить и использовать для создания хостов, обратитесь к документации:

- SSH
- Yandex Cloud
- VCloud (VMware)
- Cloud.ru Advanced
- CROC Cloud

ВАЖНО

Запрещено использовать зарезервированные слова `adpg`, `balancer`, `chrony`, `etcd` и `prometheus_monitoring` в качестве имен хостов кластера ADPG.

Убедитесь, что таблицы маршрутизации на каждом хосте содержат маршрут по умолчанию — `0.0.0.0/0`.

Используйте команду `route`, чтобы это проверить:

```
$ route
```

Вывод команды должен быть похож на следующий:

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         gateway         0.0.0.0         UG    100    0        0 eth0
10.92.40.0      0.0.0.0         255.255.252.0   U     100    0        0 eth0
```

Если маршрут по умолчанию `0.0.0.0/0` отсутствует, добавьте его с помощью следующей команды:

```
$ sudo route add default gw 0.0.0.0 eth0
```

Создание кластера

Евгения Кузина, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Загрузка бандла

Шаг 2. Загрузка бандла в ADCM

Шаг 3. Создание кластера на базе загруженного бандла

В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

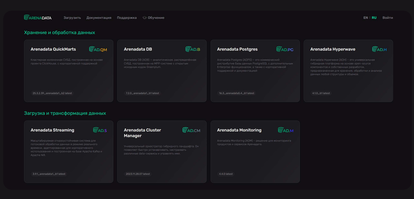
Шаги для добавления нового кластера приведены ниже.

ПРИМЕЧАНИЕ

При использовании Enterprise-версии продукта следует пропустить шаг 1, получить бандл у команды поддержки Arenadata и перейти к шагу 2.

Шаг 1. Загрузка бандла

1. Перейдите на сайт https://network.arenadata.io/ и выберите продукт ***Arenadata Postgres***.



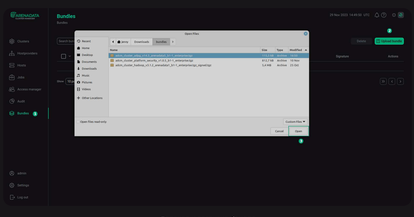
Переход к загрузке дистрибутивов

2. На следующей странице загрузите бандл для нужной версии ADPG.

Шаг 2. Загрузка бандла в ADCM

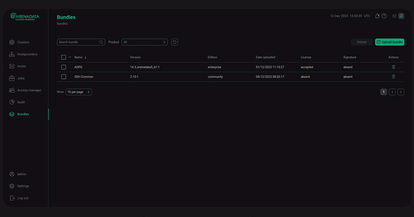
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

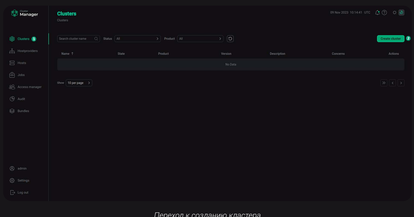
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

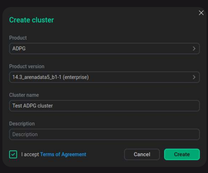
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



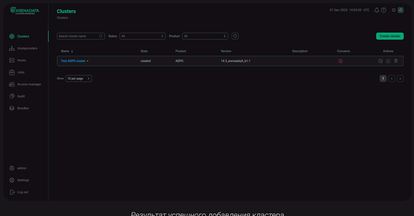
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

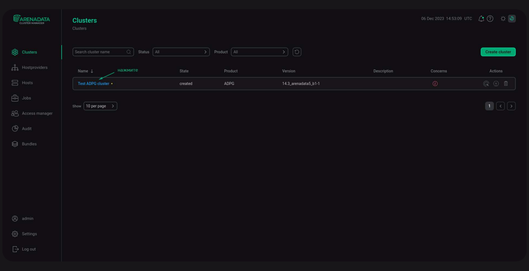
Добавление сервисов

Дарья Барышева, Евгения Кузина

Назад к содержанию

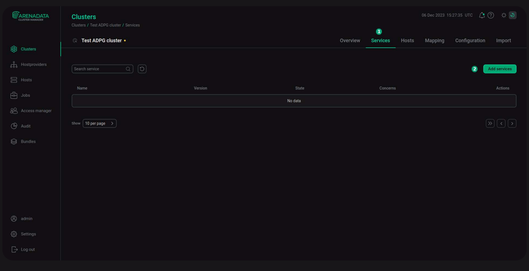
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Примеры сервисов в кластерах ADPG: ADPG, Chrony и так далее. Для добавления сервисов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



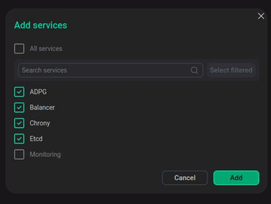
Выбор кластера

2. Откройте вкладку ***Services*** на странице кластера и нажмите ***Add service***.



Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



Выбор сервисов

ВАЖНО

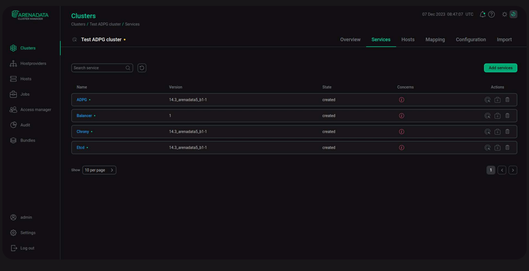
Версия ADPG Enterprise требует обязательного наличия Etcd, встроенного Etcd-сервиса или внешнего кластера, чтобы использовать его как распределенное хранилище конфигурации Patroni (Distributed Configuration Store, DCS). Настройки Etcd должны быть указаны во время установки, после установки их изменить нельзя.

Краткое описание доступных сервисов приведено ниже. CE — Community-версия ADPG (Community Edition), EE — Enterprise-версия ADPG (Enterprise Edition).

Сервисы, которые могут быть добавлены в кластер ADPG

Сервис	Обязательность	Версия	Назначение
ADPG	Да	CE/EE	Основной сервис, соответствующий объектно-реляционной СУБД PostgreSQL
ADPG Control Agents	Нет	EE	Сервис, позволяющий управлять кластером ADPG с помощью ADPG Control
Balancer	Нет	EE	В основе сервиса лежит HAProxy, который реализует балансировку нагрузки
Chrony	Нет	CE/EE	Сервис времени. Обеспечивает синхронизацию часов на хостах кластера, что необходимо для его корректной работы. Установка Chrony не требуется, если на хостах уже используется аналогичный сервис
Etcd	Нет	EE	Версия ADPG Enterprise требует Etcd, чтобы задействовать его как распределенное хранилище конфигурации Patroni (Distributed Configuration Store, DCS). Вы можете использовать встроенный ADPG-сервис Etcd или внешний кластер etcd. Если вы хотите использовать внешний кластер, не добавляйте встроенный Etcd-сервис на вкладке ***Services***. Вместо этого укажите внешние настройки etcd на вкладке ***Primary configuration*** сервиса ADPG. Чтобы их задать, откройте ***Clusters*** → ***ADPG cluster*** → ***Services*** → ***ADPG*** → ***Primary configuration***, как описано в статье Настройка сервисов
Monitoring	Нет	CE/EE	Сервис для сбора и хранения метрик мониторинга кластера ADPG на основе системы мониторинга Prometheus и сервиса визуализации метрик Grafana

4. В результате успешно добавленные сервисы отображаются на вкладке *Services*.



Результат успешного добавления сервисов в кластер

ПРИМЕЧАНИЕ

Опциональные сервисы могут быть добавлены в кластер позднее. Процесс добавления сервисов в уже развернутый кластер не отличается от описанного выше.

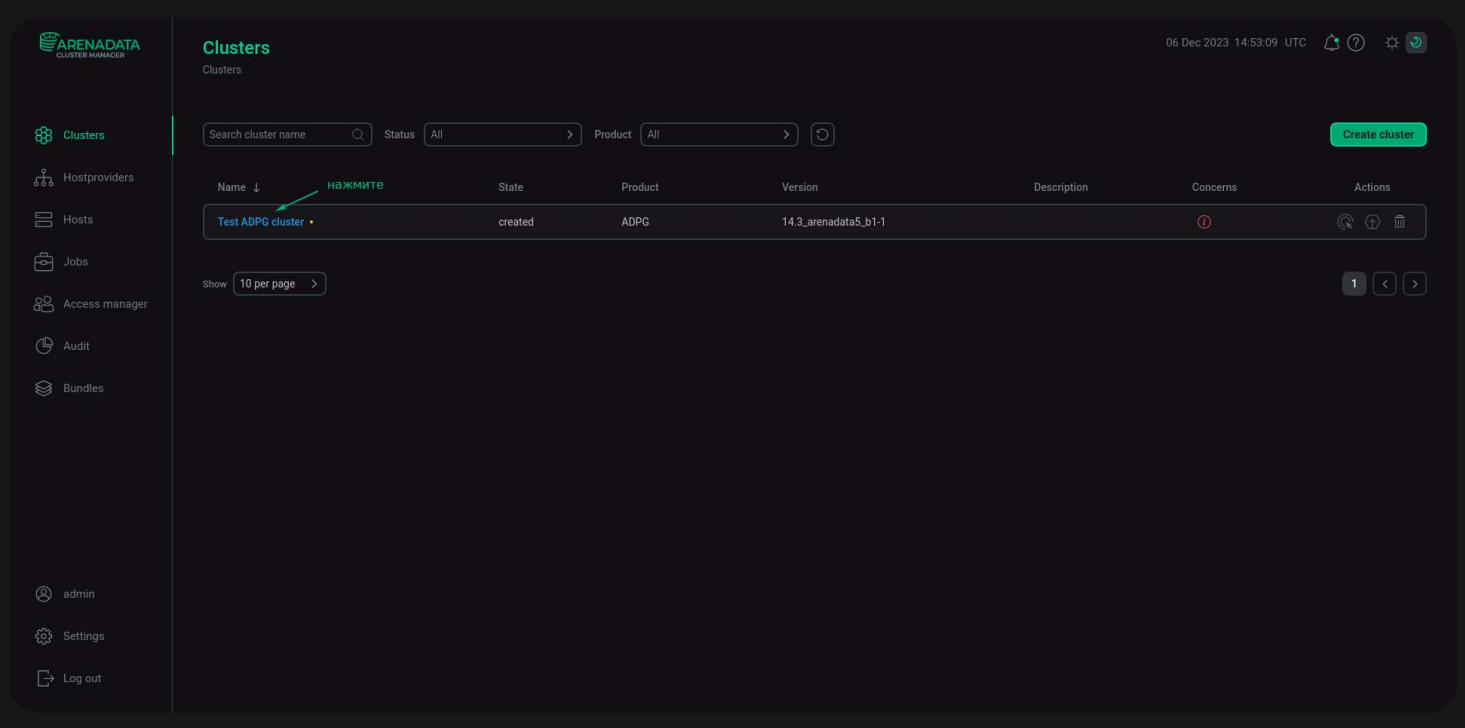
Добавление хостов в кластер

Евгения Кузина, Дарья Барышева

Назад к содержанию

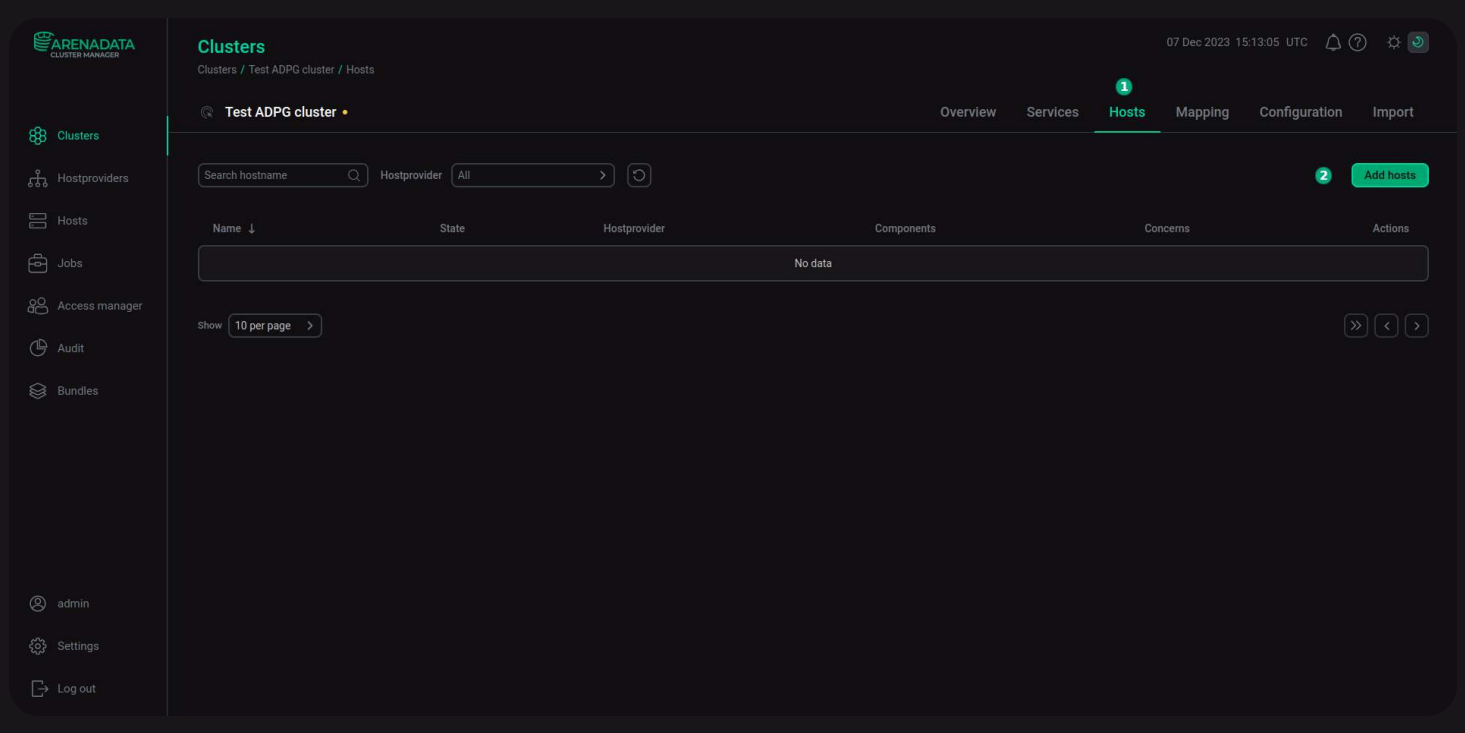
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



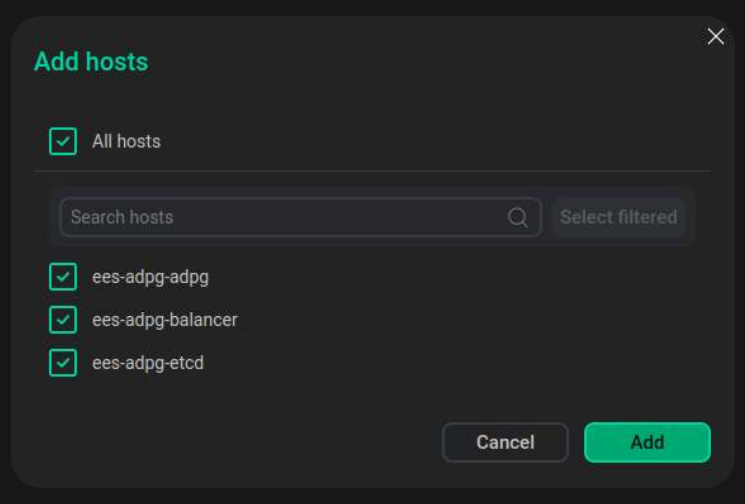
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



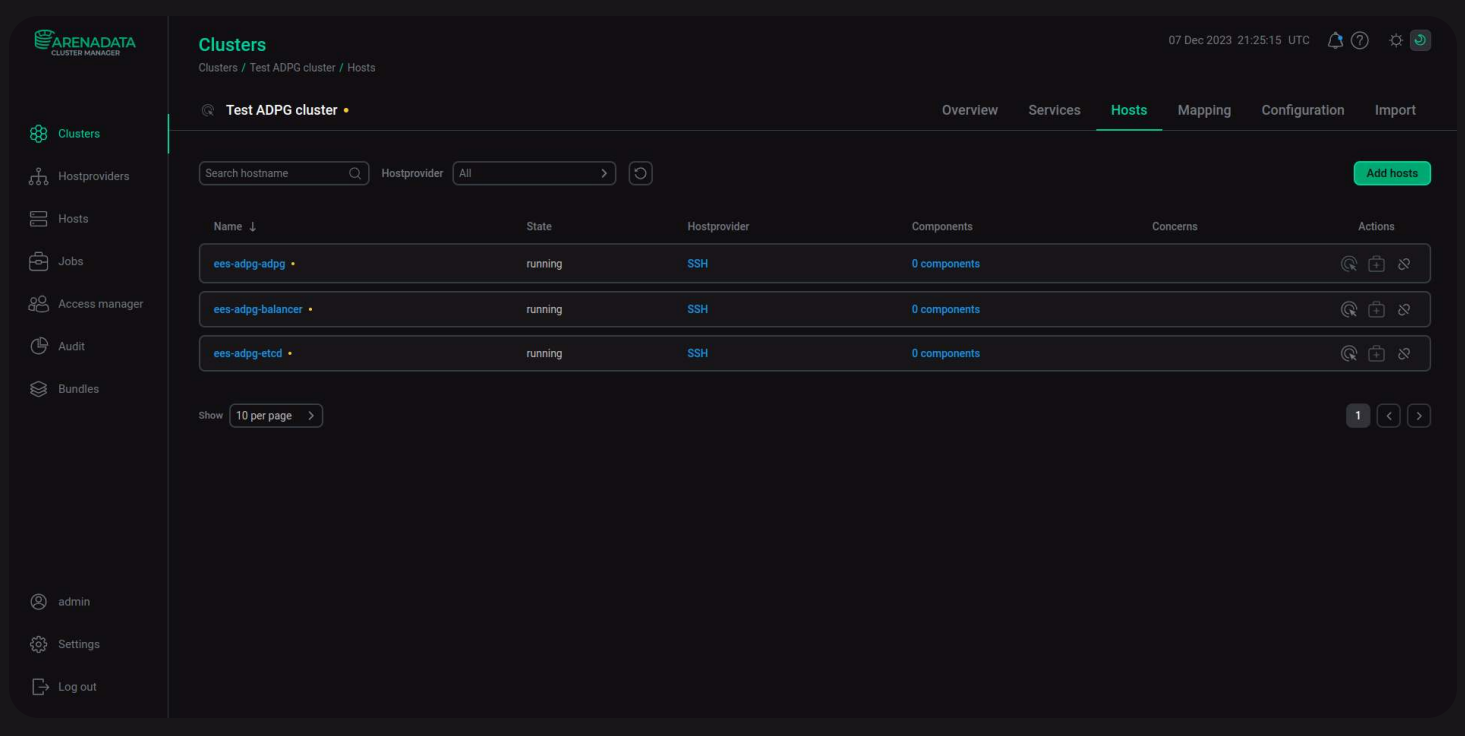
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Дарья Барышева, Евгений Кузина

Назад к содержанию

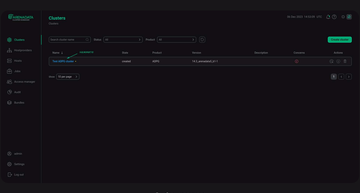
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

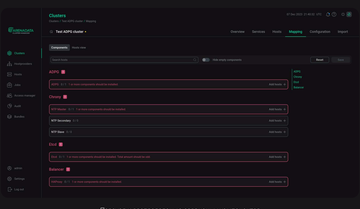
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



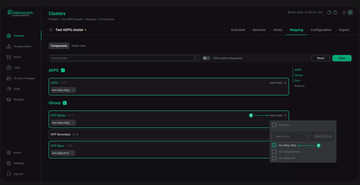
Выбор кластера

2. Откройте вкладку ***Mapping*** на странице кластера.



Переход к распределению сервисных компонентов

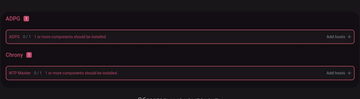
3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите ***Add hosts*** и выберите хосты в открывшемся окне.



Выбор хостов для компонента

Обязательные компоненты

Каждый сервис может иметь обязательные и опциональные компоненты. Обязательные подсвечиваются красным цветом. Также в интерфейсе ADCM для них выводится информация о том, в каком количестве необходимо установить каждый компонент.



Обязательные компоненты

В настоящий момент компоненты ADPG и NTP Master обязательны и должны быть размещены на одном хосте.

ВНИМАНИЕ

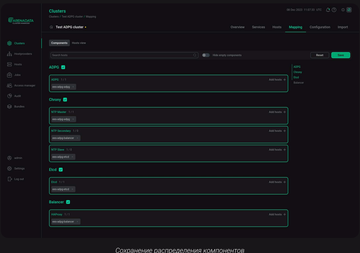
Без назначения нужного числа хостов обязательным компонентам сохранение общей схемы распределения невозможно.

Пример схемы распределения компонентов в кластере ADPG для версии Enterprise приведен в таблице ниже.

Распределение компонентов ADPG

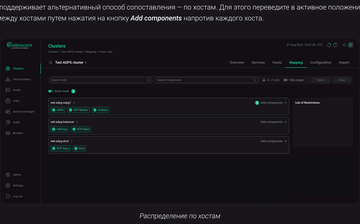
Хост	Сервис	Компонент	Примечание
Хост 1	ADPG	ADPG	Может быть лидером или репликой в архитектуре HA
	ADPG Control Agents	ADPG Control Agent	Агент, позволяющий управлять кластером ADPG с помощью ADPG Control
		ADPG Control ADBM Agent	Агент менеджера бэкапов, который использует ADPG Control
	Chrony	NTP Master	—
Хост 2	ADPG	ADPG	Может быть лидером или репликой в архитектуре HA
	ADPG Control Agents	ADPG Control Agent	Агент, позволяющий управлять кластером ADPG с помощью ADPG Control
		ADPG Control ADBM Agent	Агент менеджера бэкапов, который использует ADPG Control
	Chrony	NTP Secondary	—
Хост 3	ADPG	ADPG	Может быть лидером или репликой в архитектуре HA
	ADPG Control Agents	ADPG Control Agent	Агент, позволяющий управлять кластером ADPG с помощью ADPG Control
		ADPG Control ADBM Agent	Агент менеджера бэкапов, который использует ADPG Control
	Chrony	NTP Secondary	—
Хост 4	Etcd	Etcd	Необходимо использовать нечетное количество нод Etcd. Рекомендовано три и более
	Chrony	NTP Slave	—
Хост 5	Etcd	Etcd	Необходимо использовать нечетное количество нод Etcd. Рекомендовано три и более
	Chrony	NTP Slave	—
Хост 6	Etcd	Etcd	Необходимо использовать нечетное количество нод Etcd. Рекомендовано три и более
	Chrony	NTP Slave	—
Host 7	Balancer	HAProxy	—

4. После завершения распределения компонентов нажмите ***Save***.



Сохранение распределения компонентов

Начиная с версии 2.2.0 ADCM поддерживает альтернативный способ сопоставления — по хостам. Для этого переведите в активное положение переключатель ***Hosts mode*** и распределите компоненты между хостами путем нажатия на кнопку ***Add components*** напротив каждого хоста.



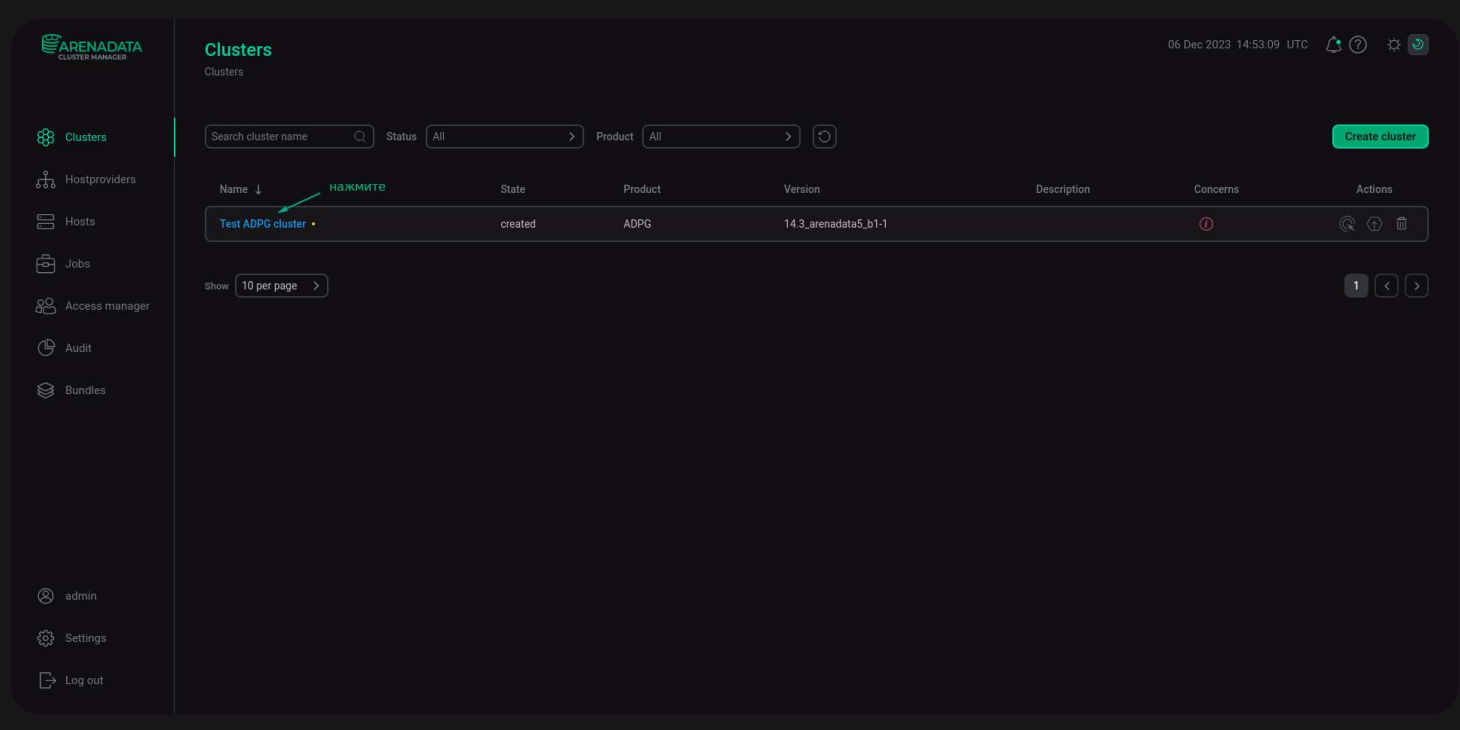
Распределение по хостам

Настройка сервисов

Дарья Барышева, Евгения Кузина

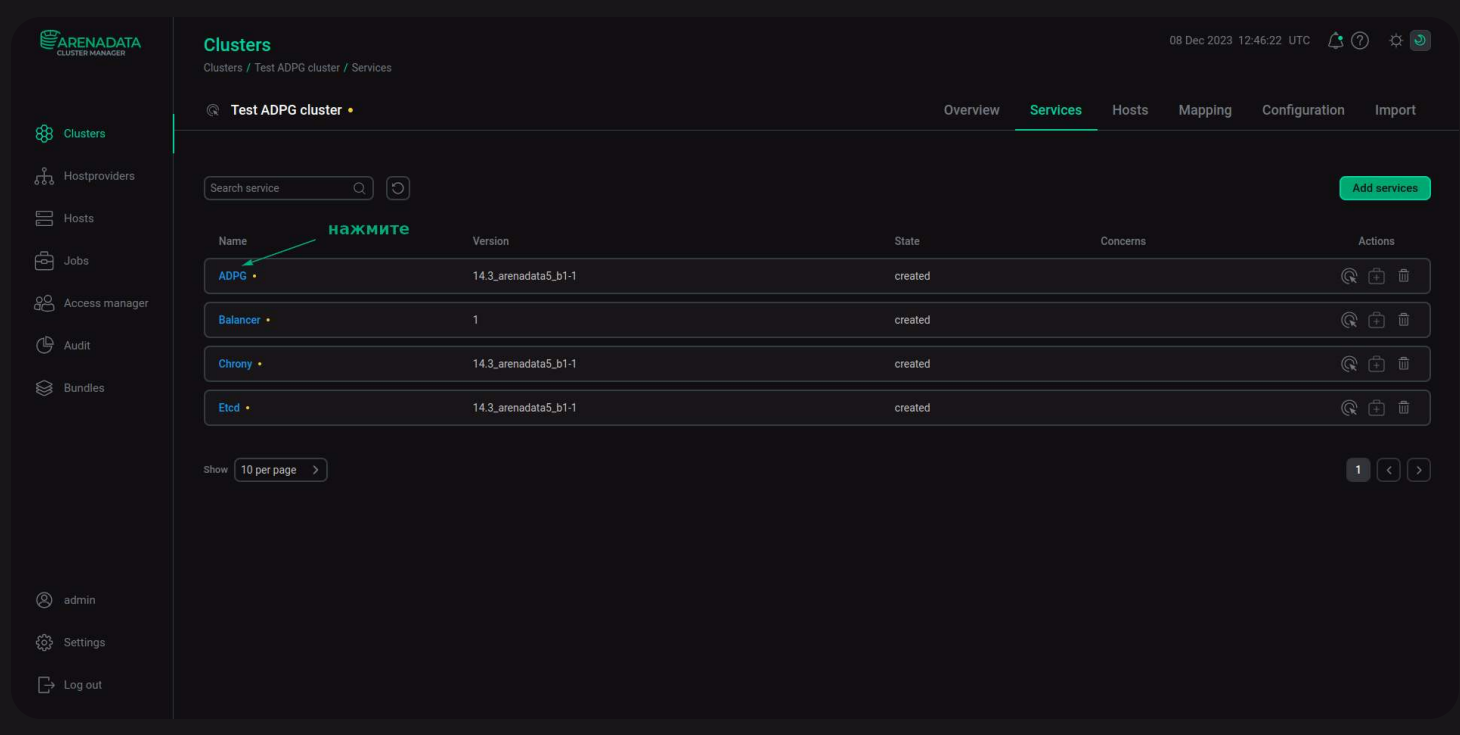
Назад к содержанию

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



Выбор кластера

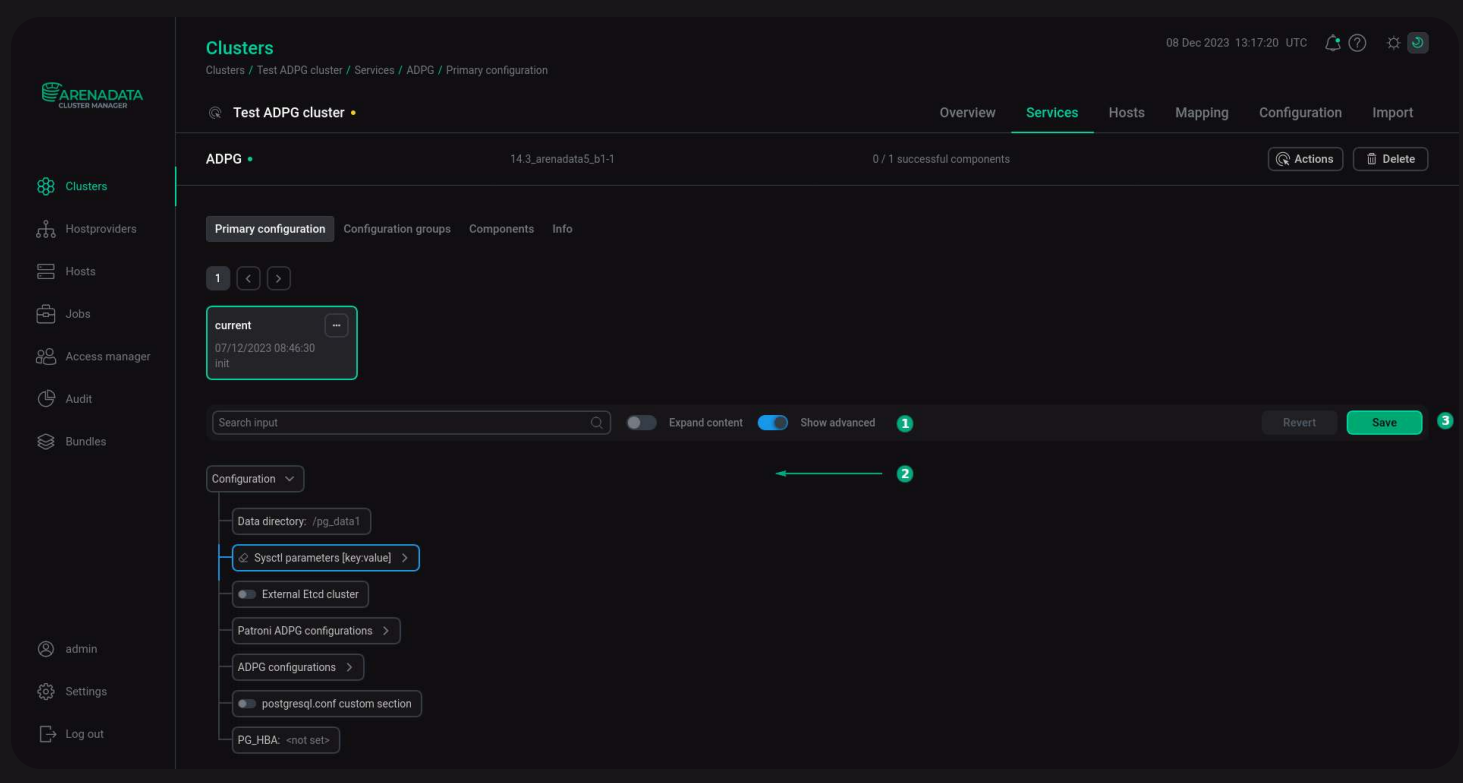
2. Откройте вкладку ***Services*** на странице кластера. Нажмите на имя сервиса в столбце ***Name***.



Переход к настройке сервиса

3. В открывшемся окне:

   - Откройте вкладку ***Primary Configuration***.
   - Переведите в активное состояние переключатель ***Show advanced***.
   - Заполните конфигурационные параметры выбранного сервиса. Поля, подсвеченные красным, обязательны для заполнения.
   - Нажмите ***Save***.



Настройка сервиса

Обратите внимание на следующие параметры:

- Сервис ***Chrony*** → Параметр ***NTP servers***. Хотя этот параметр необязателен, мы рекомендуем установить в нем адрес ближайшего NTP-сервера (например, используемого в вашей компании).

ПРИМЕЧАНИЕ

- Если вы планируете использовать компонент ADPG Control ADBM Agent сервиса ADPG Control Agents, убедитесь, что переключатель Enable backups выключен. Этот компонент представляет собой менеджер бэкапов, и использовать оба метода резервного копирования одновременно невозможно.
- Более подробную информацию по всем конфигурационным параметрам можно получить в статье Конфигурационные параметры.

Настройка кластера

Евгения Кузина, Дарья Барышева

Содержание

Назад к содержанию

Primary configuration

Configuration groups

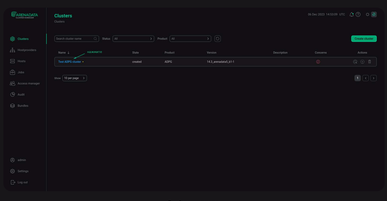
Ansible settings

ВАЖНО

В большинстве случаев кастомная настройка кластера не требуется — значения параметров можно оставить по умолчанию.

После добавления нового кластера можно выполнить его настройку. Для этого выполните шаги:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



Выбор кластера

2. Откройте вкладку ***Configuration*** на странице кластера. Вкладка ***Configuration*** состоит из трех подразделов: ***Primary configuration***, ***Configuration groups***, ***Ansible settings***.
3. Заполните необходимые конфигурационные параметры на нужной вкладке и нажмите ***Save***.

Primary configuration

Вкладка ***Configuration*** → ***Primary configuration*** содержит основные конфигурационные параметры кластера.

Конфигурационные параметры кластера перечислены ниже. В основном они описывают пути к репозиториям, которые будут использованы в процессе установки в зависимости от операционной системы кластера (ALT Linux 8, Astra Linux 1.7, CentOS 7 или Ubuntu 22.04). На серверах кластера регистрируются только отмеченные репозитории. Значения по умолчанию при необходимости могут быть изменены.

Основные конфигурационные параметры

Параметр	Описание
Cluster UUID	Уникальный идентификатор кластера
Disable firewall	Отключает **firewalld** на хостах кластера

Altlinux 8 repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

Astra linux repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

Centos 7 repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

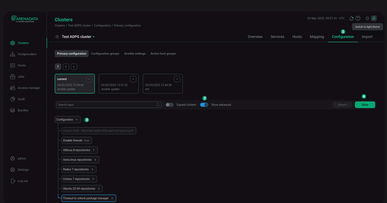
Ubuntu 22.04 repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

Timeout to unlock package manager

Раздел доступен при активации переключателя ***Show advanced***. Параметры этой секции предназначены для настройки тайм-аута ожидания разблокировки пакетного менеджера DNF. В Ubuntu после перезагрузки хоста запускается обновление, что приводит к блокировке DNF.

Параметр	Описание	Значение по умолчанию
retries	Максимальное число повторных попыток	3
delay	Тайм-аут между повторными попытками (в секундах)	10



Конфигурирование кластера

Configuration groups

Вкладка ***Configuration*** → ***Configuration groups*** предназначена для настройки конфиг-групп кластера.

Ansible settings

Вкладка ***Configuration*** → ***Ansible settings*** предназначена для ввода конфигурационных настроек Ansible на уровне кластера. Вкладка доступна начиная с версии ADCM 2.2.0.

Параметры Ansible

Имя	Описание	По умолчанию
forks	Число параллельных процессов, генерируемых при коммуникации с удаленными хостами	5

Установка кластера

Документация по установке кластера PostgreSQL

AD.PG

Содержание

Назад к содержанию

Шаг 1. Предварительная проверка (Precheck)

Шаг 2. Запуск установки

- Установка всего кластера
- Установка отдельных серверов

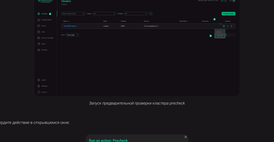
Шаг 3. Проверка статуса кластера после установки

Шаг 4. Подключение к PostgreSQL через psql

Шаг 1. Предварительная проверка (Precheck)

Перед установкой нового кластера рекомендуется выполнить его предварительную проверку – ***Precheck***. Проверка помогает убедиться в том, что все необходимые репозитории доступны, в кластере нет хостов с одинаковыми IP-адресами и все прочие требования соблюдены. Для запуска проверки:

1. Примените действие **Precheck** к кластеру, выбранному на странице **Clusters**, нажав на иконку ⚙ в столбце **Actions**.

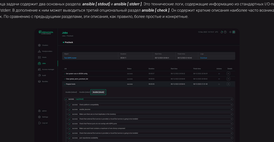


Запуск предварительной проверки кластера prechеck

2. Подтвердите действие в открывшемся окне:

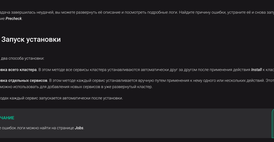
Подтверждение действия

3. Откройте страницу **Jobs**, чтобы увидеть результат действия **Precheck**. Если действие завершается успешно, соответствующая ему задача на странице **Jobs** переходит из статуса `PENDING` → `RUNNING` → `SUCCESS`. В случае ошибки – в статус `FAILED`. Чтобы получить более детальную информацию о возможном сообщении, можно перейти на страницу задачи, кликнув по ней в списке задач на странице **Jobs**.



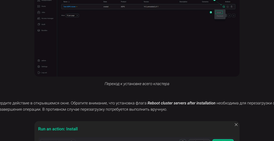
Переход к списку задач (jobs)

Откроется окно с шагами выполнения задачи. Кликните мышкой/тапом в столбце **Details**, чтобы получить дополнительную информацию об определенном шаге:



Переход на страницу с подробностями выполнения

Страница задачи содержит два основных раздела: **ansible [stdout]** и **ansible [stderr]**. Это технические логи, содержащие информацию из стандартных I/O-потоков utility-скриптов. В дополнение к ним может выводиться третий специализированный раздел **ansible [fatal]**. Он содержит краткое описание наиболее часто возникающих ошибок. По сравнению с предыдущими разделами, эти описания, как правило, более простые и конкретные.



Подробности выполнения задачи

4. Если задача завершилась неудачей, вы можете развернуть ее описание и посмотреть подробные логи. Найдите причину ошибки, устраните ее и снова запустите действие **Precheck**.

Шаг 2. Запуск установки

Существует два способа установки:

- **Установка всего кластера**. В этом методе все серверы кластера устанавливаются автоматически друг за другом после применения действия **Install** к кластеру.
- **Установка отдельных серверов**. В этом методе каждый сервер устанавливается вручную путем применения к нему одного или нескольких действий. Этот способ также можно использовать для добавления новых серверов в уже развернутый кластер.

В обоих методах каждый сервер запускается автоматически после установки.

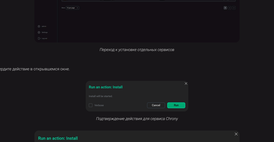
ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице **Jobs**.

Установка всего кластера

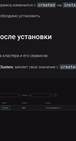
Чтобы установить весь сервера кластера за одну операцию, выполните шаги:

1. Примените действие **Install** к кластеру, выбранному на странице **Clusters**, нажав на иконку ⚙ в столбце **Actions**.



Переход к установке всего кластера

2. Подтвердите действие в открывшемся окне. Обратите внимание, что установку флага **Reboot cluster servers after installation** необходимо для перезагрузки системы после завершения операции. В противном случае перезагрузку потребуется выполнить вручную.



Подтверждение действия

3. Дождитесь завершения установки.
4. Если одна или несколько серверов не удается устранить, можно определить причину ошибок по логам, исправить их и перезапустить установку. Для этого необходимо выбрать действие **Reinstall** в столбце **Actions** на странице **Clusters**.

Перезапуск установки

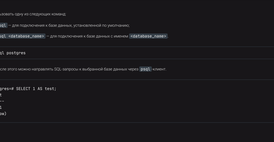
Установка отдельных серверов

Чтобы установить один или несколько серверов вручную, необходимо предварительно добавить эти серверы в кластер и затем применить к ним действия, описанные в подразделе выше.

ВНИМАНИЕ!

Обратите внимание, что серверы устанавливаются по очереди друг за другом. Не пытайтесь установить следующий сервер до окончания установки предыдущего.

1. Откройте вкладку **Services** на странице кластера. Для сервера, который должен быть установлен, нажмите на иконку ⚙ в столбце **Actions** и выберите действие **Install** из выпадающего списка.



Переход к установке отдельных серверов

2. Подтвердите действие в открывшемся окне:

Подтверждение действия для сервера Citrony

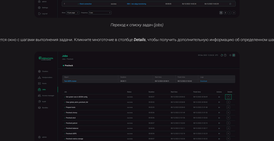
Подтверждение действия для всех серверов, кроме Citrony

3. Дождитесь завершения установки и убедитесь, что статус сервера изменился с `created` → `installed`.
4. Повторите предыдущие шаги для всех серверов, которые необходимо установить.

Шаг 3. Проверка статуса кластера после установки

В результате успешной установки произойдут обновления статуса кластера и его серверов:

- Статус кластера, отображаемый в столбце **State** на странице **Clusters**, начнет свое значение с `created` → `installed`



Статус кластера после успешной установки

- Статус серверов, отображаемый на вкладке **Services** страницы кластера, начнет свое значение с `created` → `installed`.

Статус серверов кластера после успешной установки

Шаг 4. Подключение к PostgreSQL через psql

Чтобы убедиться в успешной установке PostgreSQL и создании всех баз данных, необходимо:

1. Подключиться к серверу ADPG (например, через SSH) под пользователем `postgres`, который создается по умолчанию.

```
$ sudo su - postgres
```

2. Вывести список всех баз данных PostgreSQL.

```
$ psql -l
```

Результат команды приведен ниже.

```
                                  List of databases
   Name    | Owner  | Encoding |  Collate   |   Ctype    |  Access privileges
-----------+--------+----------+------------+------------+-----------------------
 postgres  | postgres | UTF8   | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8   | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |        |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8   | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |        |          |            |            | postgres=CTc/postgres
(3 rows)
```

3. Использовать одну из следующих команд:

- `psql` – для подключения к базе данных, установленной по умолчанию;
- `psql <database_name>` – для подключения к базе данных с именем `<database_name>`.

```
$ psql postgres
```

После этого можно направлять SQL запросы к выбранной базе данных через `psql` клиент.

```
postgres=# SELECT 1 AS test;
 test
------
    1
(1 row)
```

Установка мониторинга

Д. Евгения Кузина

Содержание

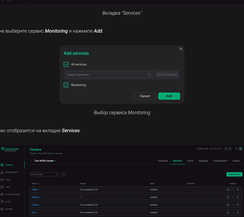
Назад к содержанию

Шаг 1. Добавление сервиса Monitoring
Шаг 2. Настройка мониторинга
Шаг 3. Установка сервиса Monitoring
Шаг 4. Просмотр дашбордов

Система мониторинга входит в бандл ADPG. Для установки мониторинга выполните шаги, описанные ниже.

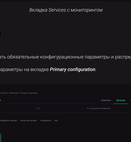
Шаг 1. Добавление сервиса Monitoring

Откройте страницу **Clusters** в пользовательском интерфейсе ADCM и выберите кластер ADPG. Откройте вкладку **Services** и нажмите **Add services**.



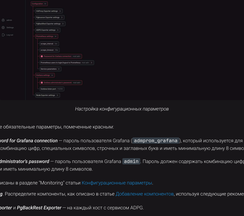
Вкладка "Services"

В открывшемся диалоговом окне выберите сервис **Monitoring** и нажмите **Add**.



Выбор сервиса Monitoring

В результате добавленный сервис отобразится на вкладке **Services**.

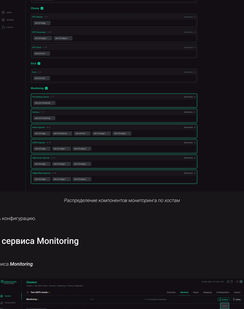


Вкладка Services с мониторингом

Шаг 2. Настройка мониторинга

Иконка ⓘ **Concerns** показывает, что необходимо указать обязательные конфигурационные параметры и распределить компоненты мониторинга по хостам.

Кликните сервис **Monitoring** и установите необходимые параметры во вкладке **Primary configuration**.



Настройка конфигурационных параметров

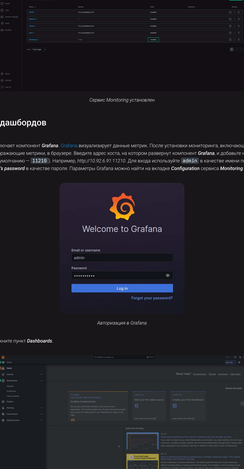
Необходимо указать следующие обязательные параметры, помеченные красным:

- **Prometheus settings** → **Password for Grafana connection** — пароль пользователя Grafana (`admin_grafana`), который используется для подключения к Prometheus. Пароль должен содержать комбинацию цифр, специальных символов, строчных и заглавных букв и иметь минимальную длину 8 символов.
- **Grafana settings** → **Grafana administrator's password** — пароль пользователя Grafana (`admin`). Пароль должен содержать комбинацию цифр, специальных символов, строчных и заглавных букв и иметь минимальную длину 8 символов.

Все параметры мониторинга описаны в разделе "Monitoring" статьи Конфигурационные параметры.

Затем откройте вкладку **Mapping**. Распределите компоненты, как описано в статье Добавление компонентов, используя следующие рекомендации:

- **ADPG Exporter**, **Pgbouncer Exporter** и **Pgbackrest Exporter** — на каждый хост с сервисом ADPG.
- **Prometheus Server** и **Grafana** — на отдельный хост.
- **Node Exporter** — на каждый хост, с которого нужно собирать метрики операционной системы.

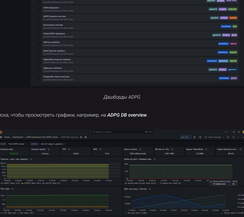


Распределение компонентов мониторинга по хостам

Нажмите **Save**, чтобы сохранить конфигурацию.

Шаг 3. Установка сервиса Monitoring

Выполните действие **Install** сервиса **Monitoring**.



Действие "Install" сервиса Monitoring

В следующем диалоговом окне нажмите **Run** для подтверждения действия. Вы также можете активировать флажок **Verbose**, чтобы увидеть дополнительные сведения о выполнении действия на странице **Jobs**.

Подтверждение действия "Install"

В результате статус сервиса Monitoring изменится на `Installed`.

Сервис Monitoring установлен

Шаг 4. Просмотр дашбордов

Система мониторинга в ADPG включает компонент **Grafana** (версия 9.5.2), визуализирует данные метрик. После установки мониторинга, включающего компонент **Grafana**, можно просматривать дашборды, отображающие метрики, в браузере. Введите адрес хоста, на котором развернут компонент **Grafana**, и добавьте номер порта — параметр **Grafana listen port** (значение по умолчанию — `11210`). Например, *http://10.92.9.91:11210*. Для входа используйте `admin` в качестве имени пользователя и значение пароля из **Grafana administrator's password** в качестве пароля. Параметры Grafana можно найти на вкладке **Configuration** сервиса **Monitoring**.

Авторизация в Grafana

Разверните меню Grafana и кликните пункт **Dashboards**.

Интерфейс Grafana

Откроется страница со списком дашбордов мониторинга. Разверните элемент **ADPG Dashboard**, для вашего кластера, чтобы отобразить доступные дашборды.

Дашборды ADPG

Нажмите на любой элемент списка, чтобы просмотреть графики, например, на **ADPG DB overview**.

Графики ADPG DB overview

Offline-установка

Назад к содержанию

Этот раздел посвящен **offline**-установке ADPG. Данный способ установки подходит для хостов без доступа к Интернету.

Для offline-установки необходимо установить и настроить Arenadata Enterprise Tools (ET). ET — это решение, предоставляющее инфраструктуру для развертывания продуктов Аренадата в среде с ограниченным доступом в интернет. Решение является частью платформы данных Аренадата и предоставляет репозитории, необходимые для установки и обновления компонентов платформы.

Процесс offline-установки через ADCM включает следующие шаги.

ПРИМЕЧАНИЕ

Эта функция доступна только в Enterprise-версии ADPG.

Шаг 1.

Установка ADCM

Шаг 2.

Подготовка хостов

Шаг 3.

Установка кластера ET

Шаг 4.

Установка кластера ADPG

Шаг 5.

Установка мониторинга

Шаг 6.

Установка ADPG Control

Установка АЛСМ

Скачать с трекинга

Содержание

Шаг 1. Предварительные требования
- Требования к железу и системе
- Установка Docker
- Установка АЛСМ
- Установка Portainer
- Установка Nginx Proxy Manager и Let's Encrypt SSL
Шаг 2. Установка АЛСМ
Шаг 3. Запуск АЛСМ
Шаг 4. Проверка установки
- Проверка статуса Docker-контейнеров
- Проверка доступности АЛСМ в браузере
- Проверка веб-интерфейса АЛСМ
Шаг 5. Обновление АЛСМ

Итоговый адрес АЛСМ будет выглядеть примерно так: https://alsm.example.com

Шаг 1. Предварительные требования

Требования к железу и системе

Для установки АЛСМ вам потребуется сервер, который соответствует следующим минимальным требованиям:

- Архитектура x86-64
- 4 ядра CPU
- 8 ГБ RAM
- 40 ГБ SSD
- Ubuntu 22.04 LTS

Также вам потребуется:

- Доменное имя, которое вы будете использовать для доступа к АЛСМ
- Публичный IP-адрес вашего сервера

Установка Docker

Установите Docker и Docker Compose, если они еще не установлены. Подключитесь к серверу по SSH и выполните команды:

```
curl -fsSL https://get.docker.com | sh
```

Скачивает скрипт установки Docker с официального сайта и сразу выполняет его. Скрипт автоматически определяет дистрибутив и устанавливает Docker Engine, CLI и containerd.

```
sudo systemctl enable --now docker
```

Включает автозапуск Docker при загрузке системы и немедленно запускает службу.

```
docker compose version
```

Проверяет, что Docker Compose установлен корректно. Должна вывестись версия, например `Docker Compose version v2.x.x`.

Установка Portainer

Portainer — это веб-интерфейс для управления Docker-контейнерами. Он значительно упрощает работу с АЛСМ.

```
docker volume create portainer_data
```

Создаёт именованный том для хранения данных Portainer (настроек, пользователей и т.д.).

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
```

Запускает Portainer в фоновом режиме (`-d`). Открывает порты 8000 и 9443, монтирует сокет Docker и том с данными, автоматически перезапускается при сбоях.

Откройте в браузере `https://<IP-сервера>:9443` и создайте учётную запись администратора.

Установка Nginx Proxy Manager и Let's Encrypt SSL

Nginx Proxy Manager будет принимать все входящие запросы и направлять их в нужные контейнеры. Также он автоматически получит SSL-сертификат.

```
mkdir -p ~/npm && cd ~/npm
```

Создаёт рабочую директорию и переходит в неё.

```
nano docker-compose.yml
```

Открывает редактор для создания файла конфигурации.

```
services:
  npm:
    image: jc21/nginx-proxy-manager:latest
    restart: always
    ports:
      - '80:80'
      - '443:443'
      - '81:81'
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
```

Конфигурация Docker Compose: образ, автоперезапуск, порты 80 (HTTP), 443 (HTTPS), 81 (панель управления), тома для данных и сертификатов.

```
docker compose up -d
```

Запускает Nginx Proxy Manager в фоновом режиме.

Откройте `http://<IP-сервера>:81` и войдите с данными по умолчанию:

- Email: `admin@example.com`
- Password: `changeme`

Сразу смените email и пароль при первом входе.

Шаг 2. Установка АЛСМ

Создайте директорию для АЛСМ и перейдите в неё:

```
mkdir ~/alsm && cd ~/alsm
```

Создаёт директорию для АЛСМ и переходит в неё.

```
nano docker-compose.yml
```

Открывает редактор для создания файла конфигурации.

```
services:
  alsm:
    image: ghcr.io/alsm/alsm:latest
    restart: always
    environment:
      - ALSM_SECRET_KEY=<сгенерируйте-случайную-строку>
      - ALSM_DB_PATH=/data/alsm.db
      - ALSM_BASE_URL=https://alsm.example.com
    volumes:
      - ./data:/data
```

Конфигурация АЛСМ: образ, автоперезапуск, переменные окружения (секретный ключ, путь к базе данных, базовый URL), том для данных.

```
openssl rand -hex 32
```

Генерирует случайную строку для `ALSM_SECRET_KEY`. Скопируйте вывод и замените `<сгенерируйте-случайную-строку>` в файле.

```
docker compose up -d
```

Запускает АЛСМ в фоновом режиме.

Шаг 3. Запуск АЛСМ

Проверьте, что все контейнеры запущены:

```
docker ps
```

Показывает список запущенных контейнеров. Должны быть видны `alsm`, `portainer` и `npm`.

```
docker logs alsm
```

Показывает логи контейнера АЛСМ. Убедитесь, что нет ошибок.

Шаг 4. Проверка установки

Проверка статуса Docker-контейнеров

Убедитесь, что все контейнеры работают:

```
docker ps
```

Показывает список запущенных контейнеров. Все должны иметь статус `Up`.

```
docker compose ps
```

Показывает статус контейнеров, управляемых Docker Compose.

```
NAME      IMAGE                                  COMMAND                  SERVICE   CREATED         STATUS         PORTS
alsm      ghcr.io/alsm/alsm:latest               "/entrypoint.sh"         alsm      2 minutes ago   Up 2 minutes   0.0.0.0:8080->8080/tcp
```

Проверка доступности по HTTPS

Откройте в браузере `https://alsm.example.com`. Должна загрузиться страница входа АЛСМ с валидным SSL-сертификатом (замок в адресной строке).

```
curl -I https://alsm.example.com
```

Проверяет доступность и SSL-сертификат. Должен вернуться `HTTP/2 200`.

Проверка веб-интерфейса АЛСМ

1. Откройте `https://alsm.example.com`
2. Создайте первого администратора при первом входе
3. Убедитесь, что все разделы интерфейса загружаются
4. Проверьте, что можно добавить тестовый объект

Интерфейс АЛСМ

Добавление тестового объекта

Установка ОВ, АЛСМ

Скачайте [дистрибутив АЛСМ](#) для вашей операционной системы.

Запустите установщик и следуйте инструкциям мастера установки.

Стартовый экран установщика

Подтверждение лицензии АЛСМ

Укажите путь установки. Рекомендуется оставить путь по умолчанию.

Выбор директории установки

Дождитесь завершения установки.

Завершение установки АЛСМ

Шаг 5. Обновление АЛСМ

Обновление Docker-контейнеров

Для обновления АЛСМ до последней версии выполните следующие команды:

```
cd ~/alsm
```

Переходит в директорию АЛСМ.

```
docker compose pull
```

Скачивает последние версии образов, указанных в `docker-compose.yml`.

```
docker compose up -d
```

Пересоздаёт контейнеры с новыми образами. Данные сохраняются в томах.

```
docker image prune -f
```

Удаляет неиспользуемые образы, освобождая место на диске.

Обновление Portainer

```
docker pull portainer/portainer-ce:latest
```

Скачивает последнюю версию Portainer.

```
docker stop portainer && docker rm portainer
```

Останавливает и удаляет старый контейнер Portainer.

```
docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer-ce:latest
```

Запускает новый контейнер Portainer с теми же параметрами. Данные сохранятся в томе `portainer_data`.

Обновление Nginx Proxy Manager

```
cd ~/npm && docker compose pull && docker compose up -d
```

Переходит в директорию Nginx Proxy Manager, скачивает последнюю версию и пересоздаёт контейнер.

Обновление Let's Encrypt SSL

Сертификаты Let's Encrypt обновляются автоматически. Nginx Proxy Manager проверяет и продлевает сертификаты за 30 дней до истечения срока действия.

Чтобы принудительно обновить сертификат, откройте веб-интерфейс Nginx Proxy Manager, перейдите в раздел **SSL Certificates**, выберите нужный сертификат и нажмите **Renew**.

Подготовка хостов

Назад к содержанию

Перед установкой нового кластера необходимо добавить информацию о его будущих хостах в ADCM. Это задача **хостпровайдеров** — специальных плагинов, отвечающих за взаимодействие между хостами и ADCM.

ADCM поддерживает несколько типов хостпровайдеров. Для получения подробной информации о том, как их настроить и использовать для создания хостов, обратитесь к документации:

- SSH
- Yandex Cloud
- VCloud (VMware)
- Cloud.ru Advanced
- CROC Cloud

Для offline-установки рекомендуется использование SSH-хостпровайдера.

ВАЖНО

Запрещено использовать зарезервированные слова `adpg`, `balancer`, `chrony`, `etcd` и `prometheus_monitoring` в качестве имен хостов кластера ADPG.

Убедитесь, что таблицы маршрутизации на каждом хосте содержат маршрут по умолчанию — `0.0.0.0/0`.

Используйте команду `route`, чтобы это проверить:

```
$ route
```

Вывод команды должен быть похож на следующий:

```
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         gateway         0.0.0.0         UG    100    0        0 eth0
10.92.40.0      0.0.0.0         255.255.252.0   U     100    0        0 eth0
```

Если маршрут по умолчанию `0.0.0.0/0` отсутствует, добавьте его с помощью следующей команды:

```
$ sudo route add default gw 0.0.0.0 eth0
```

Создание кластера

Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Получение Enterprise-версии бандла

Шаг 2. Загрузка бандла в ADCM

Шаг 3. Создание кластера на базе загруженного бандла

В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

Шаги для добавления нового кластера приведены ниже.

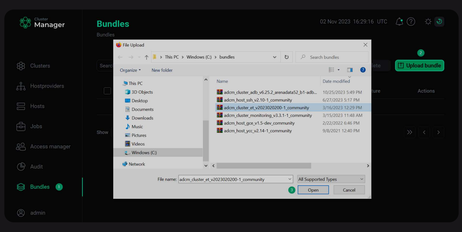
Шаг 1. Получение Enterprise-версии бандла

Пожалуйста, запросите бандл у команды поддержки Arenadata.

Шаг 2. Загрузка бандла в ADCM

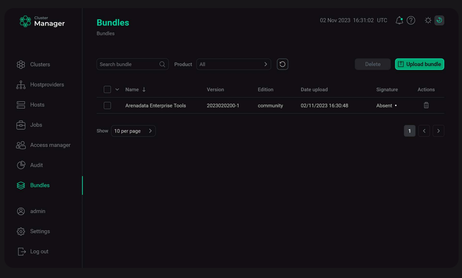
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

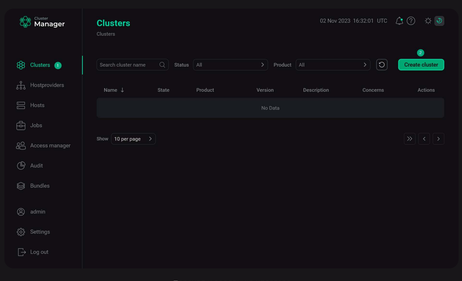
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

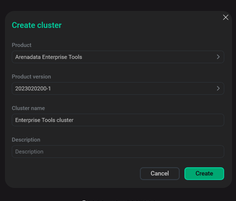
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



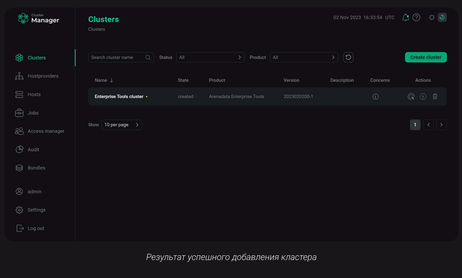
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

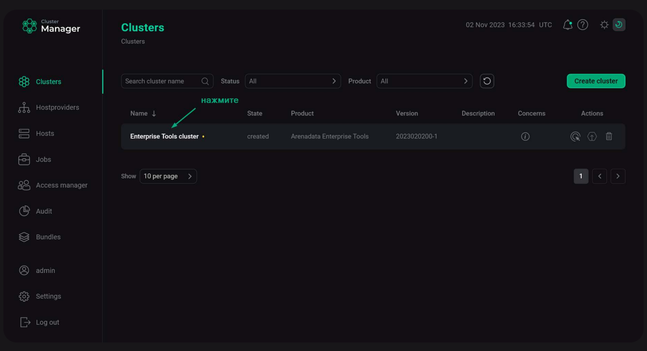
Добавление сервисов

Дарья Барышева

Назад к содержанию

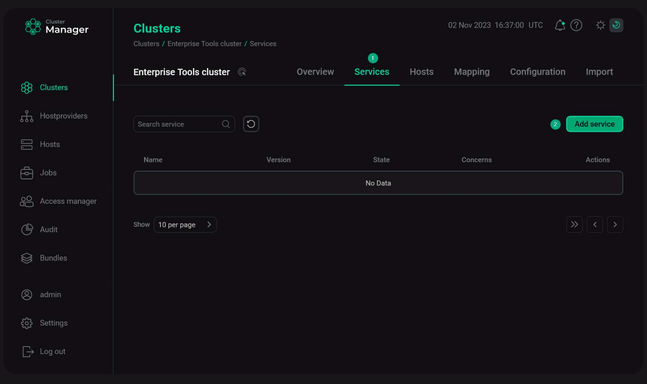
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Для добавления сервисов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



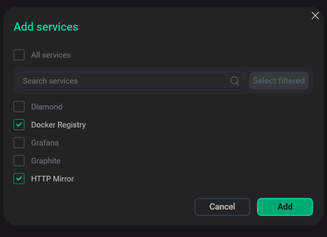
Выбор кластера

2. Откройте вкладку ***Services*** на странице кластера и нажмите ***Add service***.



Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



Выбор сервисов

Краткое описание доступных сервисов приведено ниже.

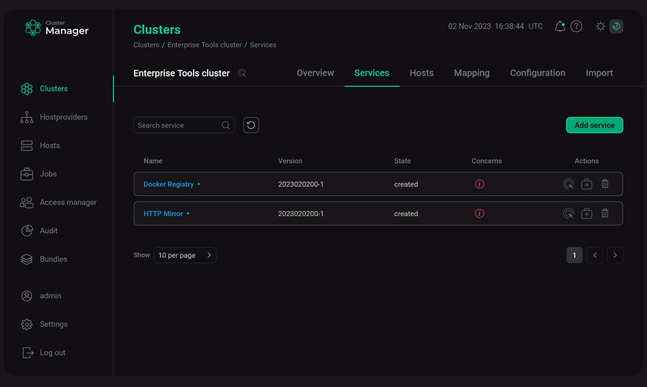
Сервисы, которые необходимо добавить в кластер Enterprise Tools

Сервис	Назначение
HTTP Mirror	Сервис, используемый для размещения пакетов репозиториев, необходимых для offline-установки продуктов
Docker Registry	Сервис, используемый для хранения образов Docker, необходимых для offline-установки продуктов

ПРИМЕЧАНИЕ

Сервис CoreDNS не используется (deprecated). Сервисы Diamond, Graphite и Grafana необходимы для настройки мониторинга — они будут описаны отдельно.

4. В результате успешно добавленные сервисы отображаются на вкладке ***Services***.



Результат успешного добавления сервисов в кластер

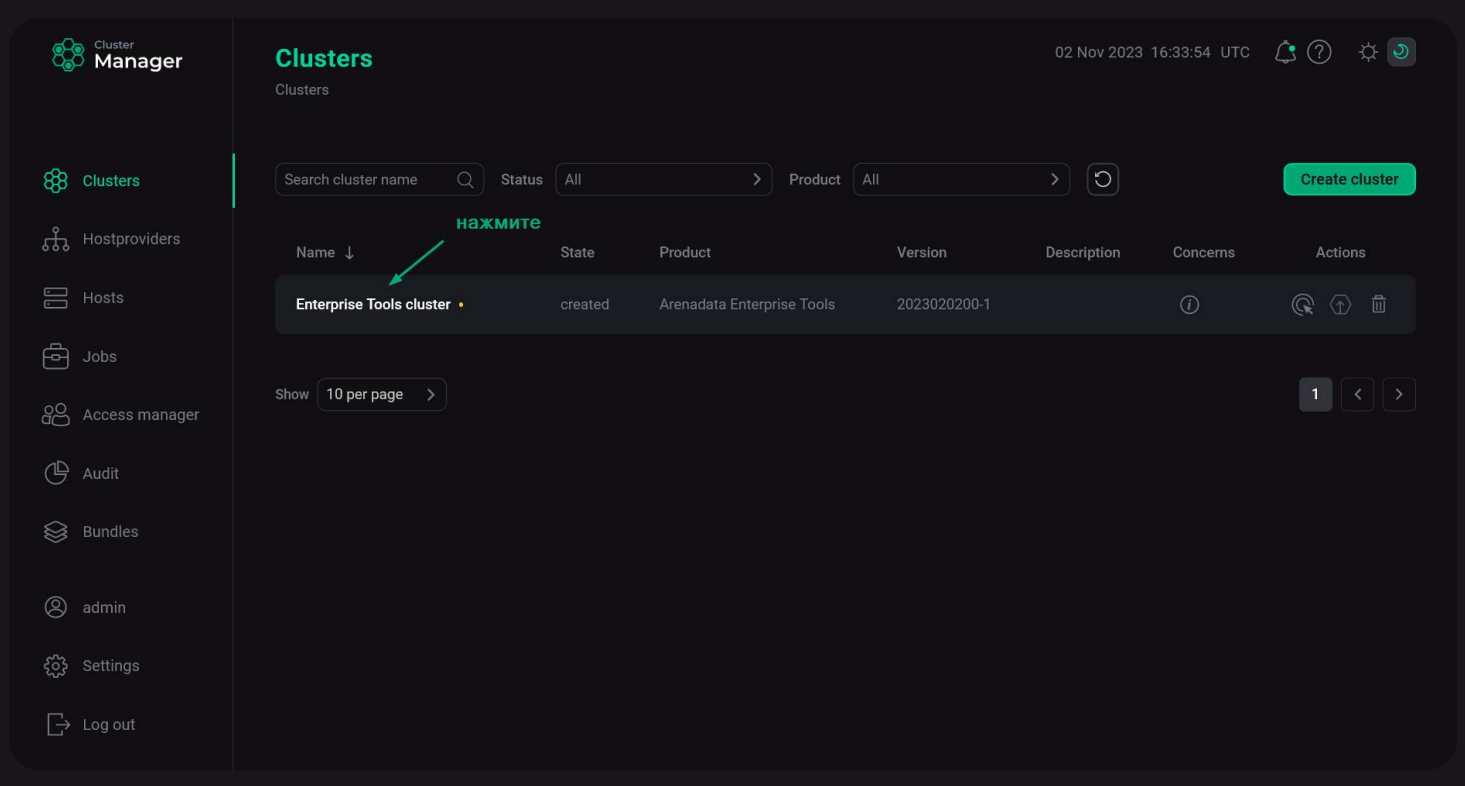
Добавление хостов в кластер

Дарья Барышева

Назад к содержанию

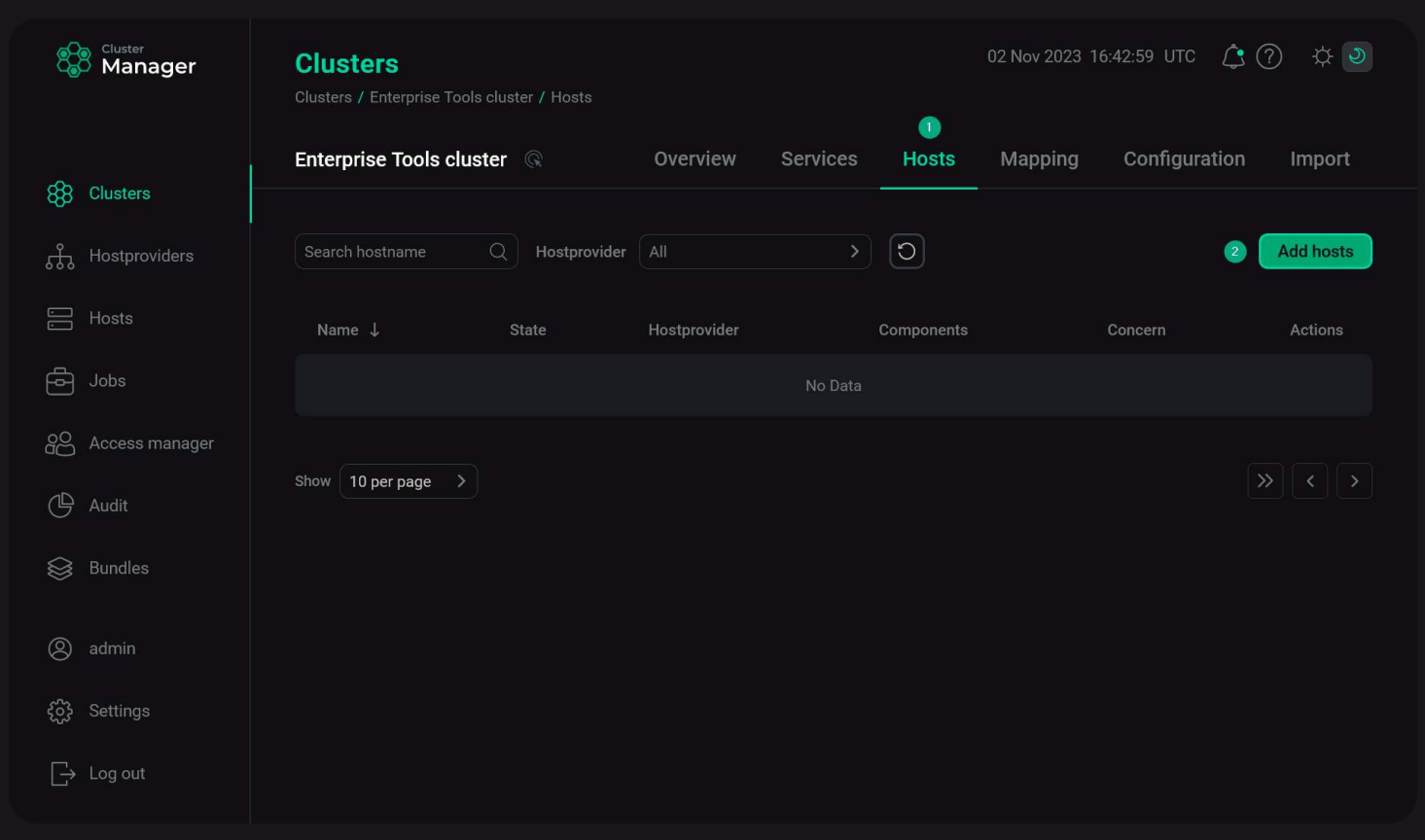
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



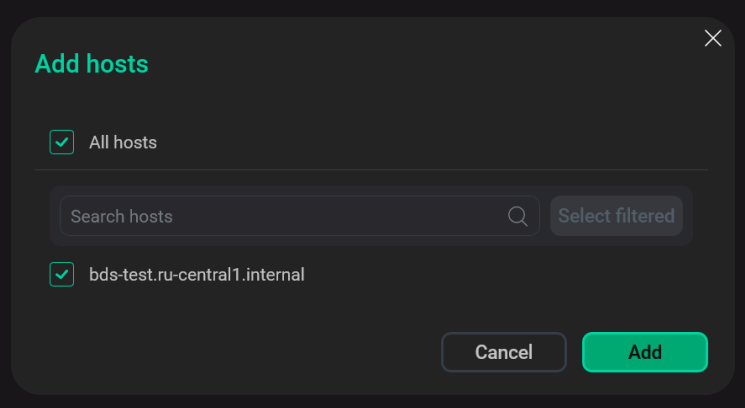
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



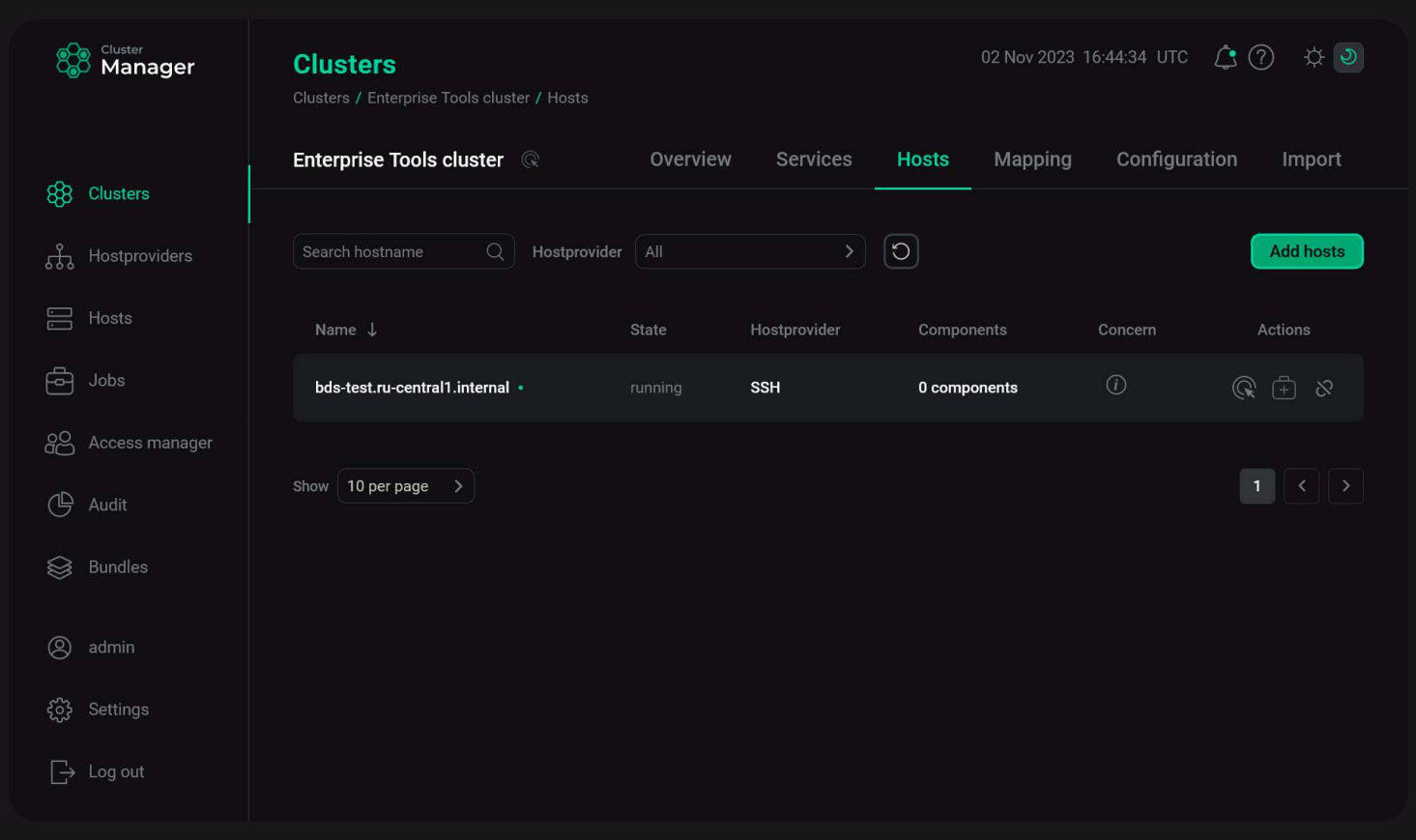
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Дарья Барышева

Назад к содержанию

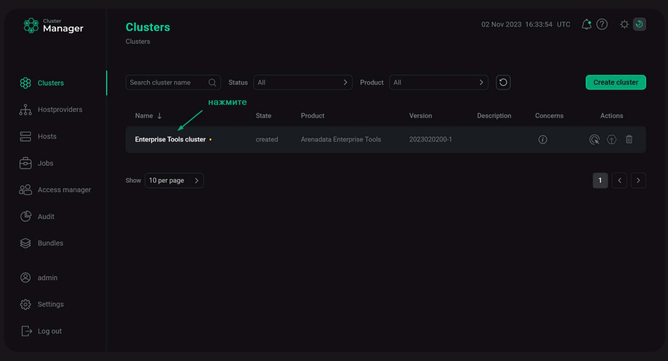
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

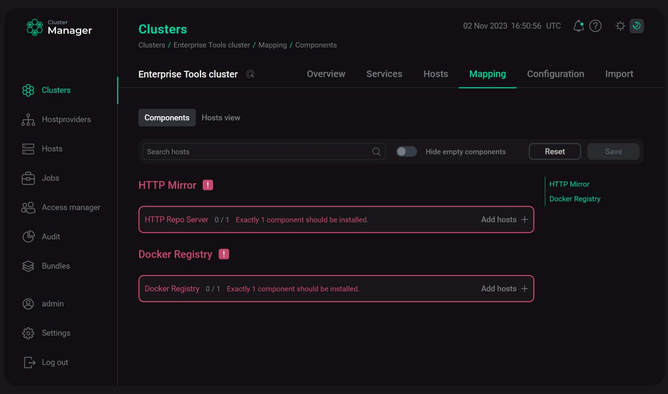
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



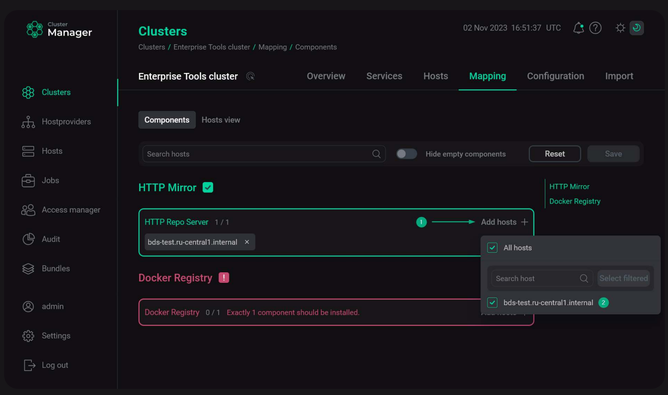
Выбор кластера

2. Откройте вкладку ***Mapping*** на странице кластера.



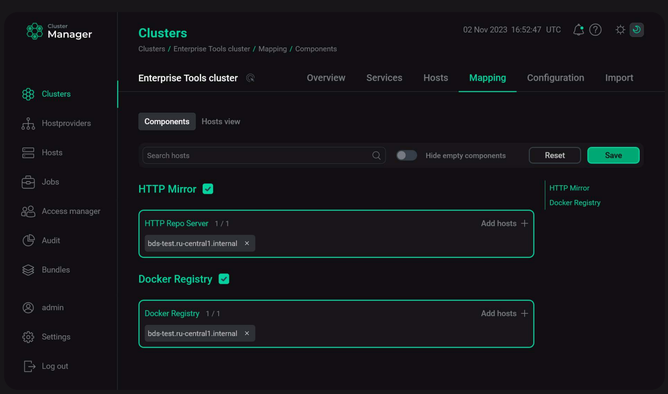
Переход к распределению сервисных компонентов

3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите ***Add hosts*** и выберите хосты в открывшемся окне.



Выбор хостов для компонента

4. После завершения распределения компонентов нажмите ***Save***.



Сохранение распределения компонентов

Установка кластера

Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Запуск установки

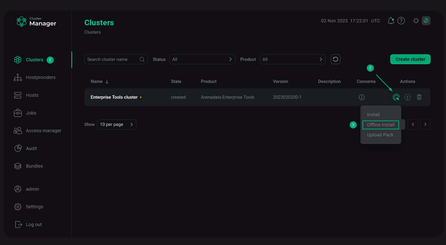
Шаг 2. Загрузка пакетов для offline-установки

Шаг 3. Проверка результатов

Шаг 1. Запуск установки

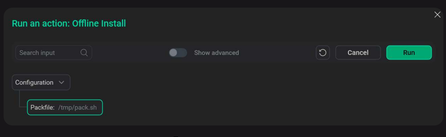
Для установки кластера Enterprise Tools выполните следующие шаги:

1. Запустите действие ***Offline Install*** для кластера, выбранного на странице ***Clusters***, нажав на иконку в столбце ***Actions***.



Переход к установке кластера

2. В открывшемся диалоговом окне введите полный путь к shell-скрипту, полученному от команды поддержки Arenadata, и нажмите ***Run***.

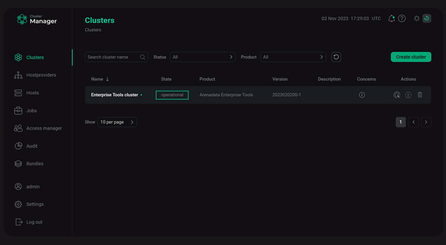


Подтверждение установки

3. Дождитесь завершения процесса установки.

ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице ***Jobs***.

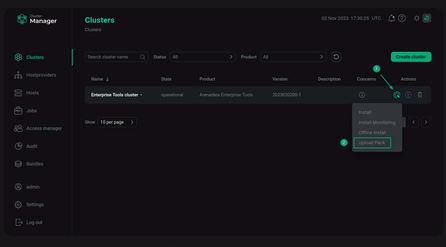


Процесс установки завершен

Шаг 2. Загрузка пакетов для offline-установки

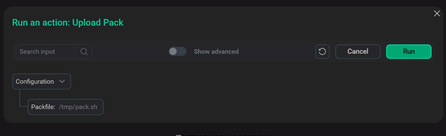
В дополнение к установке сервисов ET требуется загрузка всех установочных файлов и образов в инициализированный http-репозиторий и Docker Registry на хосте, где развернут кластер ET. Это важно для последующего offline-развертывания продуктов Arenadata. Для загрузки выполните шаги:

1. Запустите действие ***Upload Pack*** для кластера, выбранного на странице ***Clusters***, нажав на иконку в столбце ***Actions***.



Переход к загрузке пакетов

2. В открывшемся диалоговом окне введите полный путь к shell-скрипту, полученному от команды поддержки Arenadata, и нажмите ***Run***.



Подтверждение загрузки

3. Дождитесь завершения процесса загрузки.

ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице ***Jobs***.

Шаг 3. Проверка результатов

Для проверки установки на хосте, где развернут кластер, выполните шаги:

1. Убедитесь, что порты `443` и `81` находятся в статусе `LISTEN`. Первый порт используется сервисом Docker Registry, второй порт — сервисом HTTP Mirror.

```
$ sudo netstat -ntlp|grep L|grep -E "81|443"
```

Результат команды:

```
tcp6       0      0 :::443                  :::*                    LISTEN      1694/docker-proxy-c
tcp6       0      0 :::81                   :::*                    LISTEN      1708/docker-proxy-c
```

2. Проверьте наличие директории */opt/arenadata/etools/httprepo/arenadata-repo*. Она должна содержать файлы, необходимые для offline-установки продуктов, выбранных вами на этапе получения shell-скрипта от команды поддержки Arenadata.

```
$ ls -la /opt/arenadata/etools/httprepo/arenadata-repo
```

Следующий пример показывает данные, которые могут храниться в директории в случае установки ADPG.

```
total 0
drwxr-xr-x 5 root root 48 Dec 26 17:46 .
drwxr-xr-x 3 root root 28 Dec 26 17:46 ..
drwxr-xr-x 4 root root 28 Dec 26 17:46 ADM
drwxr-xr-x 3 root root 29 Dec 26 17:46 ADPG
```

Создание кластера

Евгения Кузина, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Получение Enterprise-версии бандла

Шаг 2. Загрузка бандла в ADCM

Шаг 3. Создание кластера на базе загруженного бандла

В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

Шаги для добавления нового кластера приведены ниже.

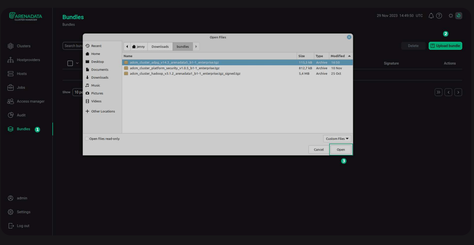
Шаг 1. Получение Enterprise-версии бандла

Пожалуйста, запросите бандл у команды поддержки Arenadata.

Шаг 2. Загрузка бандла в ADCM

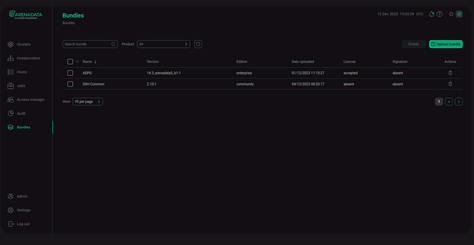
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

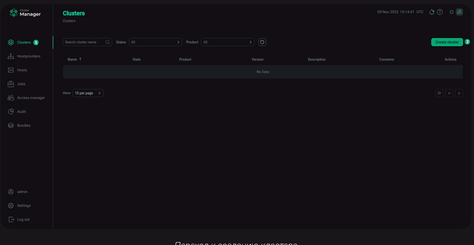
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

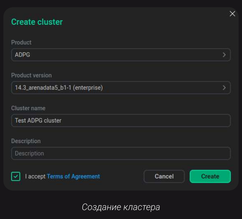
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



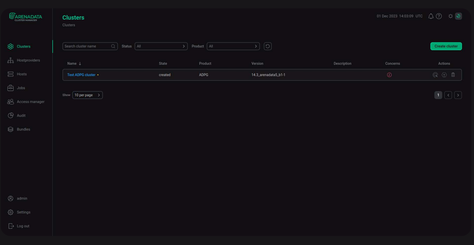
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

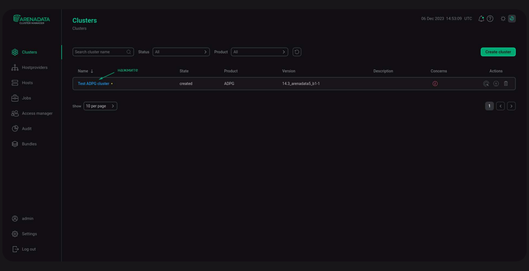
Добавление сервисов

Дарья Барышева, Евгения Кузина

Назад к содержанию

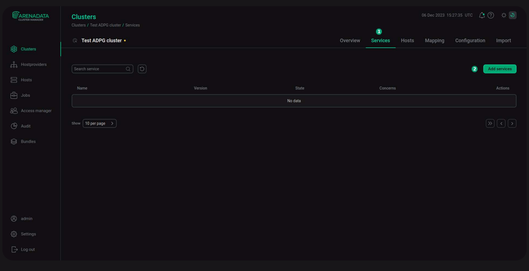
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Примеры сервисов в кластерах ADPG: ADPG, Chrony и так далее. Для добавления сервисов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



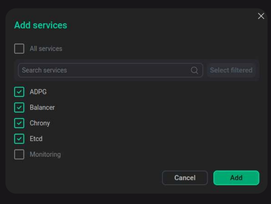
Выбор кластера

2. Откройте вкладку ***Services*** на странице кластера и нажмите ***Add service***.



Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



Выбор сервисов

ВАЖНО

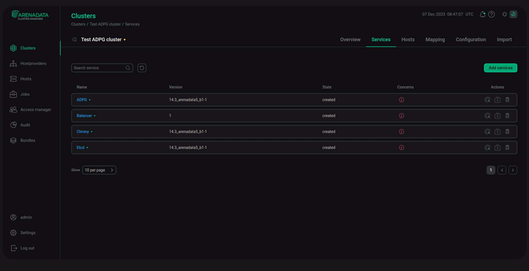
Версия ADPG Enterprise требует обязательного наличия Etcd, встроенного Etcd-сервиса или внешнего кластера, чтобы использовать его как распределенное хранилище конфигурации Patroni (Distributed Configuration Store, DCS). Настройки Etcd должны быть указаны во время установки, после установки их изменить нельзя.

Краткое описание доступных сервисов приведено ниже. CE — Community-версия ADPG (Community Edition), EE — Enterprise-версия ADPG (Enterprise Edition).

Сервисы, которые могут быть добавлены в кластер ADPG

Сервис	Обязательность	Версия	Назначение
ADPG	Да	CE/EE	Основной сервис, соответствующий объектно-реляционной СУБД PostgreSQL
ADPG Control Agents	Нет	EE	Сервис, позволяющий управлять кластером ADPG с помощью ADPG Control
Balancer	Нет	EE	В основе сервиса лежит HAProxy, который реализует балансировку нагрузки
Chrony	Нет	CE/EE	Сервис времени. Обеспечивает синхронизацию часов на хостах кластера, что необходимо для его корректной работы. Установка Chrony не требуется, если на хостах уже используется аналогичный сервис
Etcd	Нет	EE	Версия ADPG Enterprise требует Etcd, чтобы задействовать его как распределенное хранилище конфигурации Patroni (Distributed Configuration Store, DCS). Вы можете использовать встроенный ADPG-сервис Etcd или внешний кластер etcd. Если вы хотите использовать внешний кластер, не добавляйте встроенный Etcd-сервис на вкладке ***Services***. Вместо этого укажите внешние настройки etcd на вкладке ***Primary configuration*** сервиса ADPG. Чтобы их задать, откройте ***Clusters*** → ***ADPG cluster*** → ***Services*** → ***ADPG*** → ***Primary configuration***, как описано в статье Настройка сервисов
Monitoring	Нет	CE/EE	Сервис для сбора и хранения метрик мониторинга кластера ADPG на основе системы мониторинга Prometheus и сервиса визуализации метрик Grafana

4. В результате успешно добавленные сервисы отображаются на вкладке *Services*.



Результат успешного добавления сервисов в кластер

ПРИМЕЧАНИЕ

Опциональные сервисы могут быть добавлены в кластер позднее. Процесс добавления сервисов в уже развернутый кластер не отличается от описанного выше.

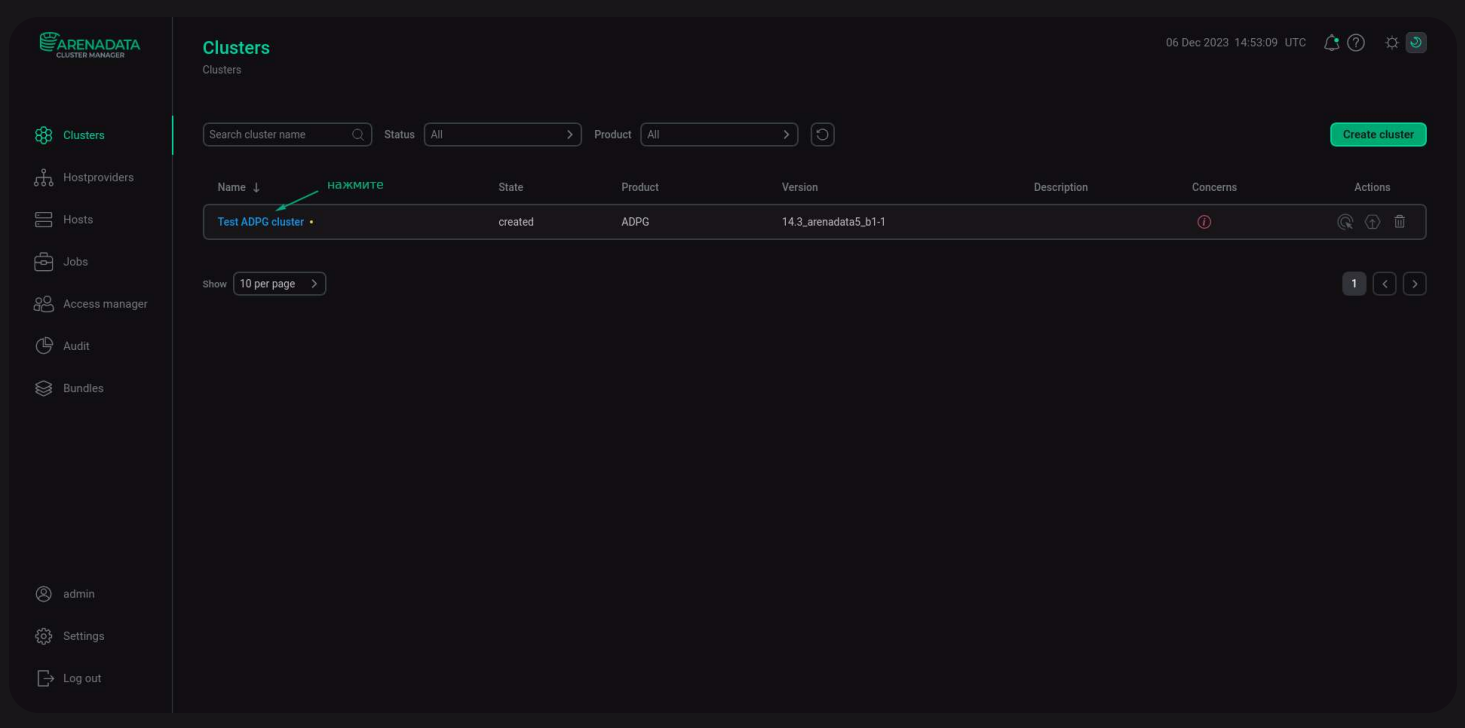
Добавление хостов в кластер

Евгения Кузина, Дарья Барышева

Назад к содержанию

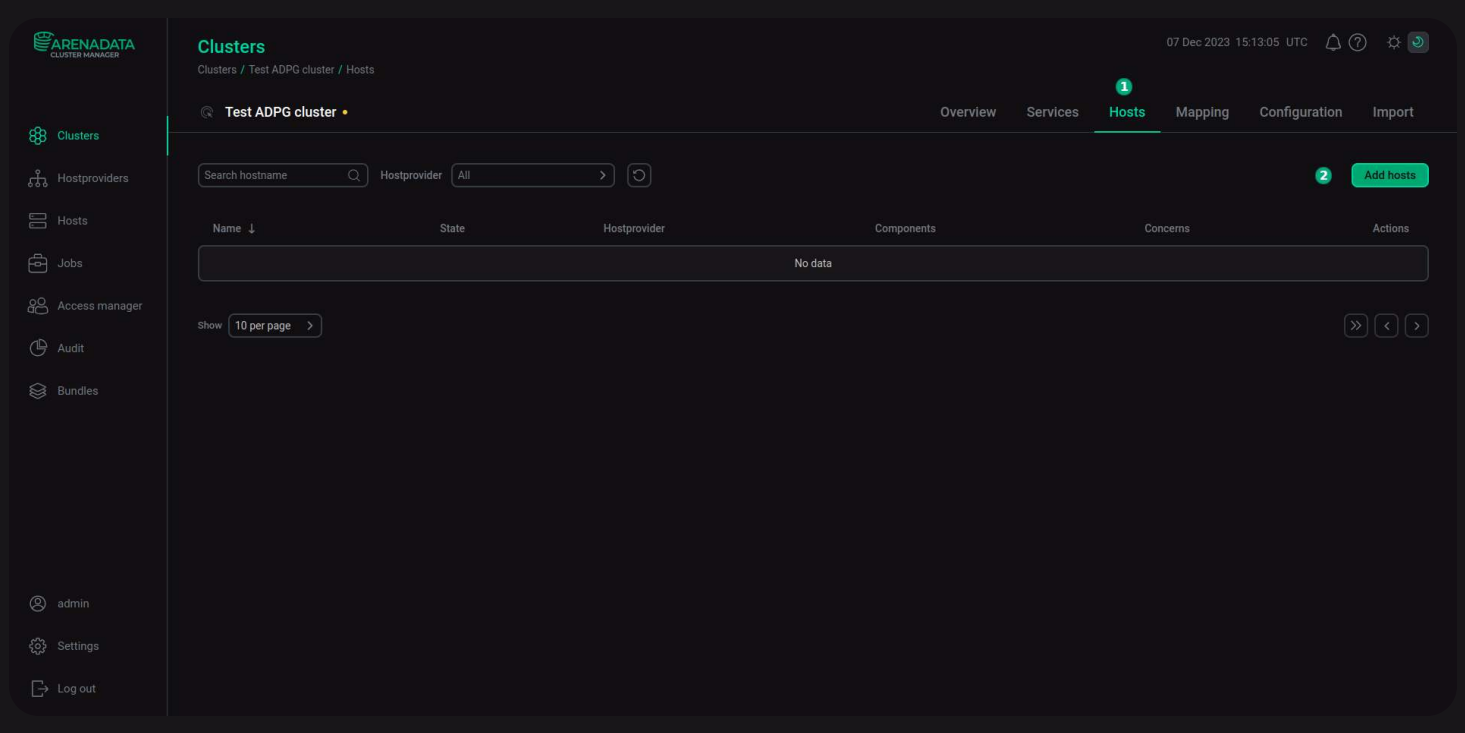
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



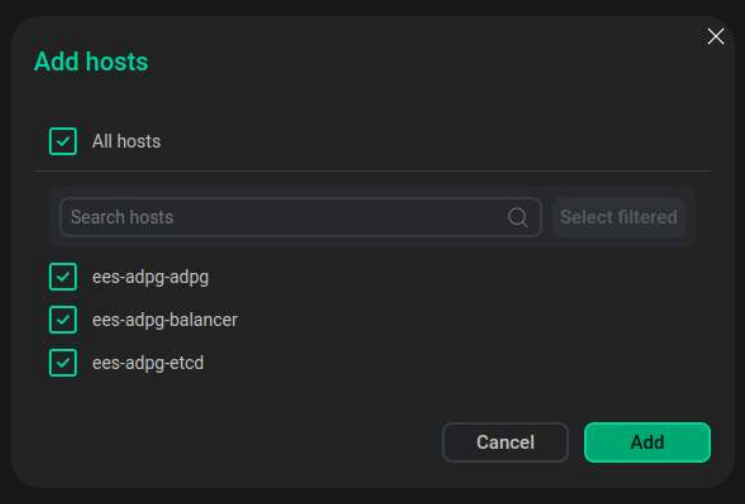
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



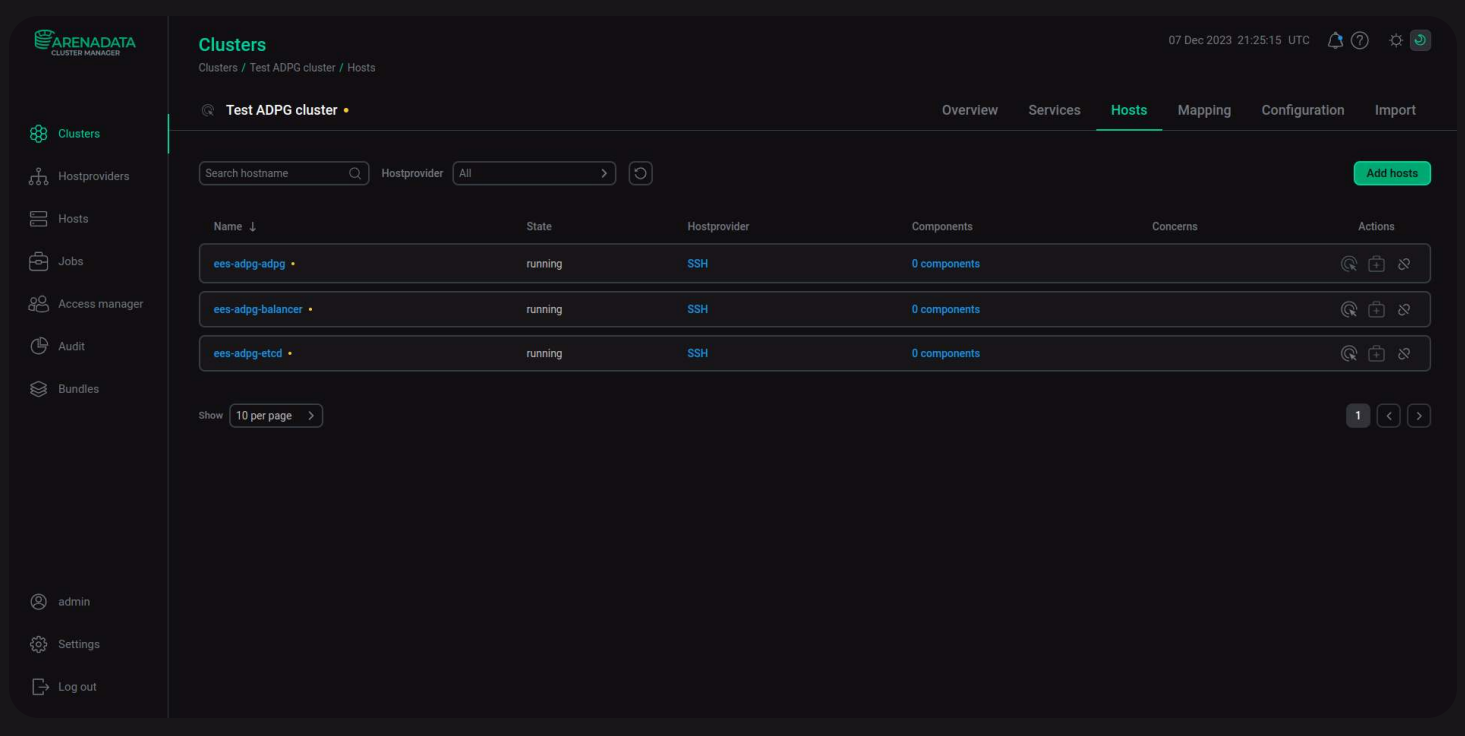
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Дарья Барышева, Евгений Кузина

Назад к содержанию

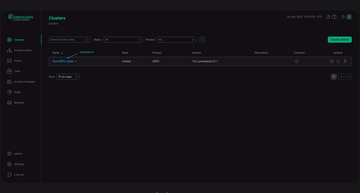
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

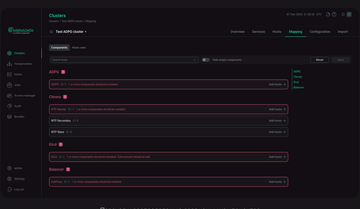
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



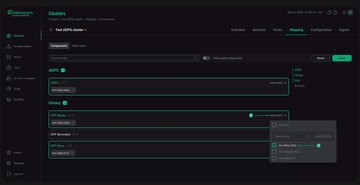
Выбор кластера

2. Откройте вкладку ***Mapping*** на странице кластера.



Переход к распределению сервисных компонентов

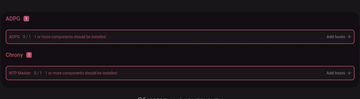
3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите ***Add hosts*** и выберите хосты в открывшемся окне.



Выбор хостов для компонента

Обязательные компоненты

Каждый сервис может иметь обязательные и опциональные компоненты. Обязательные подсвечиваются красным цветом. Также в интерфейсе ADCM для них выводится информация о том, в каком количестве необходимо установить каждый компонент.



Обязательные компоненты

В настоящий момент компоненты ADPG и NTP Master обязательны и должны быть размещены на одном хосте.

ВНИМАНИЕ

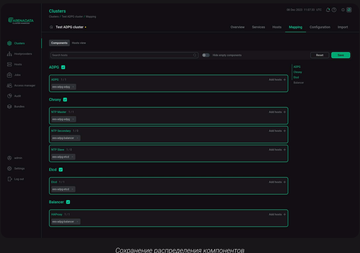
Без назначения нужного числа хостов обязательным компонентам сохранение общей схемы распределения невозможно.

Пример схемы распределения компонентов в кластере ADPG для версии Enterprise приведен в таблице ниже.

Распределение компонентов ADPG

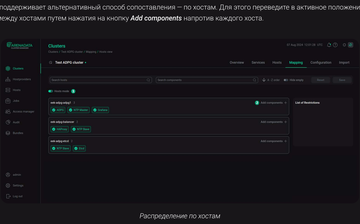
Хост	Сервис	Компонент	Примечание
Хост 1	ADPG	ADPG	Может быть лидером или репликой в архитектуре HA
	ADPG Control Agents	ADPG Control Agent	Агент, позволяющий управлять кластером ADPG с помощью ADPG Control
		ADPG Control ADBM Agent	Агент менеджера бэкапов, который использует ADPG Control
	Chrony	NTP Master	—
Хост 2	ADPG	ADPG	Может быть лидером или репликой в архитектуре HA
	ADPG Control Agents	ADPG Control Agent	Агент, позволяющий управлять кластером ADPG с помощью ADPG Control
		ADPG Control ADBM Agent	Агент менеджера бэкапов, который использует ADPG Control
	Chrony	NTP Secondary	—
Хост 3	ADPG	ADPG	Может быть лидером или репликой в архитектуре HA
	ADPG Control Agents	ADPG Control Agent	Агент, позволяющий управлять кластером ADPG с помощью ADPG Control
		ADPG Control ADBM Agent	Агент менеджера бэкапов, который использует ADPG Control
	Chrony	NTP Secondary	—
Хост 4	Etcd	Etcd	Необходимо использовать нечетное количество нод Etcd. Рекомендовано три и более
	Chrony	NTP Slave	—
Хост 5	Etcd	Etcd	Необходимо использовать нечетное количество нод Etcd. Рекомендовано три и более
	Chrony	NTP Slave	—
Хост 6	Etcd	Etcd	Необходимо использовать нечетное количество нод Etcd. Рекомендовано три и более
	Chrony	NTP Slave	—
Host 7	Balancer	HAProxy	—

4. После завершения распределения компонентов нажмите ***Save***.



Сохранение распределения компонентов

Начиная с версии 2.2.0 ADCM поддерживает альтернативный способ сопоставления — по хостам. Для этого переведите в активное положение переключатель ***Hosts mode*** и распределите компоненты между хостами путем нажатия на кнопку ***Add components*** напротив каждого хоста.



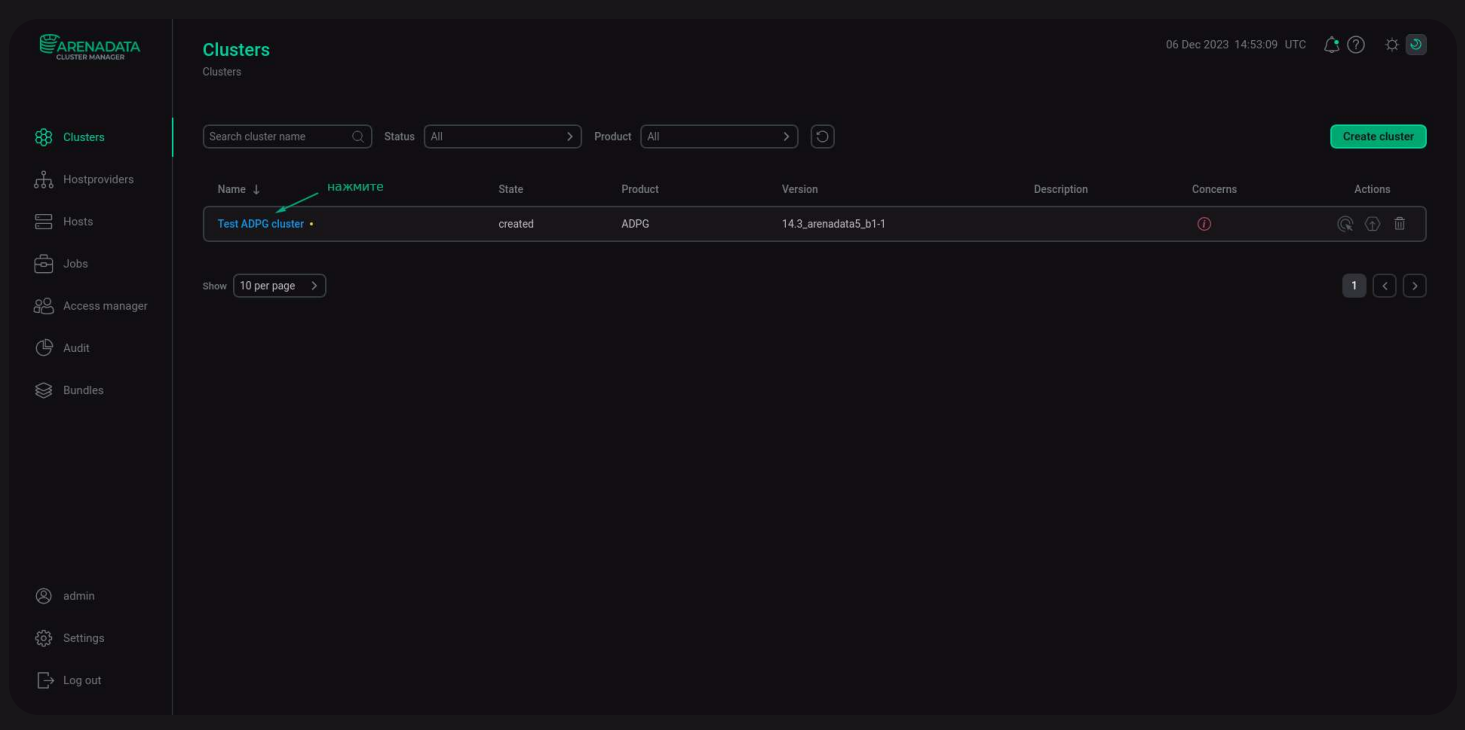
Распределение по хостам

Настройка сервисов

Дарья Барышева, Евгения Кузина

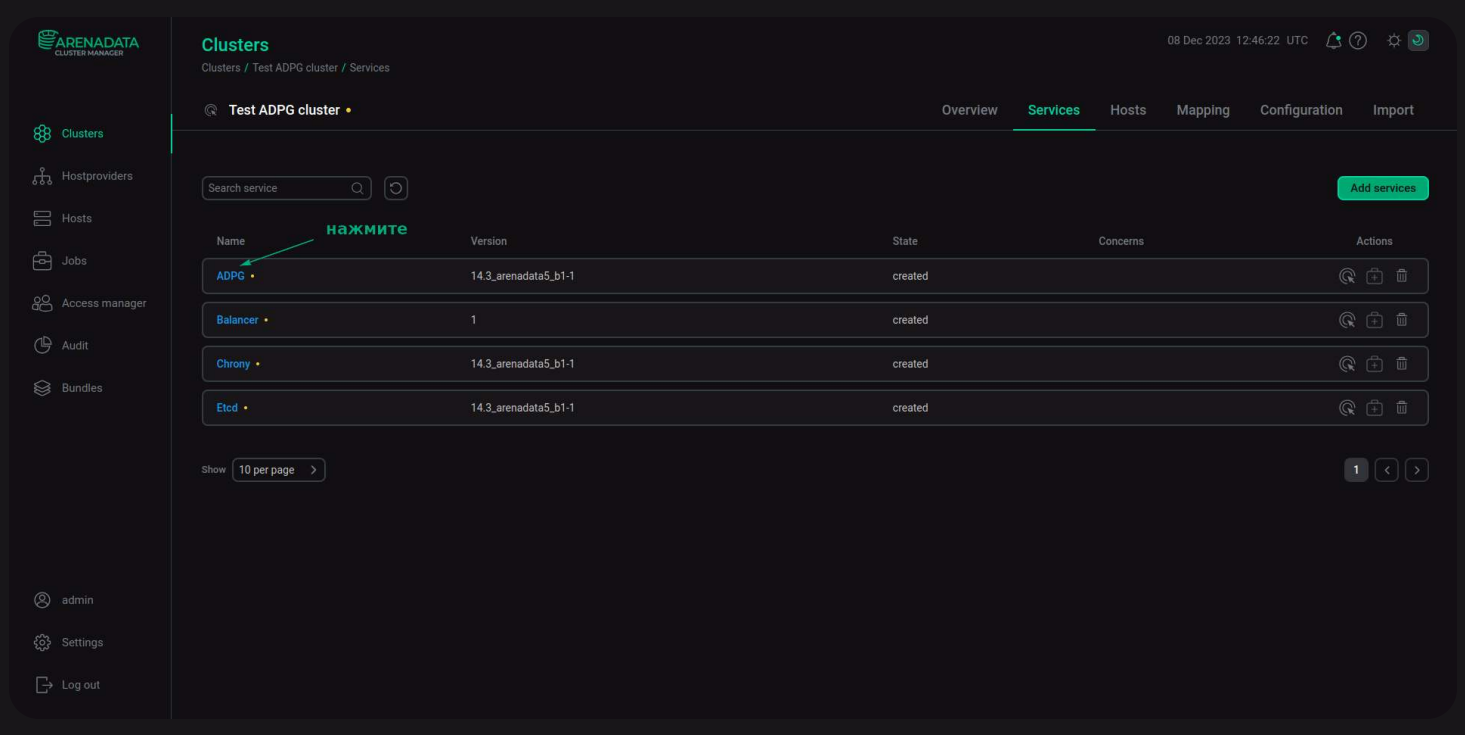
Назад к содержанию

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



Выбор кластера

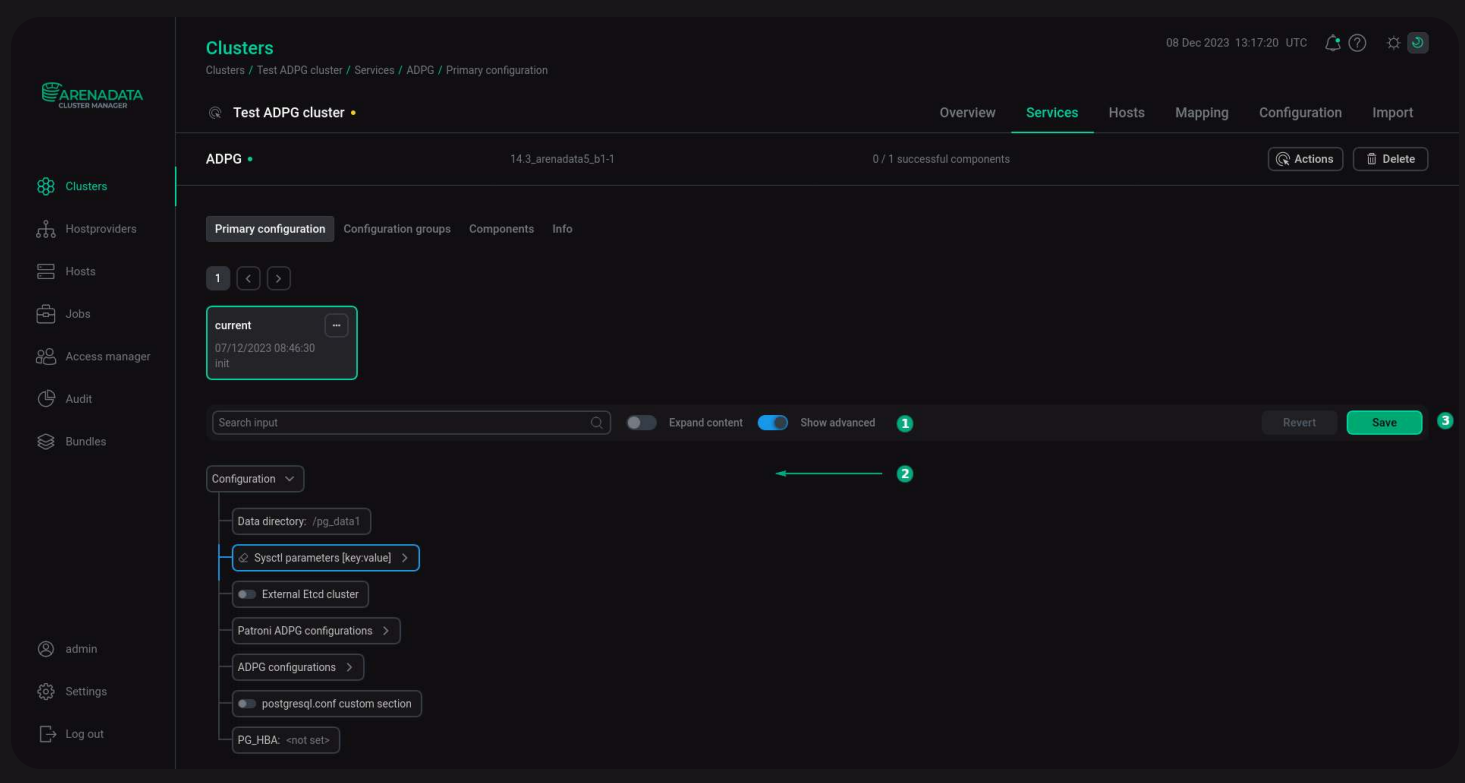
2. Откройте вкладку ***Services*** на странице кластера. Нажмите на имя сервиса в столбце ***Name***.



Переход к настройке сервиса

3. В открывшемся окне:

   - Откройте вкладку ***Primary Configuration***.
   - Переведите в активное состояние переключатель ***Show advanced***.
   - Заполните конфигурационные параметры выбранного сервиса. Поля, подсвеченные красным, обязательны для заполнения.
   - Нажмите ***Save***.



Настройка сервиса

Обратите внимание на следующие параметры:

- Сервис ***Chrony*** → Параметр ***NTP servers***. Хотя этот параметр необязателен, мы рекомендуем установить в нем адрес ближайшего NTP-сервера (например, используемого в вашей компании).

ПРИМЕЧАНИЕ

- Если вы планируете использовать компонент ADPG Control ADBM Agent сервиса ADPG Control Agents, убедитесь, что переключатель Enable backups выключен. Этот компонент представляет собой менеджер бэкапов, и использовать оба метода резервного копирования одновременно невозможно.
- Более подробную информацию по всем конфигурационным параметрам можно получить в статье Конфигурационные параметры.

Настройка кластера

Евгения Кузина, Дарья Барышева

Содержание

Назад к содержанию

Primary configuration

Configuration groups

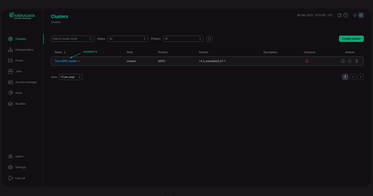
Ansible settings

ВАЖНО

В большинстве случаев кастомная настройка кластера не требуется — значения параметров можно оставить по умолчанию.

После добавления нового кластера можно выполнить его настройку. Для этого выполните шаги:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



Выбор кластера

2. Откройте вкладку ***Configuration*** на странице кластера. Вкладка ***Configuration*** состоит из трех подразделов: ***Primary configuration***, ***Configuration groups***, ***Ansible settings***.
3. Заполните необходимые конфигурационные параметры на нужной вкладке и нажмите ***Save***.

Primary configuration

Вкладка ***Configuration → Primary configuration*** содержит основные конфигурационные параметры кластера.

Конфигурационные параметры кластера перечислены ниже. В основном они описывают пути к репозиториям, которые будут использованы в процессе установки в зависимости от операционной системы кластера (ALT Linux 8, Astra Linux 1.7, CentOS 7 или Ubuntu 22.04). На серверах кластера регистрируются только отмеченные репозитории. Значения по умолчанию при необходимости могут быть изменены.

Основные конфигурационные параметры

Параметр	Описание
Cluster UUID	Уникальный идентификатор кластера
Disable firewall	Отключает **firewalld** на хостах кластера

Altlinux 8 repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

Astra linux repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

Centos 7 repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

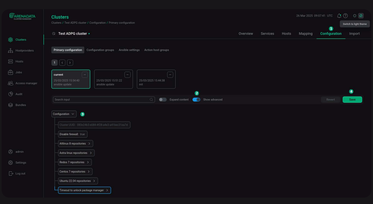
Ubuntu 22.04 repositories

Параметр	Описание
Set up ADPG repo	Необходимо ли использовать репозитории, указанные в полях `ADPG CORE repo url` и `ADPG repo url`
ADPG CORE repo url	Репозиторий ADPG CORE, необходимый для установки ADPG
ADPG repo url	Репозиторий ADPG, необходимый для установки ADPG
Set up ADPG Monitoring repo	Необходимо ли использовать репозиторий, указанный в поле `ADPG Monitoring repo url`
ADPG Monitoring repo url	Репозиторий ADPG Monitoring, необходимый для установки мониторинга

Timeout to unlock package manager

Раздел доступен при активации переключателя ***Show advanced***. Параметры этой секции предназначены для настройки тайм-аута ожидания разблокировки пакетного менеджера DNF. В Ubuntu после перезагрузки хоста запускается обновление, что приводит к блокировке DNF.

Параметр	Описание	Значение по умолчанию
retries	Максимальное число повторных попыток	3
delay	Тайм-аут между повторными попытками (в секундах)	10



Конфигурирование кластера

Configuration groups

Вкладка ***Configuration → Configuration groups*** предназначена для настройки конфиг-групп кластера.

Ansible settings

Вкладка ***Configuration → Ansible settings*** предназначена для ввода конфигурационных настроек Ansible на уровне кластера. Вкладка доступна начиная с версии ADCM 2.2.0.

Параметры Ansible

Имя	Описание	По умолчанию
forks	Число параллельных процессов, генерируемых при коммуникации с удаленными хостами	5

ПРИМЕЧАНИЕ

На этапе offline-установки параметры *Arenadata repo url* и *Monitoring repo url* игнорируются. Вместо этих репозиториев используются данные, сохраненные локально в кластере Enterprise Tools.

Импорт настроек ET

Дарья Барышева, Евгения Кузина

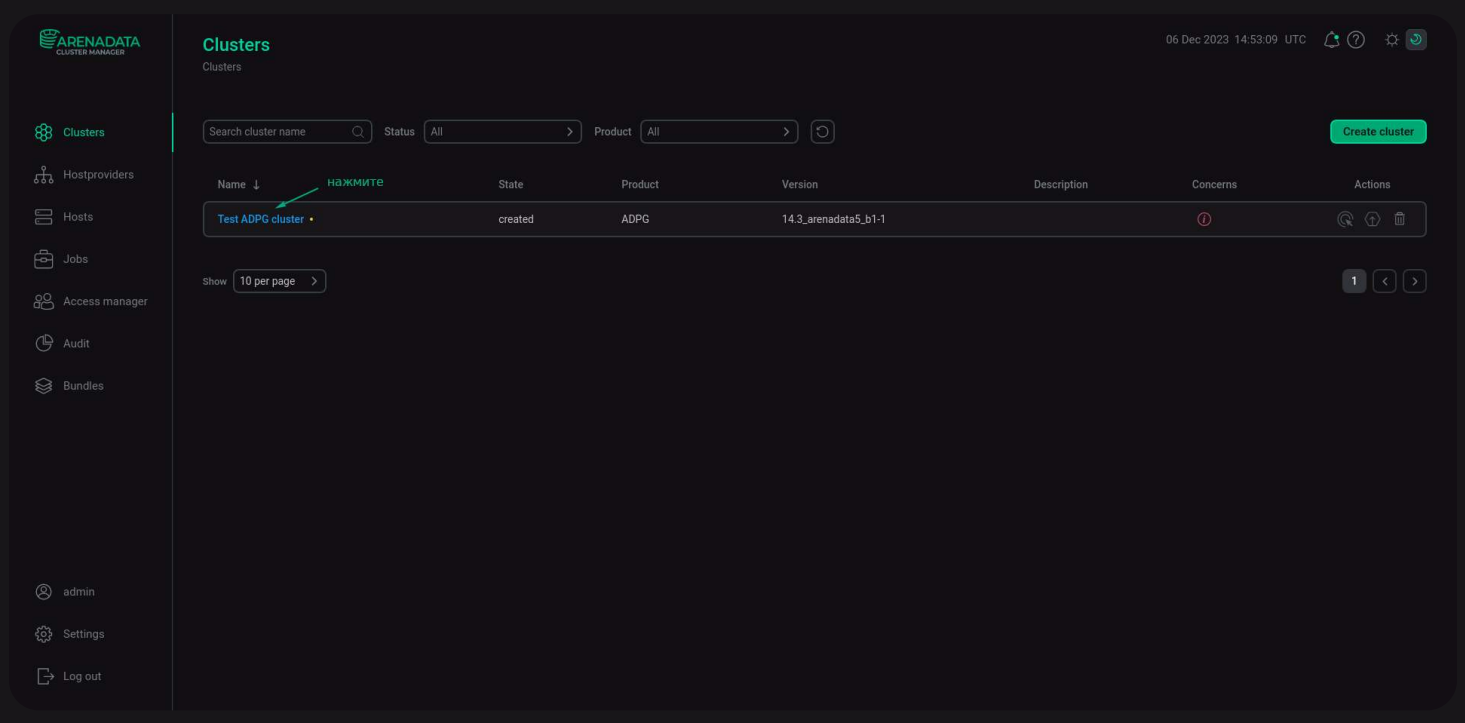
Назад к содержанию

ВАЖНО

Этот шаг необходим только в случае offline-установки.

Для установки кластера в режиме offline требуется интегрировать его с ранее созданным кластером Enterprise Tools (ET):

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.

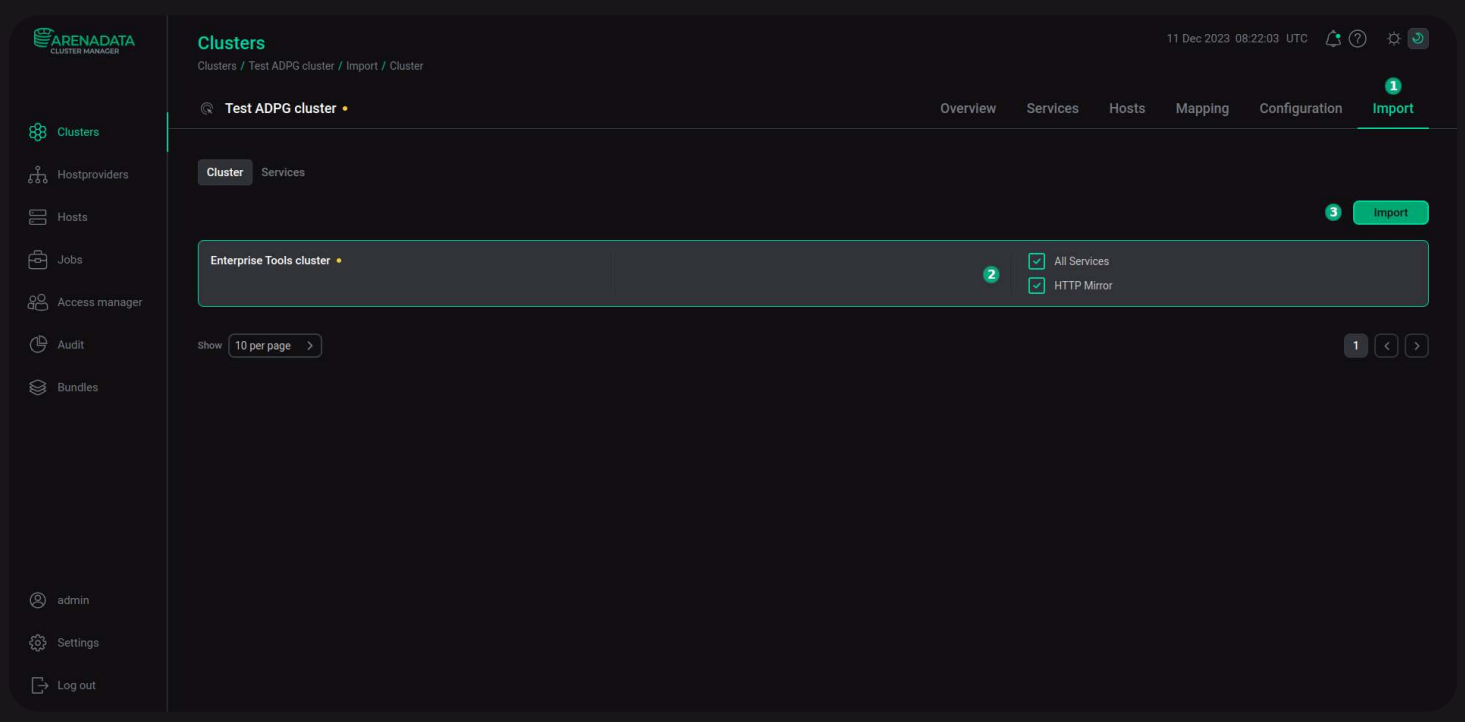


Выбор кластера

2. Откройте вкладку ***Import*** в меню кластера.

3. Выберите сервис ***HTTP Mirror*** предварительно созданного кластера ET.

4. Нажмите ***Import***.



Интеграция с кластером ET

ПРИМЕЧАНИЕ

После успешного импорта данные, хранимые локально в кластере Enterprise Tools, могут быть использованы на этапе установки продуктового кластера. Таким образом, их загрузка через Интернет не требуется.

Установка кластера

Документация по установке кластера PostgreSQL

AD.PG

Содержание

Назад к содержанию

Шаг 1. Предварительная проверка (Precheck)

Шаг 2. Запуск установки

- Установка всего кластера
- Установка отдельных серверов

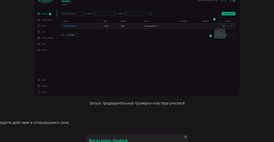
Шаг 3. Проверка статуса кластера после установки

Шаг 4. Подключение к PostgreSQL через psql

Шаг 1. Предварительная проверка (Precheck)

Перед установкой нового кластера рекомендуется выполнить его предварительную проверку – ***Precheck***. Проверка помогает убедиться в том, что все необходимые репозитории доступны, в кластере нет хостов с одинаковыми IP-адресами и все прочие требования соблюдены. Для запуска проверки:

1. Примените действие **Precheck** к кластеру, выбранному на странице **Clusters**, нажав на иконку ⚙ в столбце **Actions**.

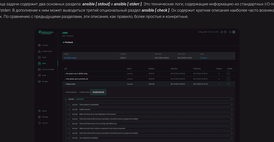


Запуск предварительной проверки кластера prechеck

2. Подтвердите действие в открывшемся окне:

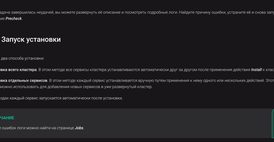
Подтверждение действия

3. Откройте страницу **Jobs**, чтобы увидеть результат действия **Precheck**. Если действие завершается успешно, соответствующая ему задача на странице **Jobs** переходит из статуса `PENDING` → `RUNNING` → `SUCCESS`. В случае ошибки – в статус `FAILED`. Чтобы получить более детальную информацию о возможном сообщении, можно перейти на страницу задачи, кликнув по ней в списке задач на странице **Jobs**.



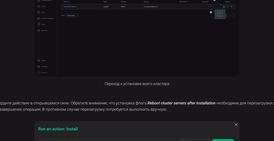
Переход к списку задач (jobs)

Откроется окно с шагами выполнения задачи. Кликните мышкой/тапом в столбце **Details**, чтобы получить дополнительную информацию об определенном шаге:



Переход на страницу с подробностями выполнения

Страница задачи содержит два основных раздела: **ansible [stdout]** и **ansible [stderr]**. Это технические логи, содержащие информацию из стандартных I/O-потоков utility-скриптов. В дополнение к ним может выводиться третий специализированный раздел **ansible [fatal]**. Он содержит краткое описание наиболее часто возникающих ошибок. По сравнению с предыдущими разделами, эти описания, как правило, более простые и конкретные.



Подробности выполнения задачи

4. Если задача завершилась неудачей, вы можете развернуть ее описание и посмотреть подробные логи. Найдите причину ошибки, устраните ее и снова запустите действие **Precheck**.

Шаг 2. Запуск установки

Существует два способа установки:

- **Установка всего кластера**. В этом методе все серверы кластера устанавливаются автоматически друг за другом после применения действия **Install** к кластеру.
- **Установка отдельных серверов**. В этом методе каждый сервер устанавливается вручную путем применения к нему одного или нескольких действий. Этот способ также можно использовать для добавления новых серверов в уже развернутый кластер.

В обоих методах каждый сервер запускается автоматически после установки.

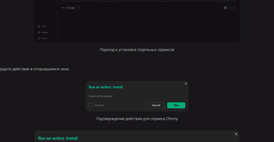
ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице **Jobs**.

Установка всего кластера

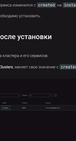
Чтобы установить весь сервера кластера за одну операцию, выполните шаги:

1. Примените действие **Install** к кластеру, выбранному на странице **Clusters**, нажав на иконку ⚙ в столбце **Actions**.



Переход к установке всего кластера

2. Подтвердите действие в открывшемся окне. Обратите внимание, что установку флага **Reboot cluster servers after installation** необходимо для перезагрузки системы после завершения операции. В противном случае перезагрузку потребуется выполнить вручную.



Подтверждение действия

3. Дождитесь завершения установки.
4. Если одна или несколько серверов не удается устранить, можно определить причину ошибок по логам, исправить их и перезапустить установку. Для этого необходимо выбрать действие **Reinstall** в столбце **Actions** на странице **Clusters**.

Перезапуск установки

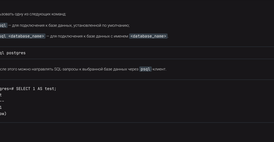
Установка отдельных серверов

Чтобы установить один или несколько серверов вручную, необходимо предварительно добавить эти серверы в кластер и затем применить к ним действия, описанные в подразделе выше.

ВНИМАНИЕ!

Обратите внимание, что серверы устанавливаются по очереди друг за другом. Не пытайтесь установить следующий сервер до окончания установки предыдущего.

1. Откройте вкладку **Services** на странице кластера. Для сервера, который должен быть установлен, нажмите на иконку ⚙ в столбце **Actions** и выберите действие **Install** из выпадающего списка.



Переход к установке отдельных серверов

2. Подтвердите действие в открывшемся окне:

Подтверждение действия для сервера Citrony

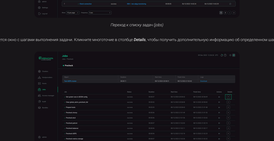
Подтверждение действия для всех серверов, кроме Citrony

3. Дождитесь завершения установки и убедитесь, что статус сервера изменился с `created` → `installed`.
4. Повторите предыдущие шаги для всех серверов, которые необходимо установить.

Шаг 3. Проверка статуса кластера после установки

В результате успешной установки произойдут обновления статуса кластера и его серверов:

- Статус кластера, отображаемый в столбце **State** на странице **Clusters**, начнет свое значение с `created` → `installed`



Статус кластера после успешной установки

- Статус серверов, отображаемый на вкладке **Services** страницы кластера, начнет свое значение с `created` → `installed`.

Статус серверов кластера после успешной установки

Шаг 4. Подключение к PostgreSQL через psql

Чтобы убедиться в успешной установке PostgreSQL и создании всех баз данных, необходимо:

1. Подключиться к серверу ADPG (например, через SSH) под пользователем `postgres`, который создается по умолчанию.

```
$ sudo su - postgres
```

2. Вывести список всех баз данных PostgreSQL.

```
$ psql -l
```

Результат команды приведен ниже.

```
                                  List of databases
   Name    | Owner  | Encoding |  Collate   |   Ctype    |  Access privileges
-----------+--------+----------+------------+------------+-----------------------
 postgres  | postgres | UTF8   | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8   | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |        |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8   | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |        |          |            |            | postgres=CTc/postgres
(3 rows)
```

3. Использовать одну из следующих команд:

- `psql` – для подключения к базе данных, установленной по умолчанию;
- `psql <database_name>` – для подключения к базе данных с именем `<database_name>`.

```
$ psql postgres
```

После этого можно направлять SQL запросы к выбранной базе данных через `psql` клиент.

```
postgres=# SELECT 1 AS test;
 test
------
    1
(1 row)
```

Установка мониторинга

Д. Евгения Кузина

Содержание

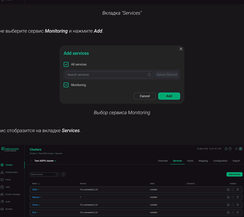
Назад к содержанию

Шаг 1. Добавление сервиса Monitoring
Шаг 2. Настройка мониторинга
Шаг 3. Установка сервиса Monitoring
Шаг 4. Просмотр дашбордов

Система мониторинга входит в бандл ADPG. Для установки мониторинга выполните шаги, описанные ниже.

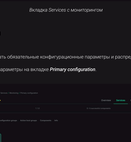
Шаг 1. Добавление сервиса Monitoring

Откройте страницу **Clusters** в пользовательском интерфейсе ADCM и выберите кластер ADPG. Откройте вкладку **Services** и нажмите **Add services**.



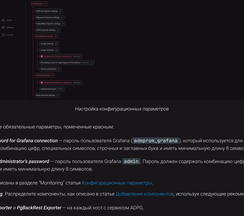
Вкладка "Services"

В открывшемся диалоговом окне выберите сервис **Monitoring** и нажмите **Add**.



Выбор сервиса Monitoring

В результате добавленный сервис отобразится на вкладке **Services**.

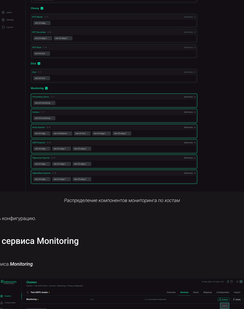


Вкладка Services с мониторингом

Шаг 2. Настройка мониторинга

Иконка ⓘ **Concerns** показывает, что необходимо указать обязательные конфигурационные параметры и распределить компоненты мониторинга по хостам.

Кликните сервис **Monitoring** и установите необходимые параметры во вкладке **Primary configuration**.



Настройка конфигурационных параметров

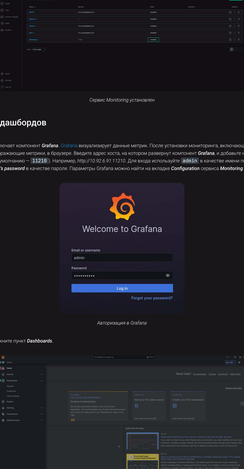
Необходимо указать следующие обязательные параметры, помеченные красным:

- **Prometheus settings** → **Password for Grafana connection** — пароль пользователя Grafana (`admin_grafana`), который используется для подключения к Prometheus. Пароль должен содержать комбинацию цифр, специальных символов, строчных и заглавных букв и иметь минимальную длину 8 символов.
- **Grafana settings** → **Grafana administrator's password** — пароль пользователя Grafana (`admin`). Пароль должен содержать комбинацию цифр, специальных символов, строчных и заглавных букв и иметь минимальную длину 8 символов.

Все параметры мониторинга описаны в разделе "Monitoring" статьи Конфигурационные параметры.

Затем откройте вкладку **Mapping**. Распределите компоненты, как описано в статье Добавление компонентов, используя следующие рекомендации:

- **ADPG Exporter**, **Pgbouncer Exporter** и **Pgbackrest Exporter** — на каждый хост с сервисом ADPG.
- **Prometheus Server** и **Grafana** — на отдельный хост.
- **Node Exporter** — на каждый хост, с которого нужно собирать метрики операционной системы.

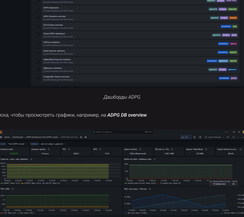


Распределение компонентов мониторинга по хостам

Нажмите **Save**, чтобы сохранить конфигурацию.

Шаг 3. Установка сервиса Monitoring

Выполните действие **Install** сервиса **Monitoring**.



Действие "Install" сервиса Monitoring

В следующем диалоговом окне нажмите **Run** для подтверждения действия. Вы также можете активировать флажок **Verbose**, чтобы увидеть дополнительные сведения о выполнении действия на странице **Jobs**.

Подтверждение действия "Install"

В результате статус сервиса Monitoring изменится на `Installed`.

Сервис Monitoring установлен

Шаг 4. Просмотр дашбордов

Система мониторинга в ADPG включает компонент **Grafana** (версия 9.5.2), визуализирует данные метрик. После установки мониторинга, включающего компонент **Grafana**, можно просматривать дашборды, отображающие метрики, в браузере. Введите адрес хоста, на котором развернут компонент **Grafana**, и добавьте номер порта — параметр **Grafana listen port** (значение по умолчанию — `11210`). Например, *http://10.92.9.91:11210*. Для входа используйте `admin` в качестве имени пользователя и значение пароля из **Grafana administrator's password** в качестве пароля. Параметры Grafana можно найти на вкладке **Configuration** сервиса **Monitoring**.

Авторизация в Grafana

Разверните меню Grafana и кликните пункт **Dashboards**.

Интерфейс Grafana

Откроется страница со списком дашбордов мониторинга. Разверните элемент **ADPG Dashboard**, для вашего кластера, чтобы отобразить доступные дашборды.

Дашборды ADPG

Нажмите на любой элемент списка, чтобы просмотреть графики, например, на **ADPG DB overview**.

Графики ADPG DB overview

Подключение к ADPG

Евгения Кузина

Назад к содержанию

Вы можете использовать следующие клиентские инструменты для подключения к ADPG:

- Стандартную утилиту PostgreSQL — psql
- Сторонние клиентские инструменты:
  - pgAdmin
  - DBeaver
  - другие

ПРИМЕЧАНИЕ

После установки ADPG вы можете использовать `psql`. Если необходимо задействовать сторонние инструменты, установите и настройте их в соответствии с вашими требованиями.

Использование стандартной утилиты PostgreSQL — psql для подключения к ADPG

Евгения Кузина

Назад к содержанию

Для подключения к ADPG можно использовать стандартное клиентское приложение PostgreSQL — psql. Это интерактивный терминал, который позволяет выполнять SQL-запросы из командной строки и из SQL-файлов и просматривать их результаты. Утилита `psql` доступна на каждой ноде кластера ADPG.

Чтобы начать работу с `psql`, выполните следующие команды:

```
$ sudo su - postgres
```

```
$ psql
```

Первая команда переключает терминал в режим работы с правами доступа для пользователя `postgres`. Вторая — запускает `psql` для базы данных по умолчанию (`postgres`).

Вы также можете использовать опции `psql` из таблицы ниже, чтобы задать базу данных, пользователя, хост и порт для подключения вместо значений по умолчанию.

Опции psql

Опция	Описание
-d <имя_базы_данных> --dbname=<имя_базы_данных>	Задает имя базы данных для подключения
-h <имя_хоста> --host=<имя_хоста>	Определяет имя хоста, на котором расположен сервер
-p <порт> --port=<порт>	Определяет TCP-порт, через который сервер принимает подключения. Значение по умолчанию — `5432`
-U <имя_пользователя> --username=<имя_пользователя>	Задает пользователя, от имени которого произойдет подключение

Пример:

```
$ psql -h 10.92.6.36 -p 5432 -U user1 -d postgres
```

РЕКОМЕНДАЦИЯ

Если у вас возникли проблемы с подключением, используйте утилиту pg_isready, чтобы проверить состояние подключения.

После выполнения команды `psql` терминал отображает приглашение `psql` с именем текущей базы данных и символом `#` в конце. Все последующие команды следует писать после `#`.

```
psql (14.3)
Type "help" for help.

postgres=#
```

Примеры выполнения SQL-запросов из командной строки вы можете найти в статье Пример работы с таблицами через psql.

Чтобы запустить SQL-запрос из файла, используйте метакоманду `\i`:

```
\i /путь/к/файлу/имя_файла.sql
```

Чтобы выйти из `psql`, используйте следующую команду:

```
\q
```

В таблице ниже перечислены некоторые полезные метакоманды.

Метакоманды psql

Команда	Описание
`\c <имя_БД>`	Устанавливает соединение с определенной базой данных
`\l`	Выводит список всех баз данных на сервере и показывает их имена, владельцев, кодировку и права доступа
`\dt`	Выводит список всех таблиц в текущей базе данных
`\dt+`	Показывает список всех таблиц в текущей базе данных с их статусом хранения (`permanent`, `temporary` или `unlogged`), физическим размером на диске и описанием
`\dn`	Выводит список всех схем в текущей базе данных
`\d+ <имя_таблицы>`	Показывает детальную информацию о таблице
`\dv`	Выводит список всех представлений в текущей базе данных
`\df`	Показывает список функций с типами данных их результатов, аргументов и классификацией: `agg` (агрегатная), `normal` (обычная), `procedure` (процедурная), `trigger` (триггерная) или `window` (оконная)
`\x`	Устанавливает режим развернутого вывода таблицы (expanded table formatting mode)
`\du`	Выводит список всех пользователей (ролей)

Метакоманды выполняют запросы к системным таблицам и представлениям, чтобы получить требуемую информацию. Можно просматривать эти запросы, присвоив переменной `ECHO_HIDDEN` значение `on`:

```
\set ECHO_HIDDEN on
```

Пример:

```
postgres=# \set ECHO_HIDDEN on
postgres=# \dt
********* QUERY **********
SELECT n.nspname as "Schema",
  c.relname as "Name",
  CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized view' WHEN 'i' THEN 'index' WHEN 'S' THEN
'sequence' WHEN 's' THEN 'special' WHEN 't' THEN 'TOAST table' WHEN 'f' THEN 'foreign table' WHEN 'p' THEN 'partitioned table'
WHEN 'I' THEN 'partitioned index' END as "Type",
  pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
     LEFT JOIN pg_catalog.pg_am am ON am.oid = c.relam
WHERE c.relkind IN ('r','p','')
      AND n.nspname <> 'pg_catalog'
      AND n.nspname !~ '^pg_toast'
      AND n.nspname <> 'information_schema'
  AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
**************************

            List of relations
 Schema  |       Name       | Type  |  Owner
---------+------------------+-------+----------
 public  | author           | table | postgres
 public  | book             | table | postgres
 public  | us_gaz           | table | postgres
 public  | us_lex           | table | postgres
 public  | us_rules         | table | postgres
 public  | users            | table | postgres
(6 rows)
```

Чтобы отключить вывод запросов вместе с результатом метакоманды, присвойте переменной `ECHO_HIDDEN` значение `off`:

```
\set ECHO_HIDDEN off
```

Использование pgAdmin для подключения к ADPG

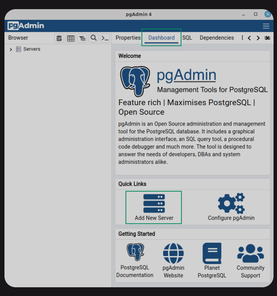
Евгения Кузина

Назад к содержанию

Для подключения к ADPG можно использовать pgAdmin — сторонний клиентский инструмент с графическим пользовательским интерфейсом.

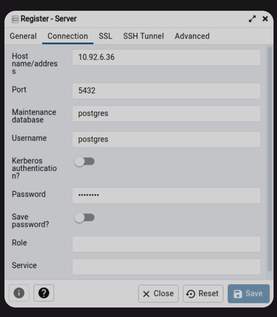
Информацию о том, как установить pgAdmin, можно найти по ссылке Download.

Чтобы подключиться к хосту ADPG, запустите pgAdmin, откройте вкладку ***Dashboard*** и кликните ***Add New Server*** в разделе Quick Links.



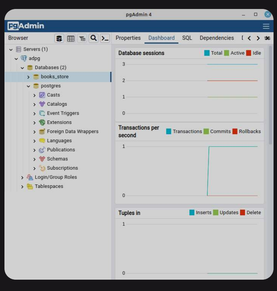
Вкладка Dashboard в pgAdmin

Укажите параметры хоста ADPG для подключения.



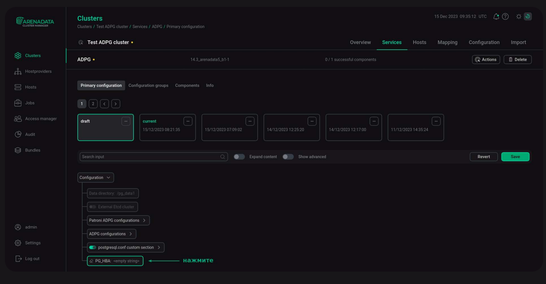
Настройка подключения в pgAdmin

После установки соединения pgAdmin отображает указанную базу данных в окне Browser.



Окно Browser в pgAdmin

Если вам нужно изменить параметры аутентификации в pg_hba.conf для подключения через pgAdmin, используйте пользовательский интерфейс ADCM. Откройте страницу ***Clusters***, выберите кластер ADPG и переключитесь на вкладку ***Services***. Выберите сервис ***ADPG***, кликните опцию ***PG_HBA*** на вкладке ***Primary configuration*** и отредактируйте значение.

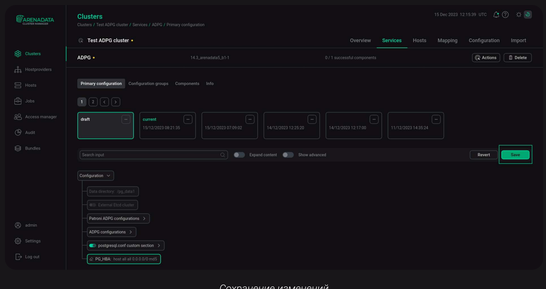


Редактирование раздела PG_HBA

В появившемся окне добавьте запись в файл pg_hba.conf и нажмите ***Apply***.

Добавление записи PG_HBA

Чтобы сохранить изменения, кликните ***Save***.



Сохранение изменений

Выполните действие Reconfigure & Restart, чтобы применить изменения.

ВНИМАНИЕ

Мы не рекомендуем редактировать *postgresql.conf* через pgAdmin. Когда сервис ADPG выполняет действие Reconfigure & Restart, *postgresql.conf* перезаписывается с настройками, заданными на вкладке ***Primary configuration*** сервиса ADPG. Все изменения, внесенные другими способами, будут потеряны.

Для получения дополнительной информации о pgAdmin обратитесь к официальной документации.

Использование DBeaver для подключения к ADPG

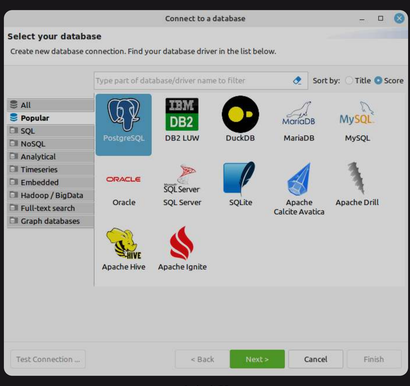
Евгения Кузина

Назад к содержанию

Для подключения к ADPG можно использовать DBeaver — сторонний клиентский инструмент, который поддерживает различные типы баз данных.

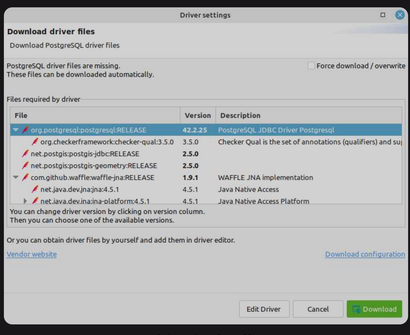
Информацию о том, как установить DBeaver, можно найти по ссылке Installation.

Чтобы подключиться к хосту ADPG, запустите DBeaver и выберите PostgreSQL в окне ***Connect to a database***.



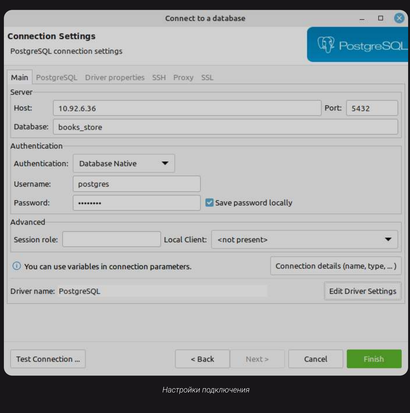
Выбор СУБД

Если вы подключаетесь к серверу PostgreSQL в первый раз, выберите драйвер.



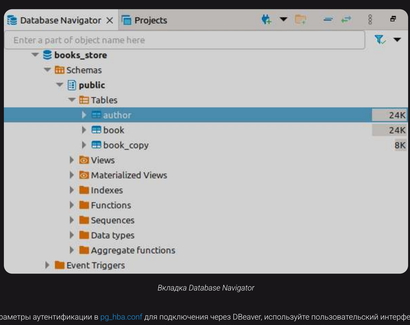
Выбор драйвера PostgreSQL

В следующем окне укажите настройки подключения к хосту ADPG.



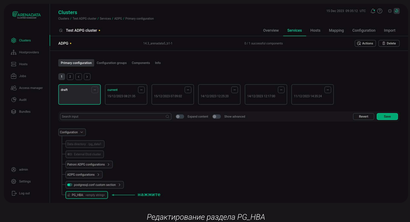
Настройки подключения

После установки соединения DBeaver отображает указанную базу данных на вкладке ***Database Navigator***.



Вкладка Database Navigator

Если вам нужно изменить параметры аутентификации в pg_hba.conf для подключения через DBeaver, используйте пользовательский интерфейс ADCM. Откройте страницу ***Clusters***, выберите кластер ADPG и переключитесь на вкладку ***Services***. Выберите сервис ***ADPG***, кликните опцию ***PG_HBA*** на вкладке ***Primary configuration*** и отредактируйте значение.

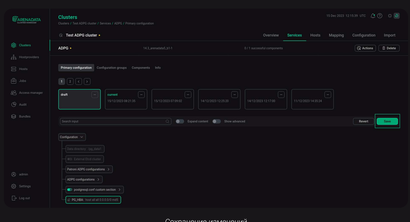


Редактирование раздела PG_HBA

В появившемся окне добавьте запись в файл pg_hba.conf и нажмите ***Apply***.

Добавление записи PG_HBA

Чтобы сохранить изменения, кликните ***Save***.



Сохранение изменений

Выполните действие Reconfigure & Restart, чтобы применить изменения.

Для получения дополнительной информации о DBeaver обратитесь к официальной документации — About DBeaver.

Аутентификация в ADPG

Евгения Кузина

Назад к содержанию

Аутентификация — это процесс идентификации пользователя, который собирается установить соединение с сервером. Сервер определяет разрешено ли клиентскому приложению или пользователю, запускающему клиентское приложение, подключаться к серверу с указанным именем. ADPG использует роли для управления правами доступа. В терминологии PostgreSQL пользователь — это роль с привилегией LOGIN.

Роли PostgreSQL отделены от пользователей операционной системы. Сервер баз данных может устанавливать удаленные соединения с пользователями (ролями PostgreSQL), не имеющими учетной записи в локальной операционной системе.

ADPG поддерживает следующие методы аутентификации: trust, по паролю, GSSAPI, SSPI, ident, peer, LDAP, RADIUS, по сертификату, PAM и BSD.

Обзор конфигурации PG_HBA

Время чтения: 5 минут

Содержание

- Формат PG_HBA
- Поля записи аутентификации
  - Connection type
    - database
    - user
    - address
    - IP-address и IP-mask
    - auth-method
    - auth-option
    - Включение внешних файлов
- Просмотр записей pg_hba.conf
- Поле PG_HBA

Назад к содержанию

В КОРУС套件 используется для настройки аутентификации клиента, конфигурационный файл pg_hba.conf в PostgreSQL. Поле **PG_HBA** содержит записи аутентификации и чем их формировать, что есть pg_hba.conf.

ВНИМАНИЕ!

Не рекомендуется редактировать файл pg_hba.conf. Когда кластер ACPO выключает действие Patroni в Astra, pg_hba.conf перезаписывается из служебных настроек и не учитывает поле **PG_HBA**, находящееся в кластере **Primary configuration** кластера ACPO.

Формат PG_HBA

Поле **PG_HBA** должно содержать набор записей с правилами аутентификации, по одному на каждый строке. Запись состоит из нескольких разделенных пробелами или табуляцией полей. Поля могут содержать пробелы, если значение заключено в кавычки. Записи не могут быть продолжены на несколько строк. Пустые строки игнорируются, как и любой текст после символа решетки (#), обозначающего начало комментария (1).

Каждая запись может иметь один из семи форматов, показанных ниже. Значение *hostnossl* может быть также записано как *hostno ssl*, с пробелом. Для совместимости с предыдущими версиями также принимается формат *local* (2), который может использоваться только для соединений Unix-domain.

```
local      database  user  auth-method  [auth-options]
host       database  user  address       auth-method  [auth-options]
hostssl    database  user  address       auth-method  [auth-options]
hostnossl  database  user  address       auth-method  [auth-options]
hostgssenc database  user  address       auth-method  [auth-options]
hostnogssenc database user address       auth-method  [auth-options]
host       database  user  IP-address  IP-mask      auth-method  [auth-options]
hostssl    database  user  IP-address  IP-mask      auth-method  [auth-options]
hostnossl  database  user  IP-address  IP-mask      auth-method  [auth-options]
hostgssenc database  user  IP-address  IP-mask      auth-method  [auth-options]
hostnogssenc database user IP-address  IP-mask      auth-method  [auth-options]

include    file
include_if_exists file
include_dir directory
```

Для поля **database** и **user** можно указать несколько имен, разделенных запятыми. Для поля **address** можно указать IP-адрес, диапазон IP-адресов или имя хоста. Для поля **IP-address** и **IP-mask** можно указать IP-адрес и маску. Для поля **auth-method** можно указать имя метода аутентификации. Для поля **auth-option** можно указать параметры, которые могут быть использованы с выбранным методом аутентификации.

Если поле **database** = **all** имеено указать пользователя через имена файла **@**. Этот файл может содержать имена баз данных или пользователей, разделенные пробелами или переводами строк. Имена групп должны быть указаны с предшествующим символом +. Например, +admins ищет в файле admins имена групп, определенные в нем. Чтобы включить имя, которое содержит пробелы или @, заключите имя в кавычки. Ссылка на файл может быть вложена, то есть один файл может включать другой. Ключевое слово **include** включает в себя один файл, а **include_if_exists** делает то же самое, но не выдает ошибку, если файл не существует. **include_dir** включает все файлы, которые имеют суффикс .conf в указанном каталоге.

Поля записи подробно описаны ниже:

local

Соответствует подключениям, которые используют доменные сокеты Unix. Без записи этого типа подключения через сокеты Unix система не допускаются.

host

Соответствует подключениям по TCP/IP. Запись **SSLKEY** соответствует как обычным, так и подключениям, защищенным SSL с помощью сертификатов клиентов. Для использования SSL необходимо предварительно настроить сервер (см. раздел «Настройка SSL»).

hostssl

Соответствует подключениям через TCP/IP, которые не используют SSL.

hostgssenc

Соответствует подключениям, выполненным по протоколу TCP/IP, если подключение устанавливается с шифрованием GSSAPI.

hostnogssenc

Соответствует подключениям по протоколу TCP/IP, не использующим шифрование GSSAPI.

database

Поле **database** указывает имена баз данных, которым соответствует запись. Это поле принимает следующие значения:

- Имя определенной базы данных или регулярное выражение, соответствующее имени базы данных. Можно использовать регулярные выражения и/или имя базы данных, начинающееся с символа +. Регулярные выражения должны начинаться с ~ и соответствовать полному имени базы данных, например, **~mydb[0-9]+$** соответствует любому имени базы данных с именем, оканчивающимся на **[0-9]**, и **~^mydb[0-9]+$** соответствует любому имени базы данных с именем, начинающимся с **[0-9]**.
- **all** — все базы данных.
- **sameuser** — имя базы данных должно быть идентично имени роли, если указанный пользователь. Например, если пользователь **postgres** — имя пользователя только с базы данных **postgres**.
- **samerole** — указывает, что пользователь должен быть членом роли с тем же именем, что и имя базы данных. Например, если пользователь **postgres** — имя пользователя только с базы данных **postgres**.
- **replication** — указывает, что запись соответствует запросам репликации (независимо от базы данных). Значение **replication** не соответствует обычным базам данных.
- Отдельный файл, содержащий имена баз данных и/или регулярные выражения. Его имени указать, чтобы перед именем файла **@**.

Например, приведенное ниже содержимое файла **user** соответствует **admin**, подключается к базам данных **blog** и **author**:

```
# type  database  user  auth-method  auth-option
host    blog,author  admin  0.0.0.0/0  md5
```

user

Поле указывает, какие имена пользователей будут соответствовать записи. Это поле допускает следующие значения:

- Имя определенной роли (или пользователя) для конкретного пользователя. Начинающегося с символа **+**, которое определяет имя роли. Вы можете добавить несколько имен пользователей или ролей, разделенных запятыми. Вы также можете указать **@**, чтобы указать файл, содержащий имена пользователей.
- **samerole** — указывает, что пользователь должен быть членом роли с тем же именем, что и имя базы данных. **samerole** — устаревшее название для этого значения.
- **replication** — указывает, что запись соответствует запросам репликации (независимо от базы данных). Значение **replication** не соответствует обычным базам данных.
- **all** — все роли (пользователи).

Например, следующий параметр имени пользователя для **access_stat**:

```
# type  database  user  auth-method  auth-option
host    access_stat  +support  0.0.0.0/0  md5
```

Приведенная ниже конфигурация позволит **support** подключаться к базе данных **access_stat**.

address

В этом поле указывается адрес хоста клиента, которому должно соответствовать запись. Оно может содержать как имена, так и IP-адреса или адреса со спецификациями маски подсети. Разрешены следующие форматы:

- Диапазон IP-адресов, указанный с помощью нотации CIDR.
- **all** — любой IP-адрес.
- **samehost** — любой из IP-адресов сервера.
- **samenet** — любой адрес в подсети сервера подключен напрямую.

Если есть имя, любое значение, которое не является IP-адресом, IP-маской или специальным ключевым словом, рассматривается как имя хоста, которое должно соответствовать обратному DNS-запросу IP-адреса клиента (например, **client.foo.com**). Сопоставление имени хоста нечувствительно к регистру. Если имя начинается с точки (.), оно соответствует суффиксу имени хоста, поэтому **.example.com** будет соответствовать **host.example.com**, но **host.example.com**.

Привязанные имена преобразуются в IP-адреса во время запуска, а не во время проверки. Любые имена, имеющие несколько адресов, могут работать.

Приведенная ниже конфигурация позволит **admin** подключаться к базе данных **blog** с IP-адреса **0.0.0.0/0**.

```
# type  database  user  address  auth-method
host    all       all   0.0.0.0/0  trust
```

Для IPv6:

```
# type  database  user  address  auth-method
host    all       all   ::1/128    trust
```

С именем хоста (обычно оказывает как IPv4, так и IPv6):

```
# type  database  user  address  auth-method
host    all       all   localhost  trust
```

Разрешает подключение любому пользователю с любого хоста с IP-адресом **192.168.54.1**:

```
# type  database  user  address          auth-method
host    all       all   192.168.54.1/24  trust
```

IP-address и IP-mask

Эти два поля можно использовать в качестве альтернативы записи IP-адрес/длина маски CIDR. Вместо указания длины маски фактическая маска задается в отдельном столбце. Например, **255.0.0.0** представляет собой IPv4 CIDR-маску длиной **8**, а **255.255.255.255** представляет собой CIDR-маску длиной **32**.

Эти поля не могут быть указаны для записей **local**.

Приведенная ниже конфигурация будет соответствовать **admin** подключение к базе данных **blog** с хоста **192.168.54.1**:

```
# type  database  user  IP-address  IP-mask          auth-method
host    all       all   127.0.0.1   255.255.255.255  trust
```

auth-method

Поле **auth-method** определяет метод аутентификации, который следует использовать для подключения. Доступны следующие варианты аутентификации:

Метод **trust** не выполняет аутентификацию. Имеет смысл использовать **trust**, только для локальных подключений. Метод **reject** блокирование аутентификации. Полезно для «фильтрации» определенных хостов из группы. Например, запись может отклонить подключение с IP-адреса, прежде чем позволить другим записям в том же диапазоне разрешить его.

Методы аутентификации **scram-sha-256**, **md5**, **ldap**, **radius** и **pam** используют пользователя пароль в процессе его аутентификации. Пароли, специфичные для SQL, хранятся в **pg_authid**.

Методы **peer**, **gss**, **sspi** и **ident** обеспечивают внешнюю аутентификацию. В результате успешной аутентификации ОС/ОС PostgreSQL получит указанное при подключении имя пользователя SQL, и подключение будет разрешено, если пользователь имеет право подключаться (т. е. существует любая соответствующая запись), дополнительный параметр **map** и **auth-option**. Параметр **map** определяет правила сопоставления внутреннего и внешнего имен.

Следующая конфигурация позволит всем пользователям подключаться к базе данных **all** с компьютера, работающего по IP-адресу **192.168.54.1**, если они предоставят правильный пароль:

```
# type  database  user  address          auth-method
hostssl   all     all   0.0.0.0/0  cert
```

auth-options

После **auth-option** определяет параметры для указанного в auth-method. Параметры следует добавлять в следующем формате: **имя=значение**.

Конфигурация **ldap** может иметь несколько параметров аутентификации. Параметр **ldapserver** определяет имя сервера LDAP, к которому нужно подключиться. Параметр **ldapport** определяет порт, который будет использоваться для подключения к LDAP-серверу. Параметр **ldapprefix** определяет строку, которая будет добавлена к имени пользователя перед выполнением поиска LDAP. Параметр **ldapsuffix** определяет строку, которая будет добавлена после имени пользователя после выполнения поиска LDAP.

Следующая конфигурация позволит **admin** подключаться к базе данных **blog** с IP-адреса **192.168.54.1**, если они предоставят правильный пароль LDAP:

```
# type  database  user  address          auth-method  auth-options
host    all       all   192.168.54.1/24  ldap         ldapserver=ldap.example.com ldapport=389 ldapprefix="cn=" ldapsuffix=",dc=example,dc=com"
```

Включение внешних файлов

Записи, перечисленные в таблице ниже, позволяют включать содержимое внешних файлов в поле **PG_HBA**

Имя	Описание
include	Включает в себя содержимое указанного файла.
include_if_exists	Делает то же самое, что и include, но не выдает ошибку, если файл не существует. В противном случае и тот, и другой вызывают сообщение о том, что файл был пропущен.
include_dir	Делает то же самое, что и include, но вместо этого включает все файлы в каталоге, которые заканчиваются на .conf. Файлы, начинающиеся с точки, игнорируются.

Пример:

```
include files
```

Просмотр записей pg_hba.conf

Можно использовать представление PG_HBA_FILE_RULES, чтобы видеть содержимое конфигурации pg_hba.conf. Это представление отображает строки для каждой записи в файле, а также любые ошибки. Например, если в pg_hba.conf есть три записи, и вторая из них содержит недопустимый метод аутентификации, представление будет выглядеть следующим образом:

```
SELECT * FROM pg_hba_file_rules;
```

Возможный результат:

```
 rule_number |    file_name     | line_number | type  | database |  user_name  |  address  | netmask | auth_method | options | error
-------------+------------------+-------------+-------+----------+-------------+-----------+---------+-------------+---------+-------
           1 | /.../pg_hba2/main/pg_hba.conf |           1 | local | {all}    | {all}       |           |         | peer        |         |
           2 | /.../pg_hba2/main/pg_hba.conf |           2 | local | {replication} | {admin,replicator} |           |         | scram-sha-256 |         |
           3 | /.../pg_hba2/main/pg_hba.conf |           3 | host  | {all}    | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
           4 | /.../pg_hba2/main/pg_hba.conf |           4 | host  | {all}    | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
           5 | /.../pg_hba2/main/pg_hba.conf |           5 | host  | {replication} | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
           6 | /.../pg_hba2/main/pg_hba.conf |           6 | host  | {all}    | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
           7 | /.../pg_hba2/main/pg_hba.conf |           7 | host  | {replication} | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
           8 | /.../pg_hba2/main/pg_hba.conf |           8 | host  | {all}    | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
           9 | /.../pg_hba2/main/pg_hba.conf |           9 | host  | {replication} | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
          10 | /.../pg_hba2/main/pg_hba.conf |          10 | host  | {all}    | {admin,replicator} | 10.92.40.185 | 255.255.255.255 | scram-sha-256 |         |
          11 | /.../pg_hba2/main/pg_hba.conf |          11 | host  | {all}    | {postgres} | 0.0.0.0   | 0.0.0.0         | trust       |         |
          12 | /.../pg_hba2/main/pg_hba.conf |          12 | host  | {all}    | {admin}    | 0.0.0.0   | 0.0.0.0         | trust       |         |
(12 rows)
```

Поле PG_HBA

Чтобы изменить поле **PG_HBA** в кластерах postgres-ресурсов КОРУС откройте раздел **Clusters**, выберите кластер ACPO и перейдите на вкладку **Services**. Выберите сервис **patroni**. В панели слева **PG_HBA** далее **Configuration** на вкладке **Primary configuration**.

Редактирование поля PG_HBA

Введите запись в открывшемся окне и нажмите **Apply**.

Добавление записи

Для сохранения изменений нажмите **Save**.

Сохранение изменений

Выполните действие Reconfigure & Restart, чтобы применить изменения.

Аутентификация по паролю

Евгения Кузина

Содержание

Назад к содержанию

- Настройка аутентификации
  - Файл pg_hba.conf
  - Параметр password_encryption
- Установка и изменение паролей пользователей
- Просмотр сохраненных паролей

ADPG поддерживает несколько методов аутентификации по паролю, которые отличаются тем, как пароль хранится на сервере и передается во время соединения. В таблице ниже перечислены поддерживаемые методы.

Методы аутентификации по паролю

Название	Описание
scram-sha-256	Выполняется аутентификация `scram-sha-256` в соответствии с RFC 7677. Она производится по схеме "вызов-ответ", которая предотвращает перехват паролей при ненадежных соединениях. Пароли хранятся на сервере в виде криптографического хеша. `scram-sha-256` — наиболее безопасный из существующих на данный момент методов, но старые клиентские библиотеки его не поддерживают
md5	Использует менее безопасный механизм "вызов-ответ". Он предотвращает перехват паролей и предусматривает хранение паролей на сервере в зашифрованном виде, но не защищает в случае хищения хешей паролей с сервера. Алгоритм хеширования md5 также уязвим для коллизионных атак. Метод `md5` не поддерживает db_user_namespace
password	Отправляет пароль в незашифрованном виде. Этот метод уязвим для атак с перехватом пароля. Вы можете использовать метод `password`, только если соединение защищено шифрованием SSL

Настройка аутентификации

Следующие опции управляют настройками аутентификации по паролю:

- Файл pg_hba.conf, содержащий параметры клиентской аутентификации
- Параметр password_encryption, который определяет алгоритм, используемый для шифрования паролей пользователей

Файл pg_hba.conf

Файл pg_hba.conf содержит набор записей, по одной на каждой строке. За подробным описанием полей записи обратитесь к статье Обзор конфигурации PG_HBA.

Для редактирования *pg_hba.conf* используйте пользовательский интерфейс ADCM. Откройте вкладку ***Clusters → ADPG cluster → Services → ADPG → Primary configuration*** и добавьте строки в поле ***PG_HBA***. Следующая запись позволяет установить соединение по протоколу TCP/IP, с любыми базами данных, любым пользователям с адресов IPv4 и методом аутентификации `scram-sha-256`:

```
host  all  all 0.0.0.0/0 scram-sha-256
```

Нажмите ***Save*** на вкладке ***Configuration*** и выполните действие Reconfigure & Restart, чтобы применить изменения. Следующие строки добавятся в *pg_data1/adpg16/pg_hba.conf*:

```
# BEGIN Customs from ADCM
host  all  all 0.0.0.0/0 scram-sha-256
# END Customs from ADCM
```

Параметр password_encryption

Если вы устанавливаете пароль SQL-командой CREATE ROLE или ALTER ROLE, алгоритм шифрования пароля определяется параметром **password_encryption**. Возможные значения: `scram-sha-256` (значение по умолчанию) и `md5`.

Если в качестве метода аутентификации в *pg_hba.conf* указан `md5`, но пароль пользователя на сервере зашифрован алгоритмом `scram-sha-256`, то используется аутентификация `scram-sha-256` как наиболее безопасная.

Вы можете изменить параметр `password_encryption` через пользовательский интерфейс ADCM. Для этого откройте вкладку ***Clusters → ADPG cluster → Services → ADPG → Primary configuration***, раскройте ноду ***ADPG configurations*** и добавьте в поле ***postgresql.conf*** новое значение `password_encryption` (см. Конфигурационные параметры). Следующая строка устанавливает алгоритм `md5` для шифрования пароля:

```
password_encryption=md5
```

Вы также можете выполнить SQL-инструкцию приведенную ниже, чтобы изменить параметр `password_encryption` для текущей сессии.

```
SET password_encryption = 'scram-sha-256';
```

Установка и изменение паролей пользователей

Если вы не установили пароль для пользователя, его пароль будет равен NULL и аутентификация по паролю для этого пользователя завершится ошибкой.

Чтобы установить пароль для новой роли, укажите параметр `PASSWORD` в SQL-команде CREATE ROLE.

```
CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
```

Чтобы изменить пароль для существующей роли, выполните команду ALTER ROLE.

```
ALTER ROLE user1 WITH PASSWORD 'password2';
```

Метакоманда psql `\password` также изменяет пароль для авторизованной роли.

```
\password
```

После выполнения команды `\password` терминал выведет приглашение ввести новый пароль дважды:

```
Enter new password for user "postgres":
Enter it again:
```

Если введенные пароли совпадут, текущий пароль будет изменен.

Просмотр сохраненных паролей

Хеш пароля для каждого пользователя базы данных хранится в каталоге pg_authid. Вы можете выполнить следующую SQL-команду, чтобы получить список ролей и паролей:

```
SELECT rolname, rolpassword FROM pg_authid;
```

Вывод команды:

```
         rolname          |          rolpassword
--------------------------+---------------------------------------
 pg_database_owner        |
 pg_read_all_data         |
 pg_write_all_data        |
 pg_monitor               |
 pg_read_all_settings     |
 pg_read_all_stats        |
 pg_stat_scan_tables      |
 pg_read_server_files     |
 pg_write_server_files    |
 pg_execute_server_program |
 pg_signal_backend        |
 postgres                 | SCRAM-SHA-256$4096:2yHlw+fjx7PWll0dScESNA==$SETiIdi
xLSGgtE/EEk/5kfRBsIpu+09XuCDPWIFmAuE=:Fyy6OPtQmm1+Ms8rvqww47NT1YhLJ6/zik4iskkm3M
o=
 user_md5                 | md5ea4a7de3c817a0c8cc2670dab9aecc47
(13 rows)
```

Хеш пароля начинается с названия алгоритма шифрования. В этом примере пароль роли `postgres` зашифрован с помощью `scram-sha-256`, а роли `user_md5` — с помощью алгоритма `md5`.

Аутентификация Kerberos

Д. Евгения Кузина

Содержание

Назад к содержанию

- Установка сервера FreeIPA
- Установка клиента FreeIPA
- Настройка ADPG для использования Kerberos
- Установка пакета openldap-clients
- Настройка ADPG для использования LDAP

Kerberos — это протокол для запросов службы аутентификации между доверенными хостами в неподвижной сети. Kerberos управляет учетными данными (аутентификация), а [LDAP](#) хранит достоверную информацию об учетных записях пользователей и их правах доступа (авторизация).

В этой статье описано создание защищенного соединения между следующими хостами с операционной системой CentOS:

- 10.92.6.96, adpg-host, adpg-host.domain.local;
- 10.92.6.201, freeipa-host, freeipa-host.domain.local.

ADPG установлен на хост adpg-host.

Установка сервера FreeIPA

Чтобы развернуть сервер FreeIPA на хосте freeipa-host, будем использовать Docker CE. Выполните шаги описанные ниже для установки:

1. Добавьте имена хостов и их IP-адреса в файл /hosts:

```
$ sudo tee -a /etc/hosts > /dev/null <<EOT
10.92.6.96  adpg-host.domain.local adpg-host
10.92.6.201 freeipa-host.domain.local freeipa-host
EOT
```

2. Используйте утилиту hostnamectl, чтобы задать имя хоста:

```
$ sudo hostnamectl set-hostname freeipa-host.domain.local
```

3. Выполните следующие команды, чтобы отключить SELinux:

```
$ sudo setenforce 0
$ sudo sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
$ sudo reboot
```

4. Установите Docker CE:

```
$ sudo -i
$ yum remove docker \
              docker-client \
              docker-client-latest \
              docker-common \
              docker-latest \
              docker-latest-logrotate \
              docker-logrotate \
              docker-engine
$ yum install -y yum-utils
$ yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
$ yum install docker-ce docker-ce-cli containerd.io docker-compose-plugin
$ systemctl start docker
$ systemctl enable docker
```

5. Создайте каталог для data для файлов конфигурации FreeIPA, укажите параметры установки FreeIPA и запишите их в файл ipa-server-install-options:

```
$ mkdir /var/lib/ipa-data
$ tee -a /var/lib/ipa-data/ipa-server-install-options > /dev/null <<EOT
--realm=DOMAIN.LOCAL
--ds-password=FreeIpaSpass
--admin-password=FreeIpaADMINpass
EOT
```

6. Установите сервер FreeIPA:

```
$ docker run -it -h freeipa-host.domain.local --read-only \
    -v /sys/fs/cgroup:/sys/fs/cgroup:ro \
    --name freipa \
    --network host \
    -v /var/lib/ipa-data:/data:Z -v /etc/hosts:/etc/hosts \
    -e PASSWORD=Password123 \
    freeipa/freeipa-server:centos-7 ipa-server-install -U -r DOMAIN.LOCAL --no-ntp
```

Установка клиента FreeIPA

Установим клиент FreeIPA на хост adpg-host.

1. Добавьте имена хостов и их IP-адреса в файл /hosts:

```
$ sudo tee -a /etc/hosts > /dev/null <<EOT
10.92.6.96  adpg-host.domain.local adpg-host
10.92.6.201 freeipa-host.domain.local freeipa-host
EOT
```

2. Используйте утилиту hostnamectl, чтобы задать имя хоста:

```
$ sudo hostnamectl set-hostname adpg-host.domain.local
```

3. Выполните следующие команды, чтобы отключить SELinux:

```
$ sudo setenforce 0
$ sudo sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
$ sudo reboot
```

4. Установите клиент FreeIPA:

```
$ sudo -i
$ yum install freeipa-client
```

5. Используйте команду ipa-client-install, чтобы сконфигурировать adpg-host для использования IPA:

```
$ ipa-client-install --uninstall
$ ipa-client-install --no-ntp --mkhomedir --domain=domain.local --server=adpg-host.domain.local
```

Терминал отобразит следующее уведомление:

```
AutoDiscovery of servers for failover cannot work with this configuration.
If you proceed with the installation, services will be configured to always access the discovered server for all operations and
will not fail over to other servers in case of failure.
Proceed with fixed values and no DNS discovery? [no]:
```

Наберите `yes` для продолжения с фиксированными значениями. Фиксированные значения будут выведены на дисплей:

```
Client hostname: adpg-host.domain.local
Realm: DOMAIN.LOCAL
DNS Domain: domain.local
IPA Server: freeipa-host.domain.local
BaseDN: dc=domain,dc=local

Continue to configure the system with these values? [no]:
```

Наберите `yes` для подтверждения.

Укажите `admin` в качестве авторизованного пользователя:

```
User authorized to enroll computers: admin
```

Введите пароль, который был передан в качестве параметра `PASSWORD` команды `docker run` (в нашем примере — `Password123`).

6. Используйте команду `klist`, чтобы получить новый мандат (ticket) Kerberos. Выполните команду klist, чтобы проверить, создан ли мандат:

```
$ kinit admin
$ klist
```

7. Добавьте пользователя и установите для него пароль:

```
$ ipa user-add pguser --first=pguser --last=PG --password
```

Результат:

```
----------------------
Added user "pguser"
----------------------
  User login: pguser
  First name: pguser
  Last name: PG
  Full name: pguser PG
  Display name: pguser PG
  Initials: gP
  Home directory: /home/pguser
  GECOS: pguser PG
  Login shell: /bin/sh
  Principal name: pguser@DOMAIN.LOCAL
  Principal alias: pguser@DOMAIN.LOCAL
  User password expiration: 20220804154601Z
  Email address: pguser@domain.local
  UID: 139160001
  GID: 139160001
  Password: True
  Member of groups: ipausers
  Kerberos keys available: True
```

8. Службам требуется SPN (**Service Principal Name**) для использования аутентификации Kerberos. Зарегистрируйте службу postgres для получения SPN:

```
$ ipa service-add postgres/adpg-host.domain.local@DOMAIN.LOCAL --force
```

Результат:

```
-------------------------------------------------------------
Added service "postgres/adpg-host.domain.local@DOMAIN.LOCAL"
-------------------------------------------------------------
  Principal name: postgres/adpg-host.domain.local@DOMAIN.LOCAL
  Principal alias: postgres/adpg-host.domain.local@DOMAIN.LOCAL
  Managed by: adpg-host.domain.local
```

Настройка ADPG для использования Kerberos

Создайте keytab-файл на хосте adpg-host:

```
$ ipa-getkeytab --principal=postgres/adpg-host.domain.local@DOMAIN.LOCAL --keytab=/etc/krb5.keytab
```

Выполните команду chown, чтобы изменить владельца и группу файла krb5.keytab:

```
$ chown postgres:postgres /etc/krb5.keytab
```

Используйте pgAdmin чтобы изменить конфигурационные файлы ADPG. Откройте вкладку **Cluster** → **ADPG cluster** → **Services** → **ADPG** → **Primary configuration**, раскройте ноду **ADPG configurations** и установите в поле **postgresql.conf** значение `listen_addresses` равным `*`.

Также добавьте в поле **postgresql.conf** новый параметр `krb_server_keyfile`:

```
$ krb_server_keyfile = '/etc/krb5.keytab'
```

Добавьте следующую строку в поле **PG_HBA**:

```
host all pguser 0.0.0.0/0 gss  include_realm=0 krb_realm=DOMAIN.LOCAL
```

Кликните кнопку **Save** и выполните действие Reconfigure & Restart, чтобы применить изменения.

Создайте пользователя на хосте adpg-host:

```
$ sudo su - postgres
$ psql -c "CREATE USER pguser"
```

Проверьте соединение с использованием GSSAPI:

```
$ kinit pguser
$ psql -h adpg-host.domain.local -U pguser postgres
```

Результат:

```
psql (14.3)
GSSAPI-encrypted connection
Type "help" for help.

postgres=#
```

Установка пакета openldap-clients

Пакет openldap-clients содержит инструменты командной строки для просмотра и изменения каталогов на сервере LDAP. Установите пакет openldap-clients на хост adpg-host:

```
$ yum install openldap-clients -y
```

Выполните команду `klist`, чтобы проверить, получил ли текущий системный пользователь мандат Kerberos с правами администратора IPA. Если нет, выполните команду `kinit admin`:

```
$ klist
$ kinit admin
```

Добавьте пользователя:

```
$ ipa user-add pgldap --first=pgldap --last=PG --password
```

Проверьте соединение:

```
$ ldapsearch -x -h freeipa-host.domain.local  -b dc=domain,dc=local uid=pgldap
```

Настройка ADPG для использования LDAP

Создайте пользователя `pgldap`:

```
$ sudo su - postgres
$ psql -c "CREATE USER pgldap"
```

Добавьте следующую строку в поле **PG_HBA**:

```
host all pgldap 0.0.0.0/0 ldap  ldapserver=freeipa-host.domain.local ldapbasedn="cn=users,cn=compat,dc=domain,dc=local"
ldapsearchattribute=uid
```

Кликните **Save** и выполните действие Reconfigure & Restart, чтобы применить изменения.

Проверьте соединение:

```
$ kinit pgldap
$ psql -h adpg-host.domain.local -U pgldap postgres
```

Результат:

```
psql (14.3)
GSSAPI-encrypted connection
Type "help" for help.

postgres=#
```

Содержание

Назад к содержанию

Обзор аутентификации LDAP
- Режим простого связывания
- Режим поиск + связывание

Настройка сервера FreeIPA

Настройка ADPG для использования LDAP

Примеры
- FreeIPA server
- MS Active Directory

Обзор аутентификации LDAP

Вы можете использовать аутентификацию LDAP для доступа к кластер ADPG. Этот метод аутентификации работает аналогично аутентификации по паролю за исключением того, что используют LDAP в качестве метода проверки. Для верификации пар имен пользователя/пароль ADPG подключается к серверу LDAP, например Active Directory или 389 Directory Server. Обратите внимание, что пользователь должен уже существовать в базе данных PostgreSQL, чтобы LDAP можно было использовать для аутентификации.

LDAP-аутентификация осуществляется в двух режимах:

- простое связывание;
- поиск + связывание.

В таблице ниже перечислены параметры конфигурации, сгруппированные по режимам связывания. Все эти параметры должны быть указаны в поле `auth-options` (см. ниже). См. Настройка ADPG для использования LDAP.

Параметры, используемые в обоих режимах

Название параметра	Описание
ldapserver	Имена или IP-адреса серверов LDAP для подключения. Можно указать несколько серверов, разделенных пробелами
ldapport	Номер порта на сервере LDAP для подключения. Если порт не указан, будет использоваться порт библиотеки LDAP по умолчанию (389)
ldapscheme	Установите `ldaps` в качестве значения этого параметра, чтобы использовать LDAPS (LDAP через TLS/SSL). поддерживаемый некоторыми реализациями сервера LDAP. В качестве альтернативы можно рассмотреть параметр `ldaptls`
ldaptls	Установите `1` в качестве значения этого параметра, чтобы соединение между ADPG/PostgreSQL и сервером LDAP использовало шифрование TLS. Этот подход использует операцию StartTLS, описанную в стандарте RFC 4513

Настройки `ldapscheme` + `ldaptls` шифруют только трафик между сервером ADPG/PostgreSQL и LDAP. Соединение между сервером ADPG/PostgreSQL и клиентом ADPG/PostgreSQL не шифруется.

Режим простого связывания

В режиме простого связывания ADPG/PostgreSQL подключается с указанным именем (Distinguished Name, DN), сформированным как `prefix` + `user name` + `suffix`, обычно префикс используется для указания `cn=` или `DOMAIN\`, а суффикс для указания остальной части DN.

Параметры, используемые в режиме простого связывания

Название параметра	Описание
ldapprefix	Указывает строку, которая добавляется перед именем пользователя при формировании DN для аутентификации
ldapsuffix	Указывает строку, которая добавляется после имени пользователя при формировании DN для аутентификации

Режим поиск + связывание

В режиме поиска ADPG/PostgreSQL подключается к каталогу LDAP с фиксированным именем пользователя и паролем, указанными с помощью параметров `ldapbinddn` и `ldapbindpasswd`, и выполняет поиск пользователя. Если имя пользователя для поиска и пароль не настроены, будет предпринята попытка анонимного подключения. Поиск выполняется в поддереве `ldapbasedn`, проверяя поле совпадения имени пользователя и значения `ldapsearchattribute`. Как только пользователь находится, сервер пытается повторно подключиться в LDAP-каталог от имени этого пользователя с паролем, предоставленным клиентом, чтобы убедиться в правильности учетной записи.

Параметры, используемые в режиме поиск + связывание

Название параметра	Описание
ldapbasedn	Корневой каталог DN для начала поиска пользователя
ldapbinddn	DN пользователя для выполнения поиска в каталоге
ldapbindpasswd	Пароль пользователя для выполнения поиска в каталоге
ldapsearchattribute	Атрибут для сопоставления с именем пользователя во время аутентификации в режиме поиск + связывание. Если не указано, будет использоваться атрибут `uid`
ldapsearchfilter	Фильтр поиска. `$username` в фильтре будет заменен на имя пользователя. Это опция реализует более гибкие фильтры поиска, чем `ldapsearchattribute`. Например, `ldapsearchfilter="(|(uid=$username)(mail=$username))"`
ldapurl	Определяет RFC 4516 LDAP URL. `ldap[s]://host[:port]/basedn[?attribute][?scope][?filter]]]` `scope` используется для указания области поиска. Допустимые значения. • `base` – поиск базового объекта. • `one` – поиск на одном уровне. • `sub` – поиск в поддереве каталога.

Настройка сервера FreeIPA

В этом разделе содержится инструкция по установке и конфигурированию сервера FreeIPA, который используется в примерах ниже.

Выполните следующие шаги для установки сервера FreeIPA на хост с операционной системой CentOS или RHEL:

1. Установите сервер FreeIPA.

```
$ sudo yum install ipa-server
```

2. В приведенных ниже примерах конфигурации LDAP-аутентификации используются имена хостов. Если в вашей сети нет службы DNS для разрешения имен хостов, вы можете выполнить следующую команду, чтобы добавить имена хостов и их IP-адреса в файл hosts:

```
$ sudo tee -a /etc/hosts > /dev/null <<EOT
10.92.42.229 adpg.example.com adpg-host
10.92.43.11 ipa.example.com ipa-host
EOT
```

Где:

- `10.92.42.229` — IP-адрес хоста с ADPG.
- `10.92.43.11` — IP-адрес хоста с сервером FreIPA.

3. Настройте сервер FreeIPA.

```
$ sudo ipa-server-install \
  --unattended \
  --hostname=ipa.example.com \
  --domain=example.com \
  --realm=EXAMPLE.COM \
  --ds-password=pass1246 \
  --admin-password=pass1234
```

4. Проверьте состояние служб, используя команду `ipactl status`.

```
$ sudo ipactl status
```

Вывод должен показать, что службы `Directory` `krb5kdc` и `kadmin` запущены:

```
Directory Service: RUNNING
krb5kdc Service: RUNNING
kadmin Service: RUNNING
httpd Service: RUNNING
ipa-custodia Service: RUNNING
ntpd Service: RUNNING
pki-tomcatd Service: RUNNING
ipa-otpd Service: RUNNING
ipa: INFO: The ipactl command was successful
```

5. Авторизуйтесь как пользователь `admin`, который создается автоматически для выполнения административных действий. Для этого выполните команду `kinit`:

```
$ kinit admin
```

При появлении следующего приглашения введите пароль, указанный в параметре `admin-password` конфигурации сервера FreeIPA (в этом примере `pass1234`).

```
Password for admin@EXAMPLE.COM:
```

6. Используйте команду `ipa user add` для создания пользователя FreeIPA.

```
$ ipa user-add John \
--first=John \
--last=Smith \
--password
```

Введите пароль нового пользователя дважды:

```
Password:
Enter Password again to verify:
```

Результат:

```
----------------------
Added user "john"
----------------------
  User login: john
  First name: John
  Last name: Smith
  Full name: John Smith
  Display name: John Smith
  Initials: JS
  Home directory: /home/john
  GECOS: John Smith
  Login shell: /bin/sh
  Principal name: john@EXAMPLE.COM
  Principal alias: john@EXAMPLE.COM
  User password expiration: 20250317131002
  Email address: john@example.com
  UID: 1414680003
  GID: 1414680003
  Password: True
  Member of groups: ipausers
  Kerberos keys available: True
```

Настройка ADPG для использования LDAP

Если в вашей сети нет службы DNS для разрешения имен хостов, выполните следующую команду на хосте ADPG/PostgreSQL, чтобы добавить имена хостов и их IP-адреса в файл hosts:

```
$ sudo tee -a /etc/hosts > /dev/null <<EOT
10.92.42.229 adpg.example.com adpg-host
10.92.43.11 ipa.example.com ipa-host
EOT
```

Где:

- `10.92.42.229` — IP-адрес хоста с ADPG.
- `10.92.43.11` — IP-адрес хоста с сервером FreeIPA.

Как упоминалось выше, пользователь должен уже существовать в базе данных, прежде чем LDAP может быть использован для аутентификации.

Создайте роль для пользователей LDAP, чтобы задействовать их в записи формата pg_hba.conf.

```
CREATE ROLE ldap_users LOGIN;
```

Создайте роль для конкретного пользователя в группе `ldap_users`. Обратите внимание, что имя пользователя ADPG должно совпадать с именем пользователя LDAP.

```
CREATE ROLE john IN GROUP ldap_users;
```

Для настройки параметров подключения откройте страницу **Clusters**, выберите кластер ADPG и перейдите на вкладку **Services**. Выберите сервис **ADPG** и кликните после на Edit на вкладке **Primary configuration** в группе **Configuration**. Добавьте новую запись в формате pg_hba.conf в это поле. Следующая запись использует режим простого связывания.

```
host    all         +ldap_users     0.0.0.0/0       ldap        ldapserver=ipa.example.com ldapprefix="uid=" ldapsuffix=",cn=users,cn=accounts,dc=example,dc=com"
```

Проверьте содержимое с помощью утилиты `psql`.

```
$ psql postgres -U john -h 10.92.42.229
```

При появлении следующего приглашения введите пароль, указанный ранее для LDAP-пользователя `john`.

```
Password for user john:
```

Примеры

Поскольку LDAP-часто используют запятые и пробелы для разделения различных частей DN, при настройке параметров LDAP может возникнуть необходимость указывать значения параметров в двойных кавычках, как показано в примерах ниже.

FreeIPA server

Конфигурация ниже использует режим поиск к связыванию. Поскольку `ldapbinddn` не указан, ADPG пытается анонимно привязаться к серверу LDAP и выполнить поиск пользователя в указанном базовом DN (параметр `ldapbasedn`).

```
host    all         +ldap_users     0.0.0.0/0       ldap        ldapserver=ipa.example.com ldapbasedn="cn=users,cn=accounts,dc=example,dc=com"
```

Пример той же конфигурации для режима поиск + связывание, но записанной в виде URL (параметр `ldapurl`).

```
host    all         +ldap_users     0.0.0.0/0       ldap        ldapurl="ldap://ipa.example.com/cn=users,cn=accounts,dc=example,dc=com?uid?sub"
```

Методы, использующие анонимное связывание, не являются безопасными. Предпочтительные решения используют `ldapbinddn` + `ldapbindpasswd` для выполнения поиска от имени авторизованного пользователя:

```
host    all         +ldap_users     0.0.0.0/0       ldap        ldapserver=ipa.example.com ldapbasedn="cn=users,cn=accounts,dc=example,dc=com" ldapbinddn="uid=admin,cn=users,cn=accounts,dc=example,dc=com" ldapbindpasswd="pass1234"
```

MS Active Directory

Следующая конфигурация включает аутентификацию LDAP в режиме поиск + связывание для MS Active Directory:

```
host    all         +ldap_users     0.0.0.0/0       ldap        ldapserver=ad01.example.ia ldapbasedn="OU=People,DC=ad,DC=ranger-test" ldapbinddn="cn=admin,dc=ad,dc=ranger-test" ldapbindpasswd="bind_pass123" ldapsearchattribute="sAMAccountName"
```

Конфигурация Active Directory, использованная в примере	
url	ldap://ad01.example.io:389
binddn	cn=admin,dc=ad,dc=ranger-test
bind password	bind_pass123
user nameattribute	sAMAccountName
searchbase	ou=People,dc=ad,dc=ranger-test
group searchbase	ou=Groups,dc=ad,dc=ranger-test
user searchbase	ou=People,dc=ad,dc=ranger-test

Включение SSL-шифрования

Содержание

Назад к содержанию

- Настройка SSL через ADCM
- Использование клиентских сертификатов
- Создание самоподписанных SSL-сертификатов
  - Генерация сертификатов для кластера ADPG
  - Генерация SSL-сертификатов для клиентов PostgreSQL
- Генерация SSL-сертификатов для клиентов PostgreSQL

ADPG поддерживает SSL-шифрование клиентских соединений для защиты данных при передаче. Настройка SSL-шифрования может быть выполнена с помощью SSL-сертификатов, предоставленных клиентом, или самоподписанных сертификатов, генерируемых кластером ADPG. Для получения дополнительной информации см. раздел Использование клиентских SSL-сертификатов.

Если SSL включен, все пароли клиентов ADPG будут использовать scram-sha-256 вместо md5 или password.

Важно

- Настоятельно рекомендуется создать файлы сертификата и ключей для использования SSL (.crt, .key) (по умолчанию - .pg_ssl/ssl.*), поскольку самоподписанные файлы сертификатов будут удалены при обновлении кластера.
- Наследник, поддерживающий сертификаты SSL (Standby), должен иметь файлы сертификата, совпадающие с файлами основного кластера. Файлы сертификата должны быть скопированы вручную.
- Использование клиентов, защищенных паролем, не поддерживается.
- При отключении SSL-шифрования в кластере, использующем SSL, кластер будет перезапущен. Это обеспечивает удаление файлов сертификатов.

Настройка SSL через ADCM

Чтобы включить SSL, выполните следующие действия:

1. Откройте страницу **Clusters** и выполните действие **Manage SSL**.

Выполнение действия "Manage SSL"

2. В открывшемся окне активируйте переключатель **Enable SSL**.

Окно "Run an action: Manage SSL"

Настройте SSL, изобразите под переключателем **Enable SSL**. Параметры, задействованные кружком, являются обязательными.

Параметры SSL

3. Укажите настройки SSL в соответствии с вашими требованиями. Для получения дополнительной информации см. раздел Использование клиентских SSL-сертификатов.

В таблице ниже перечислены параметры SSL, доступные для редактирования. Столбец **Расположение файла** описывает, на каком хосте кластера должен храниться файл сертификата.

Параметр	Описание	Расположение файла
ADPG generic certificate	Путь к общему сертификату ADPG, соответствующему публичному ключу клиента.	На каждом хосте кластера, где установлены сервисы ADPG, Balancer, End-to-end Monitoring.
ADPG generic certificate key	Путь к ключу общего сертификата ADPG, который используется для шифрования данных, передаваемых между ADPG и клиентом. Файл должен быть доступен для чтения пользователем postgres. Если ключ защищен паролем, укажите пароль в поле Passphrase for ADPG generic certificate key. Если ключ не защищен паролем, оставьте это поле пустым.	На каждом хосте кластера, где установлены сервисы ADPG, Balancer, End-to-end Monitoring.
Arenadata PostgreSQL certificate	Путь к сертификату, предоставленному от имени Arenadata PostgreSQL (PostgreSQL). Значением параметра SET_ENV_PTLS из файла postgresql.conf. Подробнее см. в Настройка.	На каждом хосте кластера, где установлен сервис ADPG.
Arenadata PostgreSQL certificate key	Путь к ключу сертификата Arenadata PostgreSQL (PostgreSQL). Значением параметра SET_ENV_PTLS из файла postgresql.conf. Подробнее см. в Настройка. Файл должен быть доступен для чтения пользователем postgres. Если ключ защищен паролем, укажите пароль в поле Passphrase for Arenadata PostgreSQL certificate key. Если ключ не защищен паролем, оставьте это поле пустым.	На каждом хосте кластера, где установлен сервис ADPG.
CA file	Укажите имя файла, содержащего сертификаты доверенных центров сертификации (CA) для SSL-соединений. Этот параметр соответствует параметру ssl_ca_file в postgresql.conf. Подробнее см. в Настройка.	На каждом хосте кластера, где установлен сервис ADPG.
SSI backup CA file	Укажите имя файла, содержащего сертификаты доверенных центров сертификации (CA) для SSL-соединений. Этот параметр соответствует параметру Peer Trust для установки в SR.	На каждом хосте кластера, где установлен сервис ADPG.

4. После указания параметров SSL нажмите **Run**

Окно "Run an action: Manage SSL" с настройками SSL

5. На вкладке **Confirmation** нажмите **Run**, чтобы выполнить действие **Manage SSL**.

Вкладка "Confirmation"

После выполнения действия **Manage SSL** на хостах SSL добавляются в раздел **SSL configuration** операции кружка ADPG. Этот раздел содержит следующие поля:

- **Enable SSL** - поле для активации SSL-соединений. Значение **true** для включения SSL, **false** для отключения.
- **postgresql.conf** - поле для указания пути к файлу postgresql.conf.
- **PG_HBA** - поле для указания пути к файлу pg_hba.conf.

5. После завершения действия **Manage SSL** откройте страницу SSL-соединения для кружка ADPG. Для этого откройте вкладку **Primary configuration** кружка ADPG, **SSL configuration** и раздел **Configuration** и нажмите на **PG_HBA**

Раздел "SSL configuration"

Добавьте строку, которая разрешает SSL-соединения в поле **PG_HBA (pg_hba.conf) configuration**. Например:

```
hostssl    all    all    0.0.0.0/0    cert
```


Поле "PG_HBA"

Эта запись позволяет всем пользователям (**all**) использовать SSL-соединение (**hostssl**) с любого адреса IPv4 (**0.0.0.0/0**). Пользователь должен предоставить действительный сертификат SSL (**cert**) для успешной аутентификации (подробнее о типах [client authentication](#) см. в документации PostgreSQL).

6. Сохраните изменения и запустите действие [Reconfig & Restart](#), чтобы применить изменения.

СПИСОК ПОРТОВ

Для действия **Manage SSL** параметрами параметры SSL, указанные в разделе **postgresql.conf** и **Custom postgresql.conf** раздела **Configuration** кластера ADPG. Обратите внимание, что параметры SSL, такие как **ssl_cert_file**, **ssl_key_file** будут приняты настройками SSL из этих разделов. Не рекомендуется использовать эти параметры для ручной регистрации сертификатов, так как они будут перезаписаны при действии **Manage SSL**.

Использование клиентских сертификатов

Для записи имени **PG_HBA** с типами соединения **hostssl**, можно выбрать вариант аутентификации **clientcert=verify-ca** или **clientcert=verify-full**.

Если указан **clientcert=verify-ca**, сервер проверяет, что клиент предоставляет доверенный сертификат. Если используется **clientcert=verify-full**, помимо истины проверку центра сертификата, он также проверяет, соответствует ли имя клиентского сертификата его имени. Дополнительную информацию см. в документации PostgreSQL.

Примечательно, что опция **clientcert** может быть использована для любого типа записи **hostssl**, **hostnossl**, **hostgssenc** или **hostnogssenc**.

Проверяется только аутентификация сертификата клиента, также могут быть включены и другие режимы аутентификации. Если **clientcert** не указан, режим проверки сертификата клиента не применяется. Для получения дополнительной информации см. раздел Настройка.

Если **clientcert** не указан, аутентификация **cert** для записей **hostssl**, **PG_HBA** (**pg_hba.conf**) не работает для аутентификации. Как и для каждого SSL-соединения, аутентификация клиента все еще выполняется, если сертификат сервера может быть проверен. Для записей **hostssl**, если **clientcert** не указан, сервер проверяет сертификат клиента в соответствии с его [client authentication](#) configuration.

- Использование любого метода аутентификации **hostssl** с включенным сертификатом через условие сертификата клиента **clientcert=verify-ca** или **clientcert=verify-full**.
- Использование любого метода аутентификации **hostssl** с включенным сертификатом через условие сертификата клиента **clientcert=verify-ca** или **clientcert=verify-full**, если **clientcert** не указан. Это позволяет использовать сертификаты клиента, которые также проверяют, что **cn** (Common Name) в сертификате клиента соответствует запрошенному имени пользователя базы данных.

Дополнительную информацию см. в документации PostgreSQL.

Создание самоподписанных SSL-сертификатов

Ввести сертификаты вручную, вы можете генерировать самоподписанные сертификаты, которые могут использоваться для разработки и тестирования, по умолчанию самоподписанные сертификаты генерируются кластером ADPG.

Генерация сертификатов для кластера ADPG

Тестовый центр (CA), для которого создаются сертификаты, включает в себя три кластера, описанных в таблице ниже.

Хост	IP	FQDN	Сервис
ssl-host-1	10.92.41.35	ssl-host-1.ru-central1.internal	ADPG
ssl-host-2	10.92.42.220	ssl-host-2.ru-central1.internal	Etcd
ssl-host-3	10.92.41.52	ssl-host-3.ru-central1.internal	Balancer

В данном примере сертификаты создаются на хосте **ssl-host-1** с сервисом ADPG.

Для генерации ADPG сертификатов понадобится OpenSSL с открытой CA (Открытый Центр Сертификации). Для этого создайте начальной конфигурационный файл с именем openssl.cnf. Вы можете использовать следующий пример, но вам следует настроить его в соответствии с вашими требованиями.

```
[req]
default_bits = 4096
prompt = no
default_md = sha256
x509_extensions = v3_req
distinguished_name = dn

[dn]
C = RU
ST = Moscow
O = Arenadata
CN = Common Name

[v3_req]
keyUsage = digitalSignature, nonRepudiation
extendedKeyUsage = serverAuth, clientAuth
subjectAltName = @alt_names

[alt_names]
IP.1 = 127.0.0.1
IP.2 = 10.92.41.35
IP.3 = 10.92.42.220
IP.4 = 10.92.41.52
DNS.1 = ssl-host-1
DNS.2 = ssl-host-1.ru-central1.internal
DNS.3 = ssl-host-2.ru-central1.internal
DNS.4 = ssl-host-3.ru-central1.internal
DNS.5 = ssl-host-1
DNS.6 = ssl-host-2
DNS.7 = ssl-host-3
```

Чтобы сгенерировать SSL сертификаты, выполните следующие действия:

1. Создайте каталог для сертификатов. Например, создайте каталог с именем ssl-certs в домашнем каталоге пользователя root.

```
$ sudo mkdir /etc/certificates
$ cd /etc/certificates
```

2. Чтобы создать собственный сервер центра сертификации, создайте закрытый ключ сертификата Certificate Signing Request. CSR и файлы сертификата с открытым ключом (**root.postgresql.crt**) на основании значения, которое **Enter** host-1. Также необходимо установить длительность.

```
$ openssl req -new -nodes -text -out root.csr -keyout root.key -subj "/CN=root.postgresql.crt"
$ chmod 600 root.key
```

3. Подпишите запрос сертификата, чтобы создать корневой центр сертификации. Обратите внимание, что путь к файлу openssl.cnf (/etc/openssl.cnf) может отличаться в вашем кластере.

```
$ openssl x509 -req -in root.csr -text -days 3650 \
-extfile /etc/openssl.cnf -extensions v3_ca \
-signkey root.key -out root.crt
```

Установите необходимые разрешения.

```
$ chmod 600 root.crt
```

В результате создаются файлы root.crt и root.key.

4. Создайте сертификат ADPG (root), подписанный новым сертификатом root.crt, и установите необходимые права.

```
$ openssl req -new -nodes -text -out postgres_1.csr -keyout postgres_1.key -subj "/CN=ssl-host-1"
$ chmod 600 postgres_1.key
$ openssl x509 -req -in postgres_1.csr -text -days 365 -CA root.crt -CAkey root.key -CAcreateserial -out postgres_1.crt
```

Сохраните файлы (ключ сервера: postgres_1.crt, ключ: postgres_1.key).

5. Проверьте сертификаты. Команды могут различаться для разных ОС.

```
$ openssl verify -CAfile root.crt postgres_1.crt
$ openssl verify -CAfile root.crt postgres_1.key
```

6. Измените разрешения для сертификата Arenadata PostgreSQL и экземпляра.

```
$ sudo chown postgres:postgres /etc/certificates/postgres_1.*
```

7. Скопируйте сертификаты CA в стандартный каталог установки для использования сертификатов доступа (по умолчанию этот каталог - /usr/local/share/ca-certificates). Создайте каталог и скопируйте сертификаты. Обратите внимание, что имена сертификатов должны заканчиваться .crt. Этот шаг необходим для включения, если хотя бы один из сертификатов не был сертификатом CA.

```
$ sudo mkdir /usr/local/share/ca-certificates/
$ sudo cp postgres_1.crt /usr/local/share/ca-certificates/
$ sudo cp root.crt /usr/local/share/ca-certificates/
$ sudo update-ca-certificates
```

8. Скопируйте файлы ключей, crt для postgres_1.crt, root.crt для использования в кластере ADPG, как описано в разделе Настройка.

```
$ sudo chmod 600 postgres_1.key
```

9. Создайте сертификат для клиента, с параметром **ADPG generic certificate key** и **ADPG generic certificate** для поля **Manage SSL**.

10. Скопируйте файлы ключей, crt для postgres_1.crt как ключ и сертификат с паролем ADPG, установите права доступа для ключа.

```
$ openssl req -new -nodes -text -out postgres_1.csr -keyout postgres_1.key -subj "/CN=ssl-host-1"
$ chmod 600 postgres_1.key
$ openssl x509 -req -in postgres_1.csr -text -days 365 -CA root.crt -CAkey root.key -CAcreateserial -out postgres_1.crt
```

Сохраните файлы (ключ сервера: postgres_1.crt, ключ: postgres_1.key).

11. Проверьте сертификаты. Команды могут различаться для разных ОС.

```
$ openssl verify -CAfile root.crt postgres_1.crt
$ openssl verify -CAfile root.crt postgres_1.key
```

12. Измените разрешения для сертификата Arenadata PostgreSQL и экземпляра.

```
$ sudo chown postgres:postgres /etc/certificates/postgres_1.*
```

13. Скопируйте сертификаты CA в стандартный каталог установки для использования сертификатов доступа (по умолчанию этот каталог - /usr/local/share/ca-certificates). Создайте каталог и скопируйте сертификаты. Обратите внимание, что имена сертификатов должны заканчиваться .crt. Этот шаг необходим для включения, если хотя бы один из сертификатов не был сертификатом CA.

```
$ sudo mkdir /usr/local/share/ca-certificates/
$ sudo cp postgres_1.crt /usr/local/share/ca-certificates/
$ sudo cp root.crt /usr/local/share/ca-certificates/
$ sudo update-ca-certificates
```

14. Скопируйте файлы ключей, crt для postgres_1.crt как ключ и сертификат с паролем ADPG, установите права доступа для ключа.

```
$ sudo chmod 600 postgres_1.key
```

15. Создайте сертификат для клиента, с параметром **ADPG generic certificate key** и **ADPG generic certificate** для поля **Manage SSL**.

16. Скопируйте файлы ключей, crt для postgres_1.crt как ключ и сертификат с паролем ADPG, установите права доступа для ключа.

Пусть у вашего кластера есть два хоста: ssl-host-1 и ssl-host-2.

Пусть у вашего кластера есть два хоста: ssl-host-1 и ssl-host-2.

Пусть у вашего кластера есть два хоста: **Arenadata PostgreSQL certificate** и **Arenadata PostgreSQL certificate key** для поля **Manage SSL**.

Пусть у вашего кластера есть два хоста: **CA file** для поля **Manage SSL**.

17. Выполните шаг 5 в начале части о сервисе ADPG.

17. Выполните шаг 5 в начале части о сервисе ADPG.

Генерация SSL-сертификата для клиентов PostgreSQL

Создайте запрос сертификата (CSR), где **CN** - имя пользователя базы данных.

```
$ openssl req -new -nodes -out "CN=имя_пользователя_базы_данных" -keyout client_key -subj /client.pem
```

Например:

```
$ openssl req -new -nodes -out "CN=postgres" -keyout client_key -subj /client.pem
```

Создайте сертификат клиента для root.crt:

```
$ chown 600 client_key
```

Создайте подписанный сертификат клиента, используя корневой сертификат, сгенерированный в прежнем виде:

```
$ openssl x509 -days 3650 -req -CAcreateserial -in client.csr -CA root.crt -CAkey root.key -out client.crt
```

Замена файл CSR:

```
rm client.csr
```

Файлы client.crt и client.key должны храниться на клиентском хосте. См. Тест по curl.

После создания сертификатов и размещения их на клиентах необходимо настроить параметры сервера ADPG через пользовательский интерфейс ADCM:

1. Добавьте пары, разделенные CN сертификата, в поле **PG_HBA** раздела **SSL Configuration**. Например:

```
hostssl all all 0.0.0.0/0 cert clientcert=1
```

2. Установите параметр **SET_ENV_PTLS** в поле **postgresql.conf** раздела **ADPG Configuration**. Для этого примера значение будет следующим:

```
ssl_ca_file='/etc/certificates/root.crt'
```

3. Выполните действие [Reconfig & Restart](#) сервиса ADPG.

Роли и привилегии базы данных

Евгения Кузина

AD.PG

Содержание

Назад к содержанию

- Создание роли
- Атрибуты роли
- Членство в ролях
  - Команда SET ROLE
  - Атрибут INHERIT
- Удаление ролей

ADPG использует роли для управления разрешениями на доступ к базе данных. Роль представляет собой пользователя базы данных или группу пользователей. Роли могут владеть объектами базы данных, например, таблицами и функциями, разрешать доступ и действия над этими объектами (назначать привилегии). Также можно предоставить членство в роли другой роли, чтобы использовать её привилегии.

Роли базы данных отделены от пользователей операционной системы, но вы можете создавать роли, соответствующие пользователям операционной системы. Роли создаются глобально для всего кластера ADPG, а не для отдельной базы данных.

Кластер ADPG содержит предопределенную роль суперпользователя `postgres`. Чтобы создать дополнительные роли, подключитесь к хосту ADPG как `postgres`.

Создание роли

Чтобы создать роль, используйте SQL-команду CREATE ROLE. Вы можете выполнять команды в командной строке psql.

```
CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
```

Приведенная выше команда создает роль `user1` с паролем `password1`. Параметр LOGIN разрешает использовать `user1` для первоначального подключения к базе данных.

Если вам нужно использовать специальные символы в имени роли, заключите имя в двойные кавычки. Дополнительные сведения об идентификаторах SQL вы можете найти по ссылке Identifiers and keywords.

Чтобы создать роль, вы также можете выполнить команду createuser из командной строки терминала. `createuser` — это обертка SQL-команды `CREATE ROLE`. Подключитесь к хосту ADPG как суперпользователь или пользователь с привилегией CREATEROLE, чтобы выполнить команду `createuser`.

```
$ createuser user2
```

Представление pg_roles предоставляет доступ к информации о ролях в базах данных. Вы можете выполнить следующий запрос, чтобы проверить добавлена ли роль:

```
SELECT rolname FROM pg_roles;
```

Роли `user1` и `user2` должны быть в результате запроса вместе с предопределенными ролями PostgreSQL:

```
pg_database_owner
pg_read_all_data
pg_write_all_data
pg_monitor
pg_read_all_settings
pg_read_all_stats
pg_stat_scan_tables
pg_read_server_files
pg_write_server_files
pg_execute_server_program
pg_signal_backend
postgres
user1
user2
```

Метакоманда psql `\du` также выводит список существующих ролей.

ПРИМЕЧАНИЕ

Только пользователь с правами суперпользователя может создать новую роль суперпользователя или предоставить привилегию суперпользователя.

Атрибуты роли

Роль может иметь атрибуты, которые определяют её привилегии и взаимодействуют с системой аутентификации клиентских приложений.

Атрибуты роли

Наименование	Описание	Пример
LOGIN	Роль можно использовать для первоначального подключения к базе данных (эквивалент команды CREATE USER)	`CREATE ROLE name LOGIN;`
SUPERUSER	Суперпользователь базы данных, который проходит все проверки разрешений, кроме права входа в систему. Поскольку эта роль имеет привилегии высокого уровня, сведите к минимуму использование этой роли из соображений безопасности	`CREATE ROLE name SUPERUSER;`
CREATEDB	Роль может создавать базы данных	`CREATE ROLE name CREATEDB;`
CREATEROLE	Роль может создавать, изменять и удалять другие роли, предоставлять или отзывать членство в ролях	`CREATE ROLE name CREATEROLE;`
REPLICATION	Роль имеет разрешение инициировать потоковую репликацию. Эта роль также должна иметь привилегию LOGIN	`CREATE ROLE name REPLICATION LOGIN;`
PASSWORD	Используйте этот атрибут, если для метода аутентификации клиента требуется пароль. Дополнительную информацию можно найти в статье Аутентификация в ADPG по паролю	`CREATE ROLE name PASSWORD 'password_string';`

ПРИМЕЧАНИЕ

Мы рекомендуем создать роль с привилегиями CREATEDB или CREATEROLE вместо роли суперпользователя и использовать эту роль для рутинных операций. Такой подход увеличивает надежность системы.

Используйте команду ALTER ROLE для изменения атрибутов роли после создания.

Следующая команда изменяет пароль роли `user1`.

```
ALTER ROLE user1 WITH PASSWORD 'password2';
```

Вы также можете добавить привилегии CREATEDB и CREATEROLE роли `user2`.

```
ALTER ROLE user2 CREATEROLE CREATEDB;
```

Членство в ролях

Вы можете группировать пользователей для управления привилегиями. Для этого создайте групповую роль:

```
CREATE ROLE group1 CREATEROLE CREATEDB;
```

Групповая роль от индивидуальной пользовательской роли отличается только способом использования. Также групповые роли обычно не имеют атрибута LOGIN.

Выполните команду GRANT, чтобы добавить участников в групповую роль. Приведенная ниже команда добавляет `user1` и `user2` в `group1`.

```
GRANT group1 TO user1, user2;
```

Чтобы удалить участника группы, вызовите команду REVOKE.

```
REVOKE group1 FROM user1;
```

Чтобы использовать привилегии группы, вызовите команду SET ROLE или добавьте атрибут `INHERIT` роли участника группы.

Команда SET ROLE

Войдите в систему как участник группы и выполните команду SET ROLE.

```
SET ROLE 'group1';
```

В результате участник группы временно "становится" ролью `group1`. Его сессия имеет доступ к привилегиям роли группы и создает объекты базы данных от имени роли группы — владельцем всех созданных объектов будет `group1`.

Атрибут INHERIT

Ролям участников группы с атрибутом `INHERIT` автоматически назначаются привилегии группы. В приведенном ниже примере `user3` может использовать привилегии `group1` сразу после входа в систему.

```
CREATE ROLE user3 LOGIN INHERIT;
GRANT group1 TO user3;
```

Удаление ролей

Чтобы удалить роль, удалите или переназначьте владельца всем объектам, которыми эта роль владеет. Предоставленные разрешения также должны быть отозваны.

Вы можете использовать команду ALTER для переназначения владельца объекта. Приведенная ниже команда назначает владельцем таблицы базы данных `table1` роль `user2`.

```
ALTER TABLE table1 OWNER TO user2;
```

Также вы можете выполнить команду REASSIGN OWNED, чтобы переназначить владельца всем объектам роли. Команда REASSIGN OWNED передает объекты базы данных новому владельцу, но не влияет на предоставленные привилегии для объектов, не принадлежащих роли. Вызовите команду DROP OWNED, чтобы удалить эти привилегии.

```
REASSIGN OWNED BY user1 TO user2;
DROP OWNED BY user1;
```

Поскольку приведенные выше команды не могут получить доступ к объектам в других базах данных, выполните их для каждой базы данных, содержащей объекты, принадлежащие роли.

После удаления всех зависимостей вызовите команду DROP ROLE, чтобы удалить роль:

```
DROP ROLE user1;
```

Вы также можете выполнить команду dropuser из командной строки терминала, чтобы удалить роль. `dropuser` — это обертка SQL-команды `DROP ROLE`. Подключитесь к хосту ADPG как суперпользователь или пользователь с привилегией CREATEROLE, чтобы выполнить команду `dropuser`.

```
$ dropuser user1
```

Управление доступом на уровне строк и столбцов

Евгения Кузина

Назад к содержанию

В дополнение к стандартной SQL-системе привилегий PostgreSQL предоставляет политики безопасности строк и дополнительные возможности для настройки безопасности столбцов.

Эти методы полезны, если вам нужно ограничить доступ пользователей к определенным строкам или столбцам в таблице. За подробным описанием обратитесь к статьям:

- Политики защиты строк
- Безопасность на уровне столбцов

Политики защиты строк

Евгения Кузина

Содержание

Назад к содержанию

Включение защиты строк

Управление политиками безопасности строк

Отключение защиты строк

К таблице можно применить **политики защиты строк (row security policies)**. Эти политики определяют, какие строки могут быть доступны для каждого пользователя. Вы можете создать политику для определенных команд, таких как `SELECT`, `INSERT`, `UPDATE` и `DELETE`, или указать её для всех команд (`ALL`). Этот функционал также известен как **безопасность на уровне строк (row-level security)**.

Подключимся к базе данных как пользователь `postgres` и создадим таблицу для примеров. Для этого можно использовать psql:

```
$ psql -h 10.92.6.36 -p 5432 -U postgres postgres
```

В приведенном выше примере `10.92.6.36` — адрес сервера, `5432` — порт для подключения, `-U postgres` — имя пользователя и `postgres` — имя базы данных.

```
CREATE TABLE users (
user_id serial PRIMARY KEY,
user_name  VARCHAR NOT NULL,
pwhash VARCHAR,
real_name  VARCHAR NOT NULL
);
```

Добавим данные в таблицу:

```
INSERT INTO users (user_name, pwhash, real_name)
  VALUES
  ('james', 'xxx', 'James Brown'),
  ('mary', 'xxx', 'Mary Smith'),
  ('alice', 'xxx', 'Alice Gray');
```

Создадим роли и воспользуемся командой GRANT, чтобы определить их привилегии доступа к таблице `users`.

```
CREATE ROLE james WITH LOGIN PASSWORD 'password1';
GRANT ALL ON users TO james;

CREATE ROLE mary WITH LOGIN PASSWORD 'password2';
GRANT ALL ON users TO mary;
```

Пользователи `james` и `mary` имеют привилегию `ALL` над таблицей `users`.

Включение защиты строк

Для того чтобы включить защиту на уровне строк, выполним команду ALTER TABLE с параметром `ENABLE ROW LEVEL SECURITY`.

```
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
```

Если для таблицы включена защита на уровне строк, доступ к таблице для выбора или изменения строк должен быть разрешен политикой безопасности. Если для таблицы не определена политика, доступ запрещен: строки не могут быть выбраны или изменены. Политика защиты строк не влияет на операции, применяемые ко всей таблице, например `TRUNCATE` или `REFERENCES`.

Политики защиты строк не распространяются на суперпользователей и роли с атрибутом `BYPASSRLS`. Также они не влияют на владельцев таблиц, но владельцы могут включить их для себя с помощью команды `ALTER TABLE … FORCE ROW LEVEL SECURITY`.

Переподключимся к базе данных как пользователь `james`.

```
$ psql -h 10.92.6.36 -p 5432 -U james postgres
```

Если попытаемся выбрать строки из таблицы `users`, получим следующий результат:

```
SELECT * FROM users;
user_id | user_name | pwhash | real_name
---------+-----------+--------+-----------
(0 rows)
```

Пользователь `james` не может получить доступ к данным таблицы, так как включена защита на уровне строк и не создана политика безопасности.

> **ПРИМЕЧАНИЕ**
>
> Только владелец таблицы может включать и отключать защиту строк и добавлять политики таблице.

Управление политиками безопасности строк

Политики безопасности строк могут быть указаны для команд, для ролей или для тех и других вместе. Политика может быть применена ко всем командам (`ALL`) или к командам `SELECT`, `INSERT`, `UPDATE` или `DELETE`. Текущей политике можно назначить несколько ролей. За подробностями обратитесь к статье CREATE POLICY.

PostgreSQL также применяет стандартные правила членства и наследования ролей к политикам безопасности строк.

У каждой политики есть имя. Для одной таблицы можно определить несколько политик. Политики относятся к таблице, политики одной таблицы должны иметь уникальные имена. К разным таблицам могут относиться политики с одинаковыми именами.

Переподключимся к базе данных как пользователь `postgres` и определим следующую политику безопасности строк:

```
$ psql -h 10.92.6.36 -p 5432 -U postgres postgres
```

```
CREATE POLICY policy1 ON users FOR ALL TO PUBLIC USING (user_name=current_user);
```

Эта политика позволяет всем ролям выполнять все команды, если значение из столбца `user_name` обрабатываемой строки таблицы `users` равно текущему имени пользователя.

`PUBLIC` используется вместо имени роли и применяет политику ко всем ролям.

Выражение `USING` представляет собой любое выражение SQL, возвращающее логическое значение.

Согласно этой политике пользователи `james` и `mary`, созданные выше, могут обрабатывать строки, содержащие информацию о каждом из них.

Переподключимся к базе данных как пользователь `james` и выберем данные:

```
$ psql -h 10.92.6.36 -p 5432 -U james postgres
```

```
SELECT * FROM users;
```

Результат:

```
user_id | user_name | pwhash |  real_name
---------+-----------+--------+-------------
       1 | james     | xxx    | James Brown
```

Переподключимся к базе данных как пользователь `mary` и выберем данные:

```
$ psql -h 10.92.6.36 -p 5432 -U mary postgres
```

```
SELECT * FROM users;
```

Результат:

```
user_id | user_name | pwhash | real_name
--------+-----------+--------+-----------
      2 | mary      | xxx    | Mary Smith
```

Переподключимся к базе данных как пользователь `postgres`, создадим нового пользователя `alice` и добавим новую политику:

```
$ psql -h 10.92.6.36 -p 5432 -U postgres postgres
```

```
CREATE ROLE alice WITH LOGIN PASSWORD 'password3';
GRANT ALL ON users TO alice;

CREATE POLICY policy2 ON users TO alice USING (true) WITH CHECK (true);
```

Выражение `WITH CHECK` должно возвращать `true`, чтобы пользователь смог вставлять и обновлять строки.

Эта политика разрешает пользователю `alice` все операции над всеми строками таблицы.

Переподключимся к базе данных как `alice` и проверим результат:

```
$ psql -h 10.92.6.36 -p 5432 -U alice postgres
```

```
DELETE FROM users WHERE user_id=1;

SELECT * FROM users;
```

Результат:

```
user_id | user_name | pwhash |  real_name
---------+-----------+--------+-------------
      2 | mary      | xxx    | Mary Smith
      3 | alice     | xxx    | Alice Gray
```

Вы также можете использовать команду ALTER POLICY для обновления политики. Следующий код переименовывает `policy1` в `policy_for_users`.

```
ALTER POLICY policy1 ON users RENAME TO policy_for_users;
```

Команда DROP POLICY удаляет политику:

```
DROP POLICY policy_for_users ON users;
```

Отключение защиты строк

Чтобы отключить защиту строк, выполним команду `ALTER TABLE` с параметром `DISABLE ROW LEVEL SECURITY`.

```
ALTER TABLE users DISABLE ROW LEVEL SECURITY;
```

Безопасность на уровне столбцов

Евгений Кузина

AD.PG

Содержание

Назад к содержанию

Создание представления

Предоставление прав доступа на определенные столбцы

Шифрование столбцов

PostgreSQL позволяет ограничить доступ пользователя к определенному столбцу или набору столбцов, чтобы пользователь не мог просматривать и изменять данные столбца.

Реализовать безопасность на уровне столбцов можно одним из следующих способов:

- Создание представления
- Предоставление прав доступа на определенные столбцы
- Шифрование столбцов

Подключимся к базе данных как пользователь `postgres` и создадим таблицу для примеров. Для этого можно использовать psql:

```
$ psql -h 10.92.6.36 -p 5432 -U postgres postgres
```

В приведенном выше примере `10.92.6.36` – адрес сервера, `5432` – порт для подключения, `-U postgres` – имя пользователя и `postgres` – имя базы данных.

```
CREATE TABLE users (
user_id serial PRIMARY KEY,
user_name VARCHAR NOT NULL,
real_name VARCHAR NOT NULL,
address VARCHAR,
salary INT
);
```

Добавим данные в таблицу:

```
INSERT INTO users (user_name, real_name, address, salary)
    VALUES
    ('james', 'James Brown', '123 2nd Street', 5000),
    ('mary', 'Mary Smith', '456 13th Street', 4000),
    ('alice', 'Alice Gray', '789 19th Street', 5700);
```

Создадим роли:

```
CREATE ROLE user1 WITH LOGIN PASSWORD 'password1';
CREATE ROLE user2 WITH LOGIN PASSWORD 'password2';
```

Создание представления

Можно создать представление, которое включает только разрешенные столбцы, и дать доступ пользователю с помощью команды GRANT к этому представлению, а не к базовой таблице. Например, мы можем скрыть столбец `salary` от пользователя `user1`. Создадим представление со столбцами `user_name`, `real_name` и `address` и предоставим права доступа пользователю `user1` к этому представлению:

```
CREATE VIEW users_info AS SELECT user_name, real_name, address FROM users;
GRANT ALL ON users_info TO user1;
```

Переподключимся к базе данных как пользователь `user1`:

```
$ psql -h 10.92.6.36 -p 5432 -U user1 postgres
```

Просмотрим данные из представления `users_info`:

```
SELECT * FROM users_info;
```

Поскольку `user1` имеет права доступа к этому представлению, результат будет следующим:

```
 user_name |  real_name  |     address
-----------+-------------+-----------------
 james     | James Brown | 123 2nd Street
 mary      | Mary Smith  | 456 13th Street
 alice     | Alice Gray  | 789 19th Street
```

Если попытаться выбрать все данные из таблицы `users` от имени `user1`, произойдет ошибка, так как пользователь `user1` не имеет прав доступа к таблице.

```
SELECT * FROM users;
```

Результат:

```
ERROR: permission denied for table users
```

Предоставление прав доступа на определенные столбцы

Можно предоставить пользователю доступ только к определенным столбцам таблицы. Например, мы можем скрыть столбец `salary` от пользователя `user2`. Переподключимся к базе данных как пользователь `postgres` и добавим разрешения пользователю `user2` на доступ ко всем столбцам таблицы `users`, кроме `salary`.

```
$ psql -h 10.92.6.36 -p 5432 -U postgres postgres
```

```
GRANT SELECT (user_name, real_name, address) ON users TO user2;
```

Переподключимся к базе данных как пользователь `user2`:

```
$ psql -h 10.92.6.36 -p 5432 -U user2 postgres
```

Выберем данные из разрешенных столбцов таблицы `users`:

```
SELECT user_name, real_name, address FROM users;
```

Результат:

```
 user_name |  real_name  |     address
-----------+-------------+-----------------
 james     | James Brown | 123 2nd Street
 mary      | Mary Smith  | 456 13th Street
 alice     | Alice Gray  | 789 19th Street
```

Обратите внимание, что если `user2` попытается выбрать все столбцы из таблицы `users`, возникнет ошибка:

```
SELECT * FROM users;
```

Результат:

```
ERROR: permission denied for table users
```

Шифрование столбцов

Также можно зашифровать определенные столбцы базы данных для их защиты. Для этого используем дополнительный модуль pgcrypto. За подробной информацией об установке и использовании этого модуля обратитесь к статье Шифрование.

В приведенном ниже примере используются функции `pgp_sym_encrypt` и `pgp_sym_decrypt` из модуля pgcrypto.

Переподключимся к базе данных как пользователь `user1` и создадим таблицу `users_encryption`:

```
$ psql -h 10.92.6.36 -p 5432 -U user1 postgres
```

```
CREATE TABLE users_encryption (
user_id serial PRIMARY KEY,
user_name VARCHAR NOT NULL,
real_name VARCHAR NOT NULL,
address VARCHAR,
salary TEXT
);
```

Добавим данные и зашифруем столбец `salary` с помощью функции `pgp_sym_encrypt`:

```
INSERT INTO users_encryption (user_name, real_name, address, salary)
    VALUES
    ('james', 'James Brown', '123 2nd Street', pgp_sym_encrypt('5000','secret_encryption_key')),
    ('mary', 'Mary Smith', '456 13th Street', pgp_sym_encrypt('4000','secret_encryption_key')),
    ('alice', 'Alice Gray', '789 19th Street', pgp_sym_encrypt('5700','secret_encryption_key'));
```

Выберем данные, чтобы проверить, зашифрован ли столбец `salary`:

```
SELECT * FROM users_encryption;
```

Результат:

```
user_id | user_name |  real_name  |     address     |           salary
--------+-----------+-------------+-----------------+-----------------------------------------------------------
      1 | james     | James Brown | 123 2nd Street  | \xc30d04070302ec410a9f16a58767ed23501d9679...
      2 | mary      | Mary Smith  | 456 13th Street | \xc30d04070302e972f63c44aa0b8270d23501ce960...
      3 | alice     | Alice Gray  | 789 19th Street | \xc30d04070302edf9f8629c8fb48669d23501560fe...
```

Переподключимся к базе данных как пользователь `postgres` и разрешим доступ к таблице `users_encryption` пользователю `user2`:

```
$ psql -h 10.92.6.36 -p 5432 -U postgres postgres
```

```
GRANT ALL ON users_encryption TO user2;
```

Переподключимся как `user2` и выберем все столбцы из `users_encryption`:

```
$ psql -h 10.92.6.36 -p 5432 -U user2 postgres
```

```
SELECT * FROM users_encryption;
```

Результат будет таким же:

```
user_id | user_name |  real_name  |     address     |           salary
--------+-----------+-------------+-----------------+-----------------------------------------------------------
      1 | james     | James Brown | 123 2nd Street  | \xc30d04070302ec410a9f16a58767ed23501d9679...
      2 | mary      | Mary Smith  | 456 13th Street | \xc30d04070302e972f63c44aa0b8270d23501ce960...
      3 | alice     | Alice Gray  | 789 19th Street | \xc30d04070302edf9f8629c8fb48669d23501560fe...
```

Вызовем функцию `pgp_sym_decrypt` в операторе `SELECT`, чтобы просмотреть расшифрованные данные из столбца `salary`:

```
SELECT user_id, user_name, real_name, address, pgp_sym_decrypt(salary::bytea, 'secret_encryption_key') AS salary FROM
users_encryption;
```

Результат:

```
user_id | user_name |  real_name  |     address     | salary
--------+-----------+-------------+-----------------+--------
      1 | james     | James Brown | 123 2nd Street  | 5000
      2 | mary      | Mary Smith  | 456 13th Street | 4000
      3 | alice     | Alice Gray  | 789 19th Street | 5700
```

Обратите внимание, что любой пользователь, у которого есть ключ шифрования, может расшифровать данные столбца. В приведенном выше примере пользователь `user1` шифрует данные, а `user2` расшифровывает данные с помощью ключа шифрования.

Управление кластером ADPG через ADCM

Евгения Кузина

Назад к содержанию

Вы можете использовать ADCM, чтобы управлять кластером ADPG.

Следующие действия доступны в пользовательском интерфейсе ADCM:

- Действия с кластером, которые позволяют управлять кластером со всеми сервисами.
- Действия с сервисом, управляющие определенным сервисом.

Кластерные действия

Евгений Кузина

Содержание

Назад к содержанию

Обзор

Check и Precheck

Manage SSL

Reinstall

Reinstall statuschecker

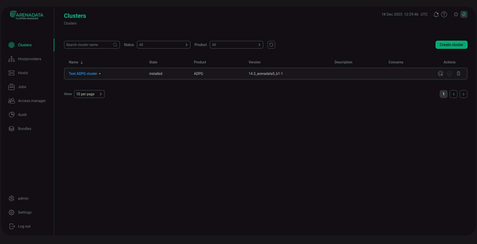
Start

Stop

Статья описывает действия, которые можно выполнять с кластером ADPG в пользовательском интерфейсе ADCM.

Обзор

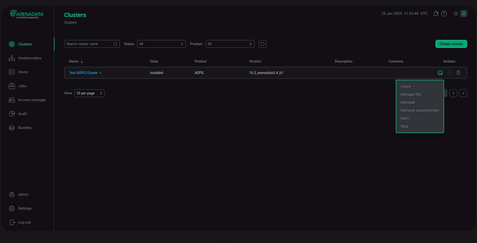
Действия выполняются на странице ***Clusters***.



Страница "Clusters"

Описание страницы ***Clusters*** и общих действий на этой странице приведено в документации ADCM.

Нажмите в столбце ***Actions***, чтобы открыть список с доступными действиями, и выберите действие для выполнения.



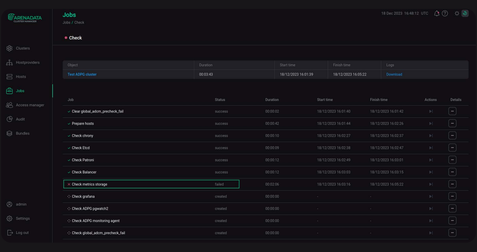
Действия

Check и Precheck

Действие ***Precheck*** проверяет соответствие настроек хостов, компонентов и сервисов требованиям кластера ADPG. ***Precheck*** доступен перед установкой кластера.

Действие ***Check*** включает в себя ***Precheck***, а также проверяет, работают ли установленные компоненты кластера правильно. ***Check*** доступен после установки кластера.

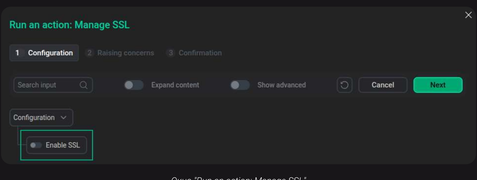
Результат действия вы можете найти на странице ***Jobs***. Если действие завершается успешно, соответствующая ему задача на странице ***Jobs*** переходит из статуса `running` в `success`, в противном случае — в статус `failed`. Также действия сохраняют результат проверки в переменную ADCM `global_adcm_precheck_fail`, реализуя механизм отложенного сбоя. Значение переменной `true`, если последнее действие ***Check*** или ***Precheck*** успешно. Значение `false` указывает на то, что во время проверки произошла ошибка. В этом случае статус операции ***Check global_adcm_precheck_fail*** меняется на `failed`. В некоторых случаях, если задача ***Check global_adcm_precheck_fail*** завершилась сбоем, необходимо просмотреть внутренние шаги действия ***Check*** или ***Precheck***, отмеченные как успешные, чтобы определить причину ошибки.



При выполнении действия возникла ошибка

Manage SSL

Действие ***Manage SSL*** включает или отключает SSL в кластере ADPG. Чтобы включить SSL, активируйте переключатель ***Enable SSL*** в окне ***Run an action: Manage SSL*** и следуйте инструкциям из статьи Включение SSL-шифрования.

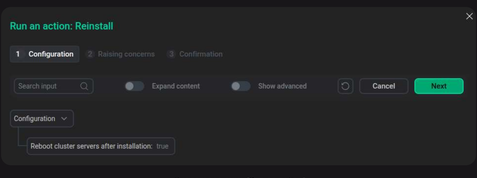


Окно "Run an action: Manage SSL"

Чтобы отключить SSL, выключите переключатель ***Enable SSL*** и выполните действие ***Manage SSL***, нажав ***Run*** на вкладке ***Confirmation***.

Reinstall

Действие ***Reinstall*** переустанавливает кластер ADPG. В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после установки.



Диалоговое окно "Run an action"

Reinstall statuschecker

Действие ***Reinstall statuschecker*** перенастраивает и перезапускает службу проверки состояния для всех сервисов кластера.

Start

Действие ***Start*** запускает все сервисы кластера.

Stop

Действие ***Stop*** останавливает все сервисы кластера.

Обновление кластера ADPG до новой версии

Ян Сергеевич Пучков

Содержание

Назад к содержанию

Подготовка кластера к обновлению
Обновление минорной версии
Обновление мажорной версии

ВНИМАНИЕ

- Настоятельно рекомендуется создавать резервную копию перед обновлением, особенно если вы собираетесь обновить ADPG до новой мажорной версии.
- Перед выполнением обновления необходимо запустить кластерную проверку [Deck](#) и убедиться в отсутствии на его шагах проверок со статусом `Failed`. При обнаружении ошибок следует их исправить и только затем запускать апгрейд.
- При обновлении с версии v1.5.3.1 включительно SSL рекомендуется отключить SSL, затем выполнить обновление до v16.3.4 и использовать кластерную работу **Manage SSL** для включения SSL.

В настоящем врезке ADPG поддерживает обновление минорных версий, а также обновление версий с v14.3.8 до v16.3.1, v15.3.3. ADPG v16.3 использует PostgreSQL 16.3.

Возможны следующие варианты мажорного обновления Enterprise Edition (EE) и Community Edition (CE):

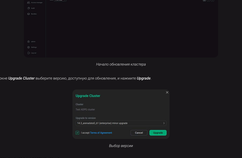
- с ADPG EE v14.3 на ADPG EE v16.3 / v15.3.3.
- с ADPG CE v14.3 на ADPG CE v16.3 / v15.3.3.

Если вам необходимо обновить EE v14.3.7 и более ранние версии до EE v16.3.1 / v15.3.3, сначала обновите EE v14.4 до EE v14.3.8, а затем обновите EE v14.3.8 до EE v16.3.1 / v15.3.3.

Если вам необходимо обновить EE v14.3.8 до EE v15.3.4 или выше, сначала обновите EE v14.3.8 до EE v16.3.1, а затем обновите EE v16.3.3 до EE v16.3.4 или выше.

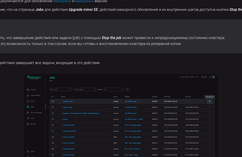
Подготовка кластера к обновлению

Загрузите новый бандл. Для этого перейдите на страницу **Bundles**, нажмите **Upload bundle** и выберите новый файл бандла в своем окружении.



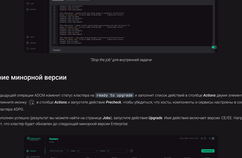
Страница "Bundles"

После загрузки нового бандла перейдите на страницу **Clusters**. В столбце **Actions** кликните иконку ⓘ, которая станет активной.



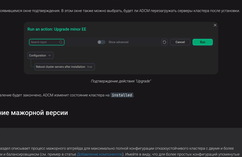
Начало обновления кластера

В открывшемся окне **Upgrade Cluster** выберите версию, доступную для обновления, и нажмите **Upgrade**.



Выбор версии

Нажмите **Run** в появившемся окне подтверждения.



Подтверждение обновления

Кластер будет подготовлен к обновлению. Вы можете открыть страницу **Jobs**, чтобы просмотреть выполнение операции и проверить, прошла ли подготовка кластера успешно.

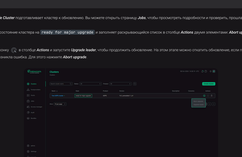
Остальные шаги аналогичны для обновления [минорной](#) и [мажорной](#) версий.

Обратите внимание, что на странице **Jobs** для действия **Upgrade minor EE** доступ для мажорного обновления и из внутреннего шелла доступна кнопка **Stop the job**.

ВНИМАНИЕ

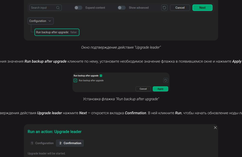
Важно помнить, что завершение действия или задачи (job) с помощью **Stop the job** может привести к непредсказуемому состоянию кластера. Используйте эту возможность только в том случае, если вы готовы к восстановлению кластера из резервной копии.

Stop the job для действия завершает все задачи, входящие в это действие.



Stop the job для действия

Stop the job для внутренней задачи действия останавливает текущую задачу, а действие продолжает выполнительчи начинает с задачи, следующей за завершенной

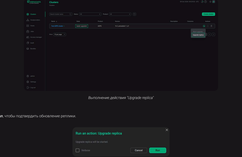


Stop the job для внутренней задачи

Обновление минорной версии

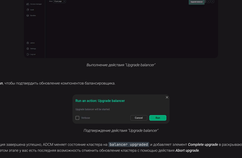
В результате предыдущей операции ADCM изменит статус кластера на `ready to upgrade` и заполнит список действий в столбце **Actions** двумя элементами: **Precheck** и **Upgrade_minor**. Кликните иконку ⚙ в столбце **Actions** и запустите действие **Precheck**, чтобы убедиться, что кластер, компоненты и сервисы настроены в соответствии с требованиями нового кластера ADPG.

Если **Precheck** выполнен успешно (результат вы можете найти на странице **Jobs**), запустите действие **Upgrade**. Оно действие включает версии CE/EE. Например, **Upgrade Minor EE** означает, что кластер будет обновлен до следующей минорной версии Enterprise.



Выполнение действия "Upgrade"

Нажмите **Run** в появившемся окне подтверждения. В этом окне также можно выбрать, будет ли ADCM перезагружать сервер кластера после установки.



Подтверждение действия "Upgrade"

Как только обновление будет закончено, ADCM изменит состояние кластера на `installed`.

Обновление мажорной версии

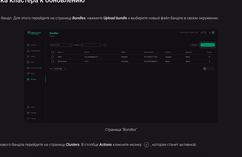
ВНИМАНИЕ

- Данный раздел описывает процесс мажорного апгрейда для максимально полной конфигурации отказоустойчивого кластера с двумя и более репликами и балансировщиком (см. пример в статье [Автоматизация отказоустойчивости](#)). Имейте в виду, что если более простая конфигурация удовлетворяет ваши действующие потребности, вы можете использовать мажорный апгрейд могут потребоваться небольшие отличия.
- Запрещено изменять конфигурацию кластера между этапами мажорного обновления. Обновление конфигурации или топологии кластера может произвольным и только после того, как кластер будет переведен в состояние `installed`.
- Перед обновлением мажорной версии, что у вашего кластера нет сервисов из импортированных кластеров. Для этого перейдите на вкладку **Import** объекта кластера и убедитесь, что в списке импортов данных с вами партнеров нет данных, или они существуют. Если это существует конфигурация, нажав на **Save**.
- Также следует учесть, что если у вас использовался кластер без реплик и апгрейд завершился с ошибкой, действие **Abort upgrade** не будет доступно и восстановление данных будет возможно только из резервной копии.
- Если вам необходимо вызвать действие **Abort upgrade**, после его успешного завершения следует запустить действие сервиса ADPG **Reconfigure & Restart** в штате — действие помогает Monitoring **Reconfigure and restart** для Monitoring использовать. В противном случае действие **Check** может завершиться неудачей.

Действие **Upgrade** Cluster подготавливает кластер к обновлению. Вы можете открыть страницу **Jobs**, чтобы просмотреть подробности и проверить, прошла ли операция успешно.

ADCM изменяет состояние кластера на `ready for main upgrade`, а заполнит раскрывающийся список в столбце **Actions** двумя элементами **Abort upgrade** и **Upgrade leader**.

1. Нажмите иконку ⚙ в столбце **Actions** и запустите **Upgrade leader**, чтобы продолжить обновление. На этом этапе можно открыть обновление, если при подготовке кластера возникли ошибки. Для этого нажмите **Abort upgrade**.



Действие для мажорного обновления

2. В появившемся окне на вкладке **Configuration** доступна опция **Run backup after upgrade**. Если не снимать установку в `True`, то создание резервной копии включено в кластере. Если вы не хотите создавать резервную копию, снимите установку с настройки. управления в модуле **Enable backup** [конфигурации](#) параметра сервиса ADPG. Если **Run backup after upgrade** включен (`True`), то создание резервных копий отключено, то в тот будет добавлена в невозможности создания бэкапа.

Окно подтверждения действия "Upgrade leader"

Для изменения значения **Run backup after upgrade** измените по нему, установите необходимое значение флажка в появившемся окне и нажмите **Apply**.

Установка флажка "Run backup after upgrade"

В окне подтверждения действия **Upgrade leader** измените **Next** → открыта во вкладка **Confirmation**. В ней кликните **Run**, чтобы начать обновление ноды лидера.

Вкладка "Confirmation" окна "Upgrade leader"

Если действие **Upgrade leader** прошло успешно, ADCM изменит состояние кластера на `leader upgraded`, а добавит элемент **Upgrade replica** в раскрывающийся список столбца **Actions**. Если обновление лидера завершилось ошибкой, запустите действие **Abort upgrade**, чтобы отменить изменения в кластере, и перейдите на страницу **Jobs**, где вы можете проверить ошибки.

3. Нажмите иконку ⚙ в столбце **Actions** и запустите **Upgrade Replica**. **Upgrade Replica** обновляет одну реплику из кластера. Эта реплика выбирается случайным образом. Изменения вносятся только на одной реплике, чтобы в случае возникновения проблем при обновлении в кластер оставалась работоспособная реплика с прошлой версией.

Выполнение действия "Upgrade replica"

4. Нажмите **Run**, чтобы подтвердить обновление реплики.

Подтверждение действия "Upgrade replica"

Если действие **Upgrade replica** завершается успешно, ADCM изменит состояние кластера на `replica upgraded`, а добавляет элемент **Upgrade Balancer** в раскрывающемся списке **Actions**. Не стоит этого вы также можете использовать **Abort upgrade** для отмены изменений.

5. Выберите **Upgrade balancer** из выпадающего списка **Actions**. **Upgrade balancer** обновляет конфигурацию балансировщика на обновленной реплике.

Выполнение действия "Upgrade balancer"

6. Нажмите **Run**, чтобы подтвердить обновление компонентов балансировщика.

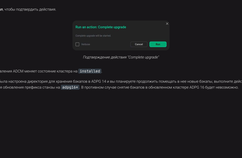
Подтверждение действия "Upgrade balancer"

Если операция завершена успешно, ADCM изменит состояние кластера на `balancer upgraded`, а добавляет элемент **Complete upgrade** в раскрывающийся список **Actions**. На этом этапе у вас есть последняя возможность отменить обновление кластера с помощью действия **Abort upgrade**.

7. Выберите **Complete upgrade** из раскрывающегося списка в столбце **Actions**. **Complete upgrade** завершает обновление кластера — обновляются оставшиеся реплики, и компоненты.

Выполнение действия "Complete upgrade"

8. Нажмите **Run**, чтобы подтвердить действие.



Подтверждение действия "Complete upgrade"

После обновления ADCM изменит состояние кластера на `installed`.

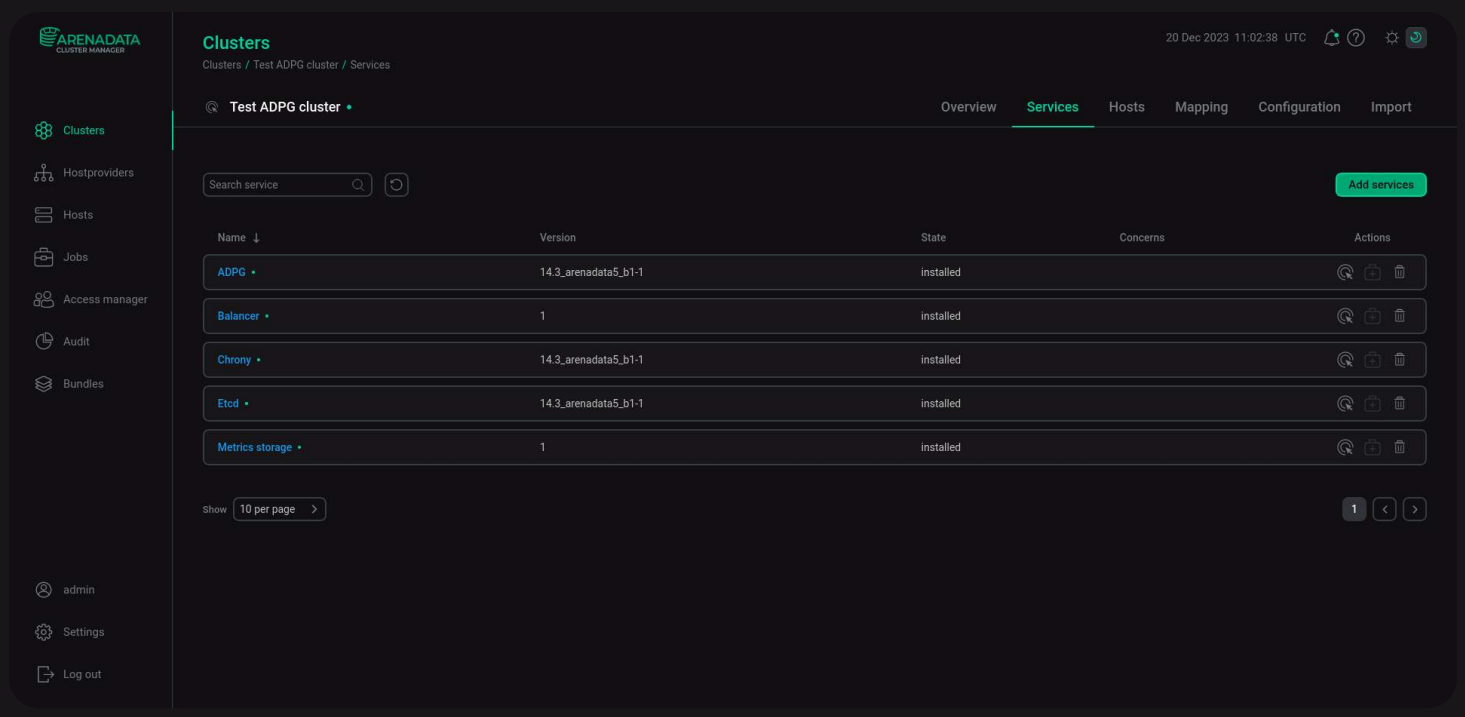
9. Если у вас была настроена директория для хранения бэкапов в ADPG 14 и вы планируете продолжить пользоваться ею новым бэкапы, выполните действие Reconfigure & Restart для обновления профиля стенды на `adpg14` в протоколе случае сначала бэкапов в обновленном кластере ADPG 16 будет невозможно.

Сервисные действия

Евгения Кузина

Назад к содержанию

Пользовательский интерфейс ADCM позволяет выполнять действия для управления сервисами. Действия доступны на вкладке ***Services***. Чтобы перейти на эту вкладку, кликните название кластера на странице ***Clusters*** и откройте вкладку ***Services***.



Сервисы кластера ADPG в интерфейсе ADCM

Описание вкладки ***Services*** и общих действий на этой вкладке приведено в документации ADCM.

Следующие статьи рассказывают о действиях для каждого сервиса кластера ADPG:

- ADPG
- ADPG Control Agents
- Balancer
- Chrony
- Etcd
- Monitoring

Действия сервиса ADPG

Д. Евгения Кузина

Содержание

Назад к содержанию

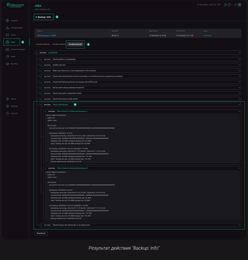
Backup: Info
Backup: Restore cluster
Backup: Start
Check
Resume Cluster
Pause Cluster
Expand
Reconfigure & Restart
Reinstall
Start
Stop

В этой статье описаны действия, доступные для сервиса ADPG.

Backup: Info

Действие **Backup: Info** отображает список резервных копий (бэкапов) кластера. Действие выполняется от имени пользователя `postgres`.

Чтобы просмотреть результат, откройте страницу **Jobs**, кликните строку **Backup: Info** в списке задач, переключитесь на вкладку задачи **Ansible [check]**, найдите строку **Check valid backup** и кликните её, чтобы раскрыть. Бэкапы отсортированы по нодам.



Результат действия 'Backup: Info'

За дополнительной информацией о создании бэкапов обратитесь к статье Резервное копирование и восстановление с использованием pgBackRest.

Backup: Restore cluster

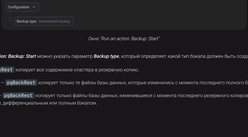
Действие **Backup: Restore cluster** восстанавливает кластер на всех нодах из последнего бэкапа. Действие выполняется от имени пользователя `postgres`.

> **ВАЖНО**
>
> Действие **Backup: Restore cluster** восстанавливает состояние кластера на конец архива WAL. Если произошло случайное удаление объектов базы данных, используйте Точное PITR-восстановление с параметром `--target`, так как наиболее вероятно, что действие **Backup: Restore Cluster** восстанавит состояние кластера на тот момент, когда объекты базы данных уже были удалены.

Backup: Start

Действие **Backup: Start** позволяет создать резервную копию кластера. Действие выполняется от имени пользователя `postgres`.

Когда вы выберете **Backup: Start**, ADCM отображает диалоговое окно для указания параметров действия.



Окно 'Run an action: Backup: Start'

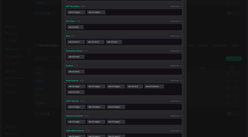
В диалоговом окне **Run an action: Backup: Start** можно указать параметр **Backup type**, который определяет какой тип бэкапа должен быть создан. Доступные значения:

- `full backup - pgbackbest` - копирует все содержимое кластера в резервную копию.
- `differential backup - pgbackbest` - копирует только те файлы базы данных, которые изменились с момента последнего полного бэкапа (full backup).
- `incremental backup - pgbackbest` - копирует только файлы базы данных, изменившиеся с момента последнего резервного копирования, которое может быть еще одним инкрементным, дифференциальным или полным бэкапом.

Check

Действие **Check** проверяет соответствие настроек хостов, компонентов и сопутствующих сервисов требованиям сервиса, для которого оно вызвано, а также правильность работы самого сервиса.

Результат действия и процесс его выполнения отображаются на странице Jobs.



Результат действия 'Check' на странице 'Jobs'

Resume Cluster

Действие **Resume Cluster** выключает режим паузы, возвращает контроль Patroni и приводит кластер в рабочее состояние. Действие также разрешает ситуацию с несколькими лидерами, если она возникла.

Для включения режима паузы используйте действие Pause Cluster.

Pause Cluster

Действие **Pause Cluster** переключает Patroni в режим паузы. В режиме паузы Patroni не управляет кластером, но сохраняет состояние кластера в DCS. Режим паузы может быть полезен во время мажорного обновления версии или восстановления после сбоя. Эти операции часто запускают и останавливают ноды по причинам, независимым Patroni. Некоторые ноды могут быть даже временно перезапущены в режиме мастера, нарушая предположение о запуске только одного лидера. По этим причинам Patroni должен иметь возможность перестать контролировать работающий кластер.

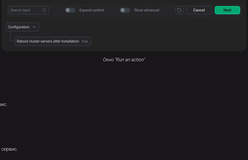
Для выключения режима паузы используйте действие Resume Cluster.

Expand

Действие **Expand** добавляет новые хосты сервису.

> **ВАЖНО**
>
> Если вы используете сервис Monitoring, выполните действие **Reconfigure and restart** сервиса Monitoring после завершения действия **Expand**, чтобы метрики начали собираться с добавленных хостов.
>
> Если вы используете ADPG Control для кластера ADPG, необходимо выполнить действия для серверов в следующем порядке:
>
> 1. действие **Add/Remove components** сервиса Monitoring в кластере ADPG;
> 2. действие **Add components** сервиса ADPG Control Agents в кластере ADPG;
> 3. действие **Expand** сервиса ADPG в кластере ADPG;
> 4. действие Apply database config менеджера Кластеров в кластере ADPG Control;
> 5. действие **Reconfigure and restart** сервиса Monitoring в кластере ADPG.

Когда вы выберете **Expand**, ADCM отображает диалоговое окно для указания параметров действия.



Окно 'Run an action'

В диалоговом окне **Run an action** можно выбрать, будет ли ADCM перезагружать серверы кластера после установки. Чтобы открыть вкладку **Host — Component**, где можно добавить хосты кластера компоненту, нажмите **Next**.

Вкладка 'Host — Component'

Компоненты сервиса, для которого выполняется действие, выделены зеленым. Кликните **Add hosts**, а затем выберите хосты для компонента из появившегося списка. Список отображает хосты, добавленные в кластер. Новые хосты в кластер добавляются на странице **Hosts**.

Добавление хоста компоненту

После того как хосты добавлены, кликните **Next**.

Reconfigure & Restart

Действие **Reconfigure & Restart** обновляет конфигурационные файлы сервиса в соответствии с параметрами конфигурации, указанными на вкладке **Primary configuration**. Если изменяемый параметр требует перезапуска сервиса, действие перезапустит сервис. В противном случае действие **Reconfigure & Restart** только перезагрузит конфигурацию.

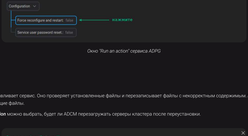
Если изменения вносились непосредственно в файлы postgresql.conf или pg_hba.conf, эти изменения будут потеряны.

В окне подтверждения действия вы можете включить опцию **Service user password reset**. Эта опция позволяет перегенерировать пароли системных пользователей Patroni `adpg_replication` и `adpg_rewind`, когда это необходимо сделать по соображениям безопасности, например, в случае утечки паролей.

Также можно активировать переключатель **Show advanced**, чтобы отобразить флажок **Force reconfigure and restart**, который отключает валидацию параметров конфигурации перед перезапуском ADPG.

Параметр **Force reconfigure and restart** следует использовать в следующих случаях:

- Если сервис ADPG остановлен и не может быть запущен из-за неправильной конфигурации. В этом случае значения параметров конфигурации будут заменены значениями, используемыми по умолчанию.
- Вы изменили параметры раздела **Patroni ADPG configurations**, требующие перезапуска сервиса Patroni, например, `patroni_log_dir`.



Окно 'Run an action' сервиса ADPG

Reinstall

Действие **Reinstall** переустанавливает сервис. Оно проверяет установленные файлы и переустанавливает файлы с некорректным содержимым. **Reinstall** также восстанавливает отсутствующие файлы.

В диалоговом окне **Run an action** можно выбрать, будет ли ADCM перезагружать серверы кластера после переустановки.

Окно 'Run an action'

Start

Действие **Start** запускает сервис.

Stop

Действие **Stop** останавливает сервис.

Действия сервиса ADPG Control Agents

Евгения Кузина

Содержание

Назад к содержанию

Add components

Check

Reinstall

Restart

Start

Stop

Uninstall

В этой статье описаны действия, доступные для сервиса ADPG Control Agents.

Add components

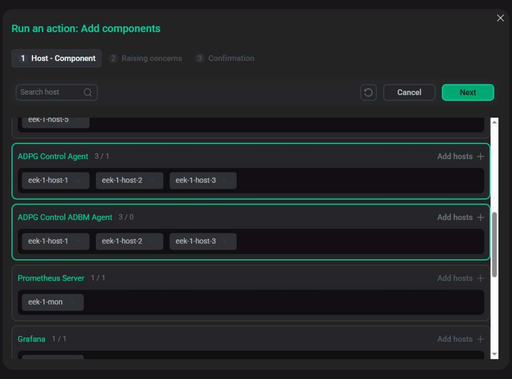
Действие ***Add components*** устанавливает компоненты сервиса ADPG Control Agents на новые хосты.

ВАЖНО

Если вы используете ADPG Control для кластера ADPG, необходимо выполнить действия для сервисов в следующем порядке:

1. действие ***Add/Remove components*** сервиса Monitoring в кластере ADPG;
2. действие ***Add components*** сервиса ADPG Control Agents в кластере ADPG;
3. действие ***Expand*** сервиса ADPG в кластере ADPG;
4. действие Apply backup config менеджера бэкапов в кластере ADPG Control;
5. действие ***Reconfigure and restart*** сервиса Monitoring в кластере ADPG.

При выборе этого действия ADCM открывает окно ***Run an action***, позволяющее добавить новые хосты, на которые необходимо установить компоненты ADPG Control Agents.



Окно действия "Add components"

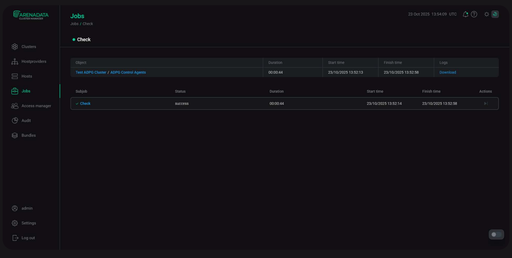
Нажмите ***Add hosts***, чтобы выбрать хост для установки соответствующего компонента. Обратите внимание, что хост должен принадлежать текущему кластеру. Новые хосты в кластер добавляются на странице Hosts.

После добавления компонентов нажмите ***Next*** в верхней части окна.

Check

Действие ***Check*** проверяет соответствие настроек хостов, компонентов и сопутствующих сервисов требованиям сервиса, для которого оно вызвано, а также правильность работы самого сервиса.

Результат действия и процесс его выполнения отображаются на странице Jobs.



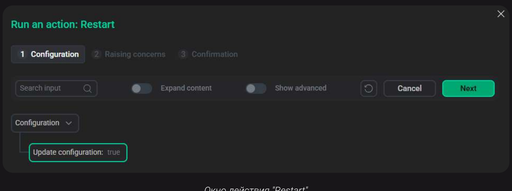
Результат действия "Check" на странице "Jobs"

Reinstall

Действие ***Reinstall*** переустанавливает сервис. Оно проверяет установленные файлы и перезаписывает файлы с некорректным содержимым. ***Reinstall*** также восстанавливает отсутствующие файлы.

Restart

Действие ***Restart*** перезапускает сервис. Если в окне подтверждения полю ***Update configuration*** присвоено значение `true`, конфигурационные параметры сервиса будут обновлены в соответствии со значениями, указанными в интерфейсе ADCM.



Окно действия "Restart"

Если для ***Update configuration*** установлено значение `false`, произойдет только перезапуск сервиса с параметрами, установленными ранее.

Start

Действие ***Start*** запускает сервис.

Stop

Действие ***Stop*** останавливает сервис.

Uninstall

Действие ***Uninstall*** удаляет сервис.

Действия сервиса Balancer

Евгения Кузина

Содержание

Назад к содержанию

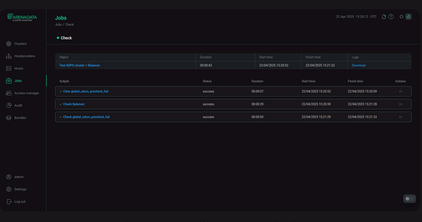
Check
Expand
Reconfigure & Restart
Reinstall
Start
Stop
Uninstall

В этой статье описаны действия, доступные для сервиса Balancer.

Check

Действие ***Check*** проверяет соответствие настроек хостов, компонентов и сопутствующих сервисов требованиям сервиса, для которого оно вызвано, а также правильность работы самого сервиса.

Результат действия и процесс его выполнения отображаются на странице Jobs.



Результат действия "Check" на странице "Jobs"

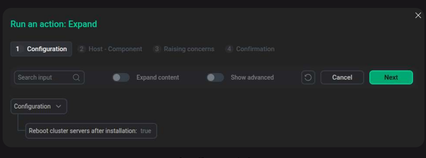
Expand

Действие ***Expand*** добавляет новые хосты сервису.

ПРИМЕЧАНИЕ

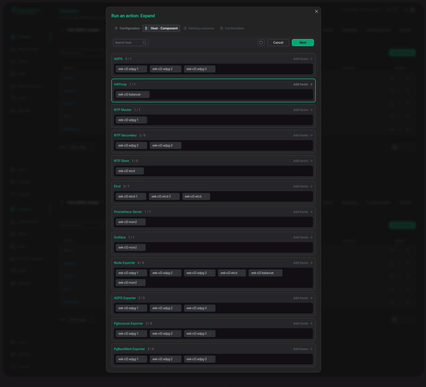
Если вы используете сервис Monitoring, вызовите действие ***Reconfigure and restart*** сервиса Monitoring после завершения действия ***Expand***, чтобы метрики начали собираться с добавленных хостов.

Когда вы выбираете ***Expand***, ADCM отображает диалоговое окно для указания параметров действия.



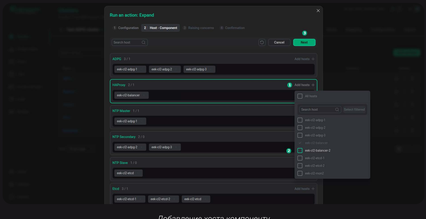
Окно "Run an action"

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после установки. Чтобы открыть вкладку ***Host — Component***, где можно добавить хосты кластера компоненту, нажмите ***Next***.



Вкладка "Host — Component"

Компоненты сервиса, для которого выполняется действие, выделены зеленым. Кликните ***Add hosts***, а затем выберите хосты для компонента из появившегося списка. Список отображает хосты, добавленные в кластер. Новые хосты в кластер добавляются на странице ***Hosts***.



Добавление хоста компоненту

После того как хосты добавлены, кликните ***Next***.

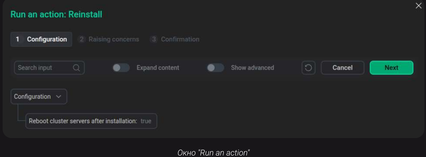
Reconfigure & Restart

Действие ***Reconfigure & Restart*** обновляет конфигурационные файлы сервиса в соответствии с параметрами конфигурации, указанными на вкладке ***Primary configuration***. Если измененный параметр требует перезапуска сервиса, действие перезапускает сервис. В противном случае действие ***Reconfigure & Restart*** только перезагружает конфигурацию.

Reinstall

Действие ***Reinstall*** переустанавливает сервис. Оно проверяет установленные файлы и перезаписывает файлы с некорректным содержимым. ***Reinstall*** также восстанавливает отсутствующие файлы.

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после переустановки.



Окно "Run an action"

Start

Действие ***Start*** запускает сервис.

Stop

Действие ***Stop*** останавливает сервис.

Uninstall

Действие ***Uninstall*** удаляет сервис.

Действия сервиса Chrony

Евгения Кузина

Содержание

Назад к содержанию

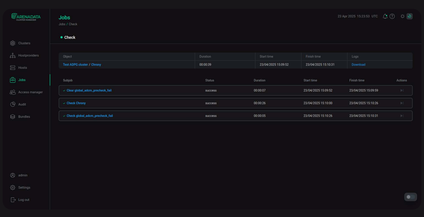
Check
Expand
Reconfigure & Restart
Reinstall
Start
Stop
Uninstall

В этой статье описаны действия, доступные для сервиса Chrony.

Check

Действие ***Check*** проверяет соответствие настроек хостов, компонентов и сопутствующих сервисов требованиям сервиса, для которого оно вызвано, а также правильность работы самого сервиса.

Результат действия и процесс его выполнения отображаются на странице Jobs.



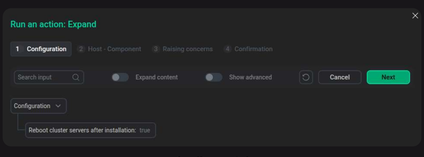
Результат действия "Check" на странице "Jobs"

Expand

Действие ***Expand*** добавляет новые хосты сервису.

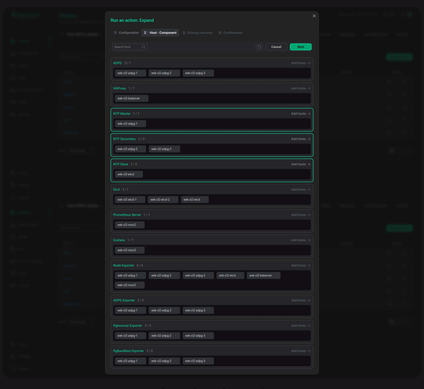
> **ПРИМЕЧАНИЕ**
>
> Если вы используете сервис Monitoring, вызовите действие ***Reconfigure and restart*** сервиса Monitoring после завершения действия ***Expand***, чтобы метрики начали собираться с добавленных хостов.

Когда вы выбираете ***Expand***, ADCM отображает диалоговое окно для указания параметров действия.



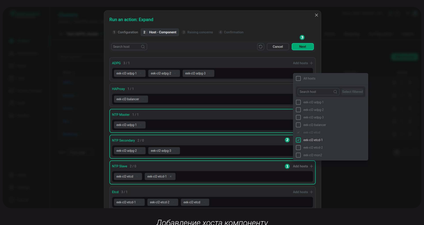
Окно "Run an action"

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после установки. Чтобы открыть вкладку ***Host — Component***, где можно добавить хосты кластера компоненту, нажмите ***Next***.



Вкладка "Host — Component"

Компоненты сервиса, для которого выполняется действие, выделены зеленым. Кликните ***Add hosts***, а затем выберите хосты для компонента из появившегося списка. Список отображает хосты, добавленные в кластер. Новые хосты в кластер добавляются на странице ***Hosts***.



Добавление хоста компоненту

После того как хосты добавлены, кликните ***Next***.

Reconfigure & Restart

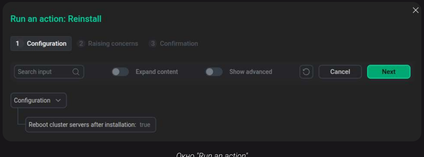
Действие ***Reconfigure & Restart*** обновляет конфигурационные файлы сервиса в соответствии с параметрами конфигурации, указанными на вкладке ***Primary configuration***. Если измененный параметр требует перезапуска сервиса, действие перезапускает сервис. В противном случае действие ***Reconfigure & Restart*** только перезагружает конфигурацию.

Если изменения вносились непосредственно в файл *chrony.conf*, эти изменения будут потеряны.

Reinstall

Действие ***Reinstall*** переустанавливает сервис. Оно проверяет установленные файлы и перезаписывает файлы с некорректным содержимым. ***Reinstall*** также восстанавливает отсутствующие файлы.

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после переустановки.



Окно "Run an action"

Start

Действие ***Start*** запускает сервис.

Stop

Действие ***Stop*** останавливает сервис.

Uninstall

Действие ***Uninstall*** удаляет сервис.

Действия сервиса Etcd

Евгения Кузина

Содержание

Назад к содержанию

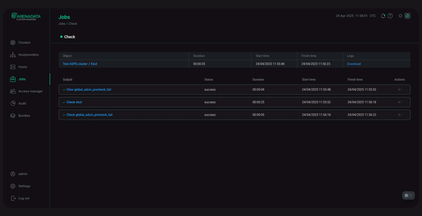
Check
Expand
Reconfigure & Restart
Reinstall
Start
Stop
Uninstall

В этой статье описаны действия, доступные для сервиса Etcd.

Check

Действие ***Check*** проверяет соответствие настроек хостов, компонентов и сопутствующих сервисов требованиям сервиса, для которого оно вызвано, а также правильность работы самого сервиса.

Результат действия и процесс его выполнения отображаются на странице Jobs.



Результат действия "Check" на странице "Jobs"

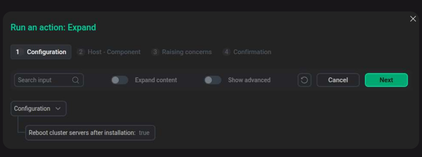
Expand

Действие ***Expand*** добавляет новые хосты сервису. Количество хостов сервиса Etcd должно оставаться нечетным.

ПРИМЕЧАНИЕ

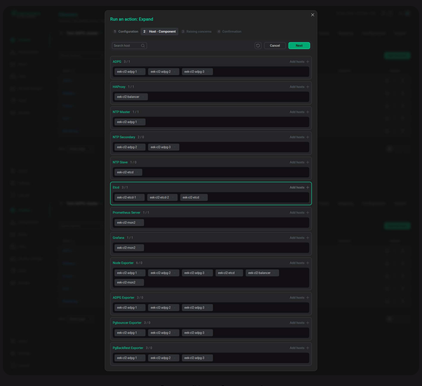
- После добавления нод для расширения сервиса Etcd необходимо выполнить действие ***Reconfigure & Restart*** сервисов ADPG и Balancer.
- Если вы используете сервис Monitoring, вызовите действие ***Reconfigure and restart*** сервиса Monitoring после завершения действия ***Expand***, чтобы метрики начали собираться с добавленных хостов.

Когда вы выбираете ***Expand***, ADCM отображает диалоговое окно для указания параметров действия.



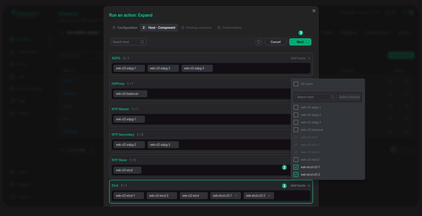
Окно "Run an action"

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после установки. Чтобы открыть вкладку ***Host ↔ Component***, где можно добавить хосты кластера компоненту, нажмите ***Next***.



Вкладка "Host ↔ Component"

Компоненты сервиса, для которого выполняется действие, выделены зеленым. Кликните ***Add hosts***, а затем выберите хосты для компонента из появившегося списка. Список отображает хосты, добавленные в кластер. Новые хосты в кластер добавляются на странице ***Hosts***.



Добавление хоста компоненту

После того как хосты добавлены, кликните ***Next***.

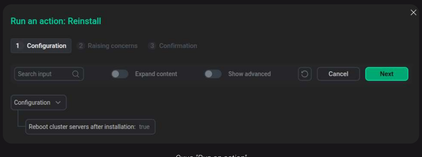
Reconfigure & Restart

Действие ***Reconfigure & Restart*** обновляет конфигурационные файлы сервиса в соответствии с параметрами конфигурации, указанными на вкладке ***Primary configuration***. Если измененный параметр требует перезапуска сервиса, действие перезапускает сервис. В противном случае действие ***Reconfigure & Restart*** только перезагружает конфигурацию.

Reinstall

Действие ***Reinstall*** переустанавливает сервис. Оно проверяет установленные файлы и перезаписывает файлы с некорректным содержимым. ***Reinstall*** также восстанавливает отсутствующие файлы.

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после переустановки.



Окно "Run an action"

Start

Действие ***Start*** запускает сервис.

Stop

Действие ***Stop*** останавливает сервис.

Uninstall

Действие ***Uninstall*** удаляет сервис.

Действия сервиса Monitoring

Евгения Кузина

Содержание

Назад к содержанию

Add/Remove components

Check

Reconfigure and restart

Reinstall

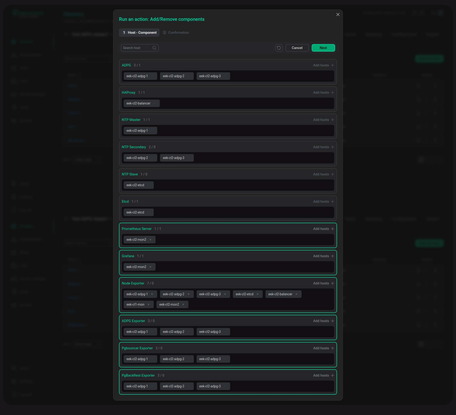
Start

Stop

В этой статье описаны действия, доступные для сервиса Monitoring.

Add/Remove components

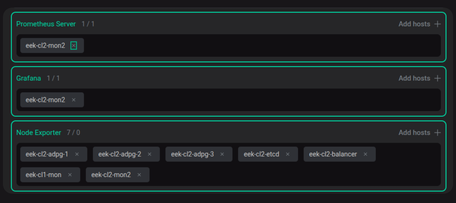
Это действие открывает интерфейс сопоставления компонентов и хостов, в котором можно добавлять и удалять компоненты сервиса Monitoring.



Интерфейс сопоставления компонентов и хостов

Для редактирования доступны только компоненты сервиса Monitoring.

Для компонента вы можете нажать ***Add hosts*** и выбрать хост для добавления из появившегося списка. Также возможно удалить хосты, сопоставленные компоненту, если этот компонент не является обязательным. Если хост можно удалить, то в нем с правой стороны будет отображен серый крестик.



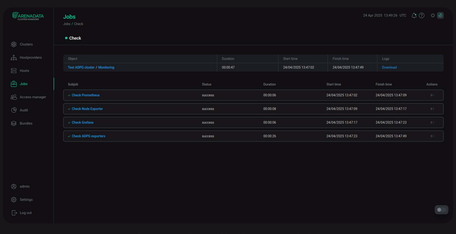
Компоненты, которые можно удалить

После перераспределения компонентов нажмите ***Next*** в верхней части окна.

Check

Действие ***Check*** проверяет соответствие настроек хостов, компонентов и сопутствующих сервисов требованиям сервиса, для которого оно вызвано, а также правильность работы самого сервиса.

Результат действия и процесс его выполнения отображаются на странице Jobs.



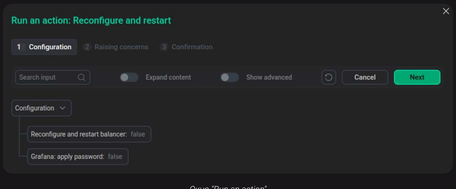
Результат действия "Check" на странице "Jobs"

Reconfigure and restart

Действие ***Reconfigure and restart*** обновляет конфигурационные файлы сервиса в соответствии с параметрами конфигурации, указанными на вкладке ***Primary configuration***. Если измененный параметр требует перезапуска сервиса, действие перезапускает сервис. В противном случае действие ***Reconfigure and restart*** только перезагружает конфигурацию.

При выполнении действия для сервиса Monitoring можно задать следующие параметры:

- ***Reconfigure and restart balancer*** — следует ли перенастроить и перезапустить сервис Balancer.
- ***Grafana: apply password*** — следует ли переустановить пароль администратора Grafana, используя значение из конфигурационного параметра ***Grafana settings → Grafana administrator's password***, расположенного на вкладке ***Primary configuration*** сервиса Monitoring.

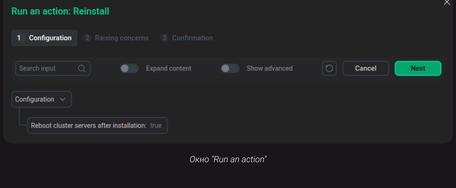


Окно "Run an action"

Reinstall

Действие ***Reinstall*** переустанавливает сервис. Оно проверяет установленные файлы и перезаписывает файлы с некорректным содержимым. ***Reinstall*** также восстанавливает отсутствующие файлы.

В диалоговом окне ***Run an action*** можно выбрать, будет ли ADCM перезагружать серверы кластера после переустановки.



Окно "Run an action"

Start

Действие ***Start*** запускает сервис.

Stop

Действие ***Stop*** останавливает сервис.

Метрики мониторинга ADPG-кластера

Назад к содержанию

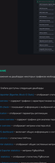
Содержание

- ADPG Checkpointer / Bgwriter / Block IO Stats
- ADPG DB overview
- ADPG Health check
- ADPG Replication
- ADPG Sessions overview
- Global ADPG dashboard
- Global ADPG dashboard
- HAProxy statistics
- Node Exporter statistics
- PgBackRest Exporter statistics
- Pgbouncer statistics
- PostgreSQL Patroni overview

Мониторинг кластера ADPG осуществляется с помощью связки из Grafana + Victoria Metrics + Victoria Logs + Alertmanager. Для сбора метрик используется экспортер postgres_exporter, который собирает данные с каждого узла кластера. Для сбора логов используется vmagent, который собирает логи с каждого узла кластера и отправляет их в Victoria Logs. Для сбора метрик с балансировщиков используется haproxy_exporter. Также с каждого узла кластера собираются метрики node_exporter. Для сбора метрик с PgBouncer используется pgbouncer_exporter. Для сбора метрик с PgBackRest используется pgbackrest_exporter.

Grafana по умолчанию доступна по адресу https://<IP-адрес мастер-узла>:3000. Для доступа к Grafana используется учетная запись admin. Пароль для учетной записи admin задается при установке кластера ADPG с помощью переменной grafana_admin_password. Если переменная grafana_admin_password не задана, то пароль для учетной записи admin генерируется автоматически и сохраняется в файле ~/adpg_admin_password.txt на мастер-узле кластера ADPG.

Стартовая страница Grafana



Авторизация в Grafana

После авторизации в Grafana откроется страница Dashboards.

Стартовая страница Dashboards

Стартовая страница Grafana содержит следующие разделы:

- Home - переход на стартовую страницу Grafana
- Starred - содержит избранные дашборды
- Dashboards - содержит список дашбордов
- Explore - позволяет создавать запросы к источникам данных
- Alerting - содержит информацию о сработавших алертах
- Connections - содержит информацию о подключениях
- Administration - содержит настройки Grafana

Дашборды ADPG

Дашборды в каталоге dashboards/ADPG/ автоматически импортируются в Grafana при установке кластера ADPG. Для этого используется переменная grafana_dashboards_import. Если переменная grafana_dashboards_import имеет значение true, то дашборды будут импортированы в Grafana. Если переменная grafana_dashboards_import имеет значение false, то дашборды не будут импортированы в Grafana.

Для работы дашбордов используется переменная datasource. Она должна иметь значение prometheus. Если переменная datasource имеет другое значение, то дашборды могут работать некорректно.

Дашборды содержат переменную instance, которая позволяет выбрать один или несколько узлов кластера ADPG. Если переменная instance имеет значение All, то будут выбраны все узлы кластера ADPG. Если переменная instance имеет значение конкретного узла, то будет выбран только этот узел.

Дашборды содержат переменную database, которая позволяет выбрать одну или несколько баз данных. Если переменная database имеет значение All, то будут выбраны все базы данных. Если переменная database имеет значение конкретной базы данных, то будет выбрана только эта база данных.

Дашборды содержат переменную timezone, которая позволяет выбрать часовой пояс. Если переменная timezone имеет значение browser, то будет использован часовой пояс браузера. Если переменная timezone имеет значение UTC, то будет использован часовой пояс UTC.

Дашборды содержат переменную interval, которая позволяет выбрать интервал времени. Если переменная interval имеет значение auto, то будет использован интервал времени, который зависит от выбранного периода времени.

ADPG Checkpointer / Bgwriter / Block IO Stats

Дашборд ADPG Checkpointer / Bgwriter / Block IO Stats

Этот дашборд содержит следующие графики:

- Checkpoints - количество контрольных точек в секунду (per second)
- Checkpoint Write Time - время записи контрольных точек в секунду (per second)
- Buffers Rate - количество буферов, записанных BGWRITER, BACKEND, FSYNC, ALLOC в секунду (per second)
- Buffer Usage - количество буферов, записанных BGWRITER, BACKEND, FSYNC, ALLOC в процентном соотношении к общему числу буферов (percentage)
- Table / Index - Read Blocks Read - общее количество блоков, прочитанных из таблиц и индексов (per second)
- Table / Index - Read Blocks Hit - общее количество блоков, прочитанных из таблиц и индексов, которые были найдены в буферном кэше (per second)

ADPG DB overview

Дашборд ADPG DB overview

Этот дашборд содержит следующие графики:

- Instance stats - Transaction rate (per second)
- Instance uptime
- TPS - количество транзакций в секунду
- APPS - количество запросов в секунду
- Query duration - среднее время выполнения запроса
- DB size (B / KB) - размер БД, указанной в переменной database
- Database / Tablespace size - размер баз данных и табличных пространств
- Table / Index - Total Blocks Read - общее количество блоков, прочитанных из таблиц и индексов (per second)
- Table / Index - Total Blocks Hit - общее количество блоков, прочитанных из таблиц и индексов, которые были найдены в буферном кэше (per second)
- Table / Index - Read Blocks Read - общее количество блоков, прочитанных из таблиц и индексов (per second)
- Table / Index - Read Blocks Hit - общее количество блоков, прочитанных из таблиц и индексов, которые были найдены в буферном кэше (per second)
- Buffers Rate - количество буферов, записанных BGWRITER, BACKEND, FSYNC, ALLOC в секунду (per second)
- Buffer Usage - количество буферов, записанных BGWRITER, BACKEND, FSYNC, ALLOC в процентном соотношении к общему числу буферов (percentage)
- Table size - Total Blocks Read - общее количество блоков, прочитанных из таблиц (per second)
- Table size - Total Blocks Hit - общее количество блоков, прочитанных из таблиц, которые были найдены в буферном кэше (per second)
- Table size - Read Blocks Read - общее количество блоков, прочитанных из таблиц (per second)
- Table size - Read Blocks Hit - общее количество блоков, прочитанных из таблиц, которые были найдены в буферном кэше (per second)
- Temp size - количество временных файлов, созданных для выполнения запросов (per second)
- Temp files - количество временных файлов, созданных для выполнения запросов (per second)
- Deadlocks - количество взаимных блокировок (per second)

ADPG Health check

Дашборд ADPG Health check

Этот дашборд содержит следующие графики:

- Instance stats - Transaction rate (per second)
- Instance uptime
- TPS - количество транзакций в секунду
- Longest query runtime
- active connections
- Max_connections
- Minimum connection
- Shared buffer hit rate
- Total size of all WALs
- XTRAbkp
- Archive Status
- Archive Ready Count
- DB size(bytes)
- DB size (Kilobytes/MB)
- Temp written(bytes)
- Max opened temp files
- DML / DDL size
- Invalid Replication Indexes
- Autovacuum Count
- Autovacuum Analyze Count
- Vacuum / Job Count
- Wait_event_type(Count)
- Temp file size(Kilobytes)
- Longest IDLE/IDLE(transaction)
- Max_connection / (Total active+ idle)
- ADMIN's open connection(s)
- ADMIN's open connection(s)
- Wait(TCP) over total time(%)
- Wait_event_type(Count)
- Max_connection / (Total active+ idle)
- Max_ DDRM Session age
- Inactive replication slots

ADPG Replication

Дашборд ADPG Replication

Этот дашборд содержит следующие графики:

- Replication_slots_slots
- replication_slots
- active replica
- replica_lag(in bytes)
- replica_lag(in seconds)
- Replication Status - Статус репликации (streaming, catchup, backup, disconnected)
- Replication Lag (in bytes) - Отставание реплики в байтах
- Replication Lag (in seconds) - Отставание реплики в секундах
- Replication Slots Usage - Использование слотов репликации
- Slot Type - Тип слота репликации (physical, logical)
- Slot Active - Активность слота репликации (true, false)
- Slot Retained WAL - Количество WAL-файлов, удерживаемых слотом репликации
- Restart LSN - Позиция LSN, с которой слот может возобновить репликацию
- Confirmed Flush LSN - Позиция LSN, подтвержденная подписчиком (только для logical slots)
- WAL Status - Состояние WAL-файлов, связанных со слотом (reserved, extended, unreserved, lost)
- Safe WAL Size - Объем WAL, который слот может безопасно удерживать (в байтах)
- Two Phase - Поддержка двухфазного коммита (true, false)
- Conflicting - Конфликтует ли слот с текущим состоянием (true, false)

ADPG Sessions overview

Дашборд ADPG Sessions overview

Этот дашборд содержит следующие графики:

- TPS (all)
- Longest query duration
- Longest TX duration
- Longest wait duration
- Longest session duration
- Longest connection duration - # of sessions
- Session Count
- Sessions from extensions

Elovl Cluster overview

Дашборд Elovl Cluster overview

Этот дашборд содержит следующие графики:

- Cluster Nodes Count
- Cluster Nodes Status
- Total Keys
- Total Memory Used
- Total Memory Peak
- Total Memory Lua
- Total Clients
- Total Ops/Sec
- Total Hits/Sec
- Total Misses/Sec
- Total Hits ratio/Sec
- Total Key Expired
- Total Key Evicted
- Total Key Hits / Misses ratio
- Total Key network bytes/sec
- Total Key read connections
- Total Key write connections
- Total Key Hits / Misses ratio per node

Global ADPG dashboard

Дашборд Global ADPG dashboard

Этот дашборд содержит следующие переменные:

- Cluster
- Node
- Interval

Этот дашборд содержит следующие графики:

- Cluster Nodes Count
- Cluster Nodes Status
- Total Memory Allocated
- Total Key Allocated
- Total Key Peak
- Total Key Lua
- Total Key Ops/Sec
- Total Key Hits/Sec
- Total Key Misses/Sec
- Total Key Hits ratio/Sec
- Total Key Expired
- Total Key Evicted
- Total Key Hits / Misses ratio
- Total Key network bytes/sec
- Total Key read connections
- Total Key write connections
- Total Key Hits / Misses ratio per node

HAProxy statistics

Дашборд HAProxy statistics

Дашборд HAProxy statistics содержит следующие графики:

- Current Sessions - количество текущих сессий
- Current Sessions Rate - скорость создания текущих сессий
- Current Requests Rate - скорость текущих запросов
- Current Connections - количество текущих подключений
- Current Connections Rate - скорость создания текущих подключений
- Up Servers - количество работающих серверов
- Current Backend Queue - количество запросов в очереди бэкенда
- Current Server Queue - количество запросов в очереди сервера
- Sessions by Status - количество сессий по статусам
- Sessions by Type - количество сессий по типам
- Sessions Rate by Type - скорость создания сессий по типам
- Bytes In - количество входящих байт
- Bytes Out - количество исходящих байт
- Bytes In Rate - скорость входящих байт
- Bytes Out Rate - скорость исходящих байт
- Denied Requests - количество отклоненных запросов
- Denied Responses - количество отклоненных ответов
- Request Errors - количество ошибок запросов
- Connection Errors - количество ошибок подключений
- Response Errors - количество ошибок ответов
- Retry Warnings - количество предупреждений о повторных попытках
- Redispatch Warnings - количество предупреждений о перенаправлении
- Request Rate - скорость запросов
- Request Rate by Code - скорость запросов по кодам ответа
- Request Rate by Method - скорость запросов по методам
- Request Rate by Path - скорость запросов по путям
- Request Rate by Host - скорость запросов по хостам
- Request Rate by Server - скорость запросов по серверам
- Request Rate by Backend - скорость запросов по бэкендам
- Request Rate by Frontend - скорость запросов по фронтендам
- Request Rate by Listener - скорость запросов по слушателям
- Request Rate by Process - скорость запросов по процессам
- Request Rate by Service - скорость запросов по сервисам
- Request Rate by Cluster - скорость запросов по кластерам
- Request Rate by Instance - скорость запросов по инстансам
- Request Rate by Job - скорость запросов по заданиям
- Request Rate by Environment - скорость запросов по окружениям
- Request Rate by Region - скорость запросов по регионам
- Request Rate by Zone - скорость запросов по зонам
- Request Rate by Availability Zone - скорость запросов по зонам доступности
- Request Rate by Instance Type - скорость запросов по типам инстансов
- Request Rate by Image - скорость запросов по образам
- Request Rate by Tag - скорость запросов по тегам
- Request Rate by Label - скорость запросов по меткам
- Request Rate by Annotation - скорость запросов по аннотациям
- Request Rate by Environment Variable - скорость запросов по переменным окружения
- Request Rate by Secret - скорость запросов по секретам
- Request Rate by Config Map - скорость запросов по картам конфигурации
- Request Rate by Persistent Volume Claim - скорость запросов по запросам на постоянные тома
- Request Rate by Persistent Volume - скорость запросов по постоянным томам
- Request Rate by Storage Class - скорость запросов по классам хранения
- Request Rate by Network Policy - скорость запросов по политикам сети
- Request Rate by Ingress - скорость запросов по входящим
- Request Rate by Egress - скорость запросов по исходящим
- Request Rate by Service Account - скорость запросов по учетным записям служб
- Request Rate by Role - скорость запросов по ролям
- Request Rate by Cluster Role - скорость запросов по ролям кластера
- Request Rate by Role Binding - скорость запросов по привязкам ролей
- Request Rate by Cluster Role Binding - скорость запросов по привязкам ролей кластера
- Request Rate by Pod Security Policy - скорость запросов по политикам безопасности подов
- Request Rate by Admission Webhook - скорость запросов по вебхукам допуска
- Request Rate by Mutating Webhook - скорость запросов по изменяющим вебхукам
- Request Rate by Validating Webhook - скорость запросов по проверяющим вебхукам
- Request Rate by Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов
- Request Rate by Custom Resource - скорость запросов по пользовательским ресурсам
- Request Rate by Operator - скорость запросов по операторам
- Request Rate by Operator Version - скорость запросов по версиям операторов
- Request Rate by Operator Group - скорость запросов по группам операторов
- Request Rate by Operator Namespace - скорость запросов по пространствам имен операторов
- Request Rate by Operator Cluster - скорость запросов по кластерам операторов
- Request Rate by Operator Region - скорость запросов по регионам операторов
- Request Rate by Operator Zone - скорость запросов по зонам операторов
- Request Rate by Operator Availability Zone - скорость запросов по зонам доступности операторов
- Request Rate by Operator Instance Type - скорость запросов по типам инстансов операторов
- Request Rate by Operator Image - скорость запросов по образам операторов
- Request Rate by Operator Tag - скорость запросов по тегам операторов
- Request Rate by Operator Label - скорость запросов по меткам операторов
- Request Rate by Operator Annotation - скорость запросов по аннотациям операторов
- Request Rate by Operator Environment Variable - скорость запросов по переменным окружения операторов
- Request Rate by Operator Secret - скорость запросов по секретам операторов
- Request Rate by Operator Config Map - скорость запросов по картам конфигурации операторов
- Request Rate by Operator Persistent Volume Claim - скорость запросов по запросам на постоянные тома операторов
- Request Rate by Operator Persistent Volume - скорость запросов по постоянным томам операторов
- Request Rate by Operator Storage Class - скорость запросов по классам хранения операторов
- Request Rate by Operator Network Policy - скорость запросов по политикам сети операторов
- Request Rate by Operator Ingress - скорость запросов по входящим операторов
- Request Rate by Operator Egress - скорость запросов по исходящим операторов
- Request Rate by Operator Service Account - скорость запросов по учетным записям служб операторов
- Request Rate by Operator Role - скорость запросов по ролям операторов
- Request Rate by Operator Cluster Role - скорость запросов по ролям кластера операторов
- Request Rate by Operator Role Binding - скорость запросов по привязкам ролей операторов
- Request Rate by Operator Cluster Role Binding - скорость запросов по привязкам ролей кластера операторов
- Request Rate by Operator Pod Security Policy - скорость запросов по политикам безопасности подов операторов
- Request Rate by Operator Admission Webhook - скорость запросов по вебхукам допуска операторов
- Request Rate by Operator Mutating Webhook - скорость запросов по изменяющим вебхукам операторов
- Request Rate by Operator Validating Webhook - скорость запросов по проверяющим вебхукам операторов
- Request Rate by Operator Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов операторов
- Request Rate by Operator Custom Resource - скорость запросов по пользовательским ресурсам операторов
- Request Rate by Operator Operator - скорость запросов по операторам операторов
- Request Rate by Operator Operator Version - скорость запросов по версиям операторов операторов
- Request Rate by Operator Operator Group - скорость запросов по группам операторов операторов
- Request Rate by Operator Operator Namespace - скорость запросов по пространствам имен операторов операторов
- Request Rate by Operator Operator Cluster - скорость запросов по кластерам операторов операторов
- Request Rate by Operator Operator Region - скорость запросов по регионам операторов операторов
- Request Rate by Operator Operator Zone - скорость запросов по зонам операторов операторов
- Request Rate by Operator Operator Availability Zone - скорость запросов по зонам доступности операторов операторов
- Request Rate by Operator Operator Instance Type - скорость запросов по типам инстансов операторов операторов
- Request Rate by Operator Operator Image - скорость запросов по образам операторов операторов
- Request Rate by Operator Operator Tag - скорость запросов по тегам операторов операторов
- Request Rate by Operator Operator Label - скорость запросов по меткам операторов операторов
- Request Rate by Operator Operator Annotation - скорость запросов по аннотациям операторов операторов
- Request Rate by Operator Operator Environment Variable - скорость запросов по переменным окружения операторов операторов
- Request Rate by Operator Operator Secret - скорость запросов по секретам операторов операторов
- Request Rate by Operator Operator Config Map - скорость запросов по картам конфигурации операторов операторов
- Request Rate by Operator Operator Persistent Volume Claim - скорость запросов по запросам на постоянные тома операторов операторов
- Request Rate by Operator Operator Persistent Volume - скорость запросов по постоянным томам операторов операторов
- Request Rate by Operator Operator Storage Class - скорость запросов по классам хранения операторов операторов
- Request Rate by Operator Operator Network Policy - скорость запросов по политикам сети операторов операторов
- Request Rate by Operator Operator Ingress - скорость запросов по входящим операторов операторов
- Request Rate by Operator Operator Egress - скорость запросов по исходящим операторов операторов
- Request Rate by Operator Operator Service Account - скорость запросов по учетным записям служб операторов операторов
- Request Rate by Operator Operator Role - скорость запросов по ролям операторов операторов
- Request Rate by Operator Operator Cluster Role - скорость запросов по ролям кластера операторов операторов
- Request Rate by Operator Operator Role Binding - скорость запросов по привязкам ролей операторов операторов
- Request Rate by Operator Operator Cluster Role Binding - скорость запросов по привязкам ролей кластера операторов операторов
- Request Rate by Operator Operator Pod Security Policy - скорость запросов по политикам безопасности подов операторов операторов
- Request Rate by Operator Operator Admission Webhook - скорость запросов по вебхукам допуска операторов операторов
- Request Rate by Operator Operator Mutating Webhook - скорость запросов по изменяющим вебхукам операторов операторов
- Request Rate by Operator Operator Validating Webhook - скорость запросов по проверяющим вебхукам операторов операторов
- Request Rate by Operator Operator Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов операторов операторов
- Request Rate by Operator Operator Custom Resource - скорость запросов по пользовательским ресурсам операторов операторов
- Request Rate by Operator Operator Operator - скорость запросов по операторам операторов операторов
- Request Rate by Operator Operator Operator Version - скорость запросов по версиям операторов операторов операторов
- Request Rate by Operator Operator Operator Group - скорость запросов по группам операторов операторов операторов
- Request Rate by Operator Operator Operator Namespace - скорость запросов по пространствам имен операторов операторов операторов
- Request Rate by Operator Operator Operator Cluster - скорость запросов по кластерам операторов операторов операторов
- Request Rate by Operator Operator Operator Region - скорость запросов по регионам операторов операторов операторов
- Request Rate by Operator Operator Operator Zone - скорость запросов по зонам операторов операторов операторов
- Request Rate by Operator Operator Operator Availability Zone - скорость запросов по зонам доступности операторов операторов операторов
- Request Rate by Operator Operator Operator Instance Type - скорость запросов по типам инстансов операторов операторов операторов
- Request Rate by Operator Operator Operator Image - скорость запросов по образам операторов операторов операторов
- Request Rate by Operator Operator Operator Tag - скорость запросов по тегам операторов операторов операторов
- Request Rate by Operator Operator Operator Label - скорость запросов по меткам операторов операторов операторов
- Request Rate by Operator Operator Operator Annotation - скорость запросов по аннотациям операторов операторов операторов
- Request Rate by Operator Operator Operator Environment Variable - скорость запросов по переменным окружения операторов операторов операторов
- Request Rate by Operator Operator Operator Secret - скорость запросов по секретам операторов операторов операторов
- Request Rate by Operator Operator Operator Config Map - скорость запросов по картам конфигурации операторов операторов операторов
- Request Rate by Operator Operator Operator Persistent Volume Claim - скорость запросов по запросам на постоянные тома операторов операторов операторов
- Request Rate by Operator Operator Operator Persistent Volume - скорость запросов по постоянным томам операторов операторов операторов
- Request Rate by Operator Operator Operator Storage Class - скорость запросов по классам хранения операторов операторов операторов
- Request Rate by Operator Operator Operator Network Policy - скорость запросов по политикам сети операторов операторов операторов
- Request Rate by Operator Operator Operator Ingress - скорость запросов по входящим операторов операторов операторов
- Request Rate by Operator Operator Operator Egress - скорость запросов по исходящим операторов операторов операторов
- Request Rate by Operator Operator Operator Service Account - скорость запросов по учетным записям служб операторов операторов операторов
- Request Rate by Operator Operator Operator Role - скорость запросов по ролям операторов операторов операторов
- Request Rate by Operator Operator Operator Cluster Role - скорость запросов по ролям кластера операторов операторов операторов
- Request Rate by Operator Operator Operator Role Binding - скорость запросов по привязкам ролей операторов операторов операторов
- Request Rate by Operator Operator Operator Cluster Role Binding - скорость запросов по привязкам ролей кластера операторов операторов операторов
- Request Rate by Operator Operator Operator Pod Security Policy - скорость запросов по политикам безопасности подов операторов операторов операторов
- Request Rate by Operator Operator Operator Admission Webhook - скорость запросов по вебхукам допуска операторов операторов операторов
- Request Rate by Operator Operator Operator Mutating Webhook - скорость запросов по изменяющим вебхукам операторов операторов операторов
- Request Rate by Operator Operator Operator Validating Webhook - скорость запросов по проверяющим вебхукам операторов операторов операторов
- Request Rate by Operator Operator Operator Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов операторов операторов операторов
- Request Rate by Operator Operator Operator Custom Resource - скорость запросов по пользовательским ресурсам операторов операторов операторов
- Request Rate by Operator Operator Operator Operator - скорость запросов по операторам операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Version - скорость запросов по версиям операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Group - скорость запросов по группам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Namespace - скорость запросов по пространствам имен операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Cluster - скорость запросов по кластерам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Region - скорость запросов по регионам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Zone - скорость запросов по зонам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Availability Zone - скорость запросов по зонам доступности операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Instance Type - скорость запросов по типам инстансов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Image - скорость запросов по образам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Tag - скорость запросов по тегам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Label - скорость запросов по меткам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Annotation - скорость запросов по аннотациям операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Environment Variable - скорость запросов по переменным окружения операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Secret - скорость запросов по секретам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Config Map - скорость запросов по картам конфигурации операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Persistent Volume Claim - скорость запросов по запросам на постоянные тома операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Persistent Volume - скорость запросов по постоянным томам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Storage Class - скорость запросов по классам хранения операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Network Policy - скорость запросов по политикам сети операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Ingress - скорость запросов по входящим операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Egress - скорость запросов по исходящим операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Service Account - скорость запросов по учетным записям служб операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Role - скорость запросов по ролям операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Cluster Role - скорость запросов по ролям кластера операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Role Binding - скорость запросов по привязкам ролей операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Cluster Role Binding - скорость запросов по привязкам ролей кластера операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Pod Security Policy - скорость запросов по политикам безопасности подов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Admission Webhook - скорость запросов по вебхукам допуска операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Mutating Webhook - скорость запросов по изменяющим вебхукам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Validating Webhook - скорость запросов по проверяющим вебхукам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Custom Resource - скорость запросов по пользовательским ресурсам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator - скорость запросов по операторам операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Version - скорость запросов по версиям операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Group - скорость запросов по группам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Namespace - скорость запросов по пространствам имен операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Cluster - скорость запросов по кластерам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Region - скорость запросов по регионам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Zone - скорость запросов по зонам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Availability Zone - скорость запросов по зонам доступности операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Instance Type - скорость запросов по типам инстансов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Image - скорость запросов по образам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Tag - скорость запросов по тегам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Label - скорость запросов по меткам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Annotation - скорость запросов по аннотациям операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Environment Variable - скорость запросов по переменным окружения операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Secret - скорость запросов по секретам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Config Map - скорость запросов по картам конфигурации операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Persistent Volume Claim - скорость запросов по запросам на постоянные тома операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Persistent Volume - скорость запросов по постоянным томам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Storage Class - скорость запросов по классам хранения операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Network Policy - скорость запросов по политикам сети операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Ingress - скорость запросов по входящим операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Egress - скорость запросов по исходящим операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Service Account - скорость запросов по учетным записям служб операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Role - скорость запросов по ролям операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Cluster Role - скорость запросов по ролям кластера операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Role Binding - скорость запросов по привязкам ролей операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Cluster Role Binding - скорость запросов по привязкам ролей кластера операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Pod Security Policy - скорость запросов по политикам безопасности подов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Admission Webhook - скорость запросов по вебхукам допуска операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Mutating Webhook - скорость запросов по изменяющим вебхукам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Validating Webhook - скорость запросов по проверяющим вебхукам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Custom Resource - скорость запросов по пользовательским ресурсам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator - скорость запросов по операторам операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Version - скорость запросов по версиям операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Group - скорость запросов по группам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Namespace - скорость запросов по пространствам имен операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Cluster - скорость запросов по кластерам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Region - скорость запросов по регионам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Zone - скорость запросов по зонам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Availability Zone - скорость запросов по зонам доступности операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Instance Type - скорость запросов по типам инстансов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Image - скорость запросов по образам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Tag - скорость запросов по тегам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Label - скорость запросов по меткам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Annotation - скорость запросов по аннотациям операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Environment Variable - скорость запросов по переменным окружения операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Secret - скорость запросов по секретам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Config Map - скорость запросов по картам конфигурации операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Persistent Volume Claim - скорость запросов по запросам на постоянные тома операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Persistent Volume - скорость запросов по постоянным томам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Storage Class - скорость запросов по классам хранения операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Network Policy - скорость запросов по политикам сети операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Ingress - скорость запросов по входящим операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Egress - скорость запросов по исходящим операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Service Account - скорость запросов по учетным записям служб операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Role - скорость запросов по ролям операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Cluster Role - скорость запросов по ролям кластера операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Role Binding - скорость запросов по привязкам ролей операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Cluster Role Binding - скорость запросов по привязкам ролей кластера операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Pod Security Policy - скорость запросов по политикам безопасности подов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Admission Webhook - скорость запросов по вебхукам допуска операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Mutating Webhook - скорость запросов по изменяющим вебхукам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Validating Webhook - скорость запросов по проверяющим вебхукам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Custom Resource Definition - скорость запросов по определениям пользовательских ресурсов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Custom Resource - скорость запросов по пользовательским ресурсам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator - скорость запросов по операторам операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Version - скорость запросов по версиям операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Group - скорость запросов по группам операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Namespace - скорость запросов по пространствам имен операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Cluster - скорость запросов по кластерам операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Region - скорость запросов по регионам операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Zone - скорость запросов по зонам операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Availability Zone - скорость запросов по зонам доступности операторов операторов операторов операторов операторов операторов операторов
- Request Rate by Operator Operator Operator Operator Operator Operator Operator Instance Type - скорость запросов по типам инстансов операторов операторов операторов операторов операторов операторов операторов

Обзор отказоустойчивости ADPG

Евгения Кузина

Содержание

Назад к содержанию

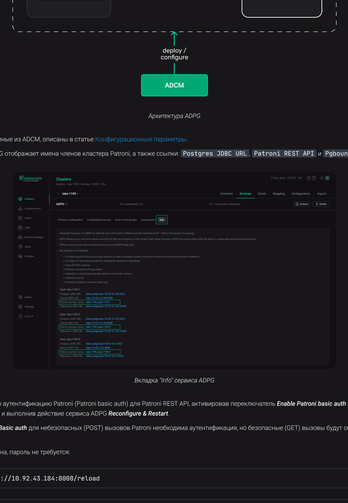
- Режимы репликации
  - Асинхронный режим
  - Синхронный режим
- Кластер etcd

ПРИМЕЧАНИЕ

Функционал, поддерживающий отказоустойчивость (High Availability, HA), реализован в версии ADPG Enterprise.

Серверы базы данных могут работать совместно для обеспечения возможности быстрого переключения на другой сервер в случае отказа основного. Такое поведение называется **отказоустойчивостью (High Availability, HA)**. ADPG использует Patroni для создания кластера на основе потоковой репликации PostgreSQL и реализации отказоустойчивости. Кластер ADPG, подразумевающий HA, должен содержать несколько нод ADPG.

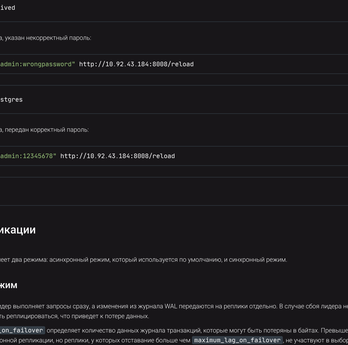
Каждая нода содержит сервер PostgreSQL и агент Patroni, обслуживающий локальный экземпляр PostgreSQL. Одна нода ADPG является **лидером**, остальные — **репликами**. Лидер обслуживает транзакции на чтение и запись (read/write), если иное не указано в настройках сервиса балансировки, а реплики только запросы на чтение (read-only). Потоковая репликация PostgreSQL — это метод, при котором записи WAL передаются от лидера к репликам. В случае отказа лидера Patroni выбирает нового лидера из реплик и кластер ADPG продолжает работу.



Архитектура ADPG

Параметры Patroni, доступные из ADCM, описаны в статье Конфигурационные параметры.

Вкладка **Info** сервиса ADPG отображает имена членов кластера Patroni, а также ссылки `Postgres JDBC URL`, `Patroni REST API` и `Pgbouncer JDBC URL`.



Вкладка 'Info' сервиса ADPG

Можно включить базовую аутентификацию Patroni (Patroni basic auth) для Patroni REST API, активировав переключатель ***Enable Patroni basic auth*** в конфигурационных настройках сервиса ADPG и выполнив действие сервиса ADPG ***Reconfigure & Restart***.

После включения ***Patroni Basic auth*** для небезопасных (POST) вызовов Patroni необходима аутентификация, но безопасные (GET) вызовы будут обрабатываться без нее. Примеры:

Аутентификация выключена, пароль не требуется:

```
$ curl -X POST http://10.92.43.184:8008/reload
```

```
reload scheduled
```

Аутентификация включена, пароль не указан.

```
$ curl -X POST http://10.92.43.184:8008/reload
```

```
no auth header received
```

Аутентификация включена, указан некорректный пароль:

```
$ curl -X POST -u "admin:wrongpassword" http://10.92.43.184:8008/reload
```

```
not authenticatedpostgres
```

Аутентификация включена, передан корректный пароль:

```
$ curl -X POST -u "admin:12345678" http://10.92.43.184:8008/reload
```

```
reload scheduled
```

Режимы репликации

Потоковая репликация имеет два режима: асинхронный режим, который используется по умолчанию, и синхронный режим.

Асинхронный режим

В асинхронном режиме лидер выполняет запросы сразу, а изменения из журнала WAL передаются на реплики отдельно. В случае сбоя лидера незавершенные транзакции могут не успеть реплицироваться, что приведет к потере данных.

Параметр `maximum_lag_on_failover` определяет количество данных журнала транзакций, которые могут быть потеряны в байтах. Превышение этого параметра не влияет на процесс асинхронной репликации, но реплики, у которых отставание больше чем `maximum_lag_on_failover`, не участвуют в выборах лидера. Параметр `maximum_lag_on_failover` можно изменить на вкладке ***Primary configuration*** сервиса ADPG. Чтобы это сделать, откройте ***Clusters → Кластер ADPG → Services → ADPG → Primary configuration***, как описано в статье Настройка сервисов и раскройте ноду ***Patroni ADPG configurations***, содержащую настройки Patroni.

На самом деле количество потерянных данных может быть больше. Для худшего случая его можно рассчитать как сумму `maximum_lag_on_failover` байт журнала транзакций и количества данных, которые были записаны в последние ttl секунд. Однако типичная задержка репликации составляет менее секунды.

Если лидер выходит из строя или становится недоступным по любой другой причине, Patroni автоматически промоутит реплику до лидера. После этой операции невозможно восстановить транзакции, которые не были реплицированы.

Преимущество асинхронного режима репликации — быстрое подтверждение транзакций, не нужно ждать пока все реплики применят изменения.

Синхронный режим

Синхронная репликация гарантирует, что все изменения, сделанные транзакцией, перенесутся на одну или несколько реплик. Каждый коммит транзакции записи ожидает подтверждения того, что коммит записан в WAL на диске ноды лидера и ноды реплики. Синхронная репликация обеспечивает более высокий уровень отказоустойчивости, но также увеличивает время отклика для запрашивающей транзакции. Минимальное время ожидания — это время обмена данными между лидером и репликой.

Следующие транзакции не ждут ответа от реплик:

- транзакции, выполняющие операции только на чтение;
- откаты транзакций (rollbacks);
- коммиты подтранзакций.

Длительные действия, такие как загрузка данных или построение индекса, не ждут окончательного коммит-сообщения.

Чтобы включить синхронный режим Patroni, активируйте флажок `synchronous_mode` на вкладке ***Primary configuration*** сервиса ADPG. Для этого откройте ***Clusters → Кластер ADPG → Services → ADPG → Primary configuration***, как описано в Настройка сервисов.

В этом режиме Patroni может запрокотить до лидера только те реплики, которые содержат все транзакции, вернувшие клиенту статус успешного коммита. Однако синхронный режим Patroni не гарантирует, что коммиты будут храниться на нескольких нодах при любых обстоятельствах. Когда подходящая реплика недоступна, лидер по-прежнему предоставляет доступ для записи, но не гарантирует репликацию. Если лидер выходит из строя, никакая другая реплика не становится лидером. Когда нода, которая раньше была лидером, становится снова доступной для Patroni, он автоматически делает её лидером, если системный администратор не выполнил переключение лидера вручную. Такое поведение позволяет использовать синхронный режим в кластерах, состоящих из двух ADPG-нод.

В случае сбоя реплики коммиты блокируются до тех пор, пока Patroni не запустится и не переключит лидера в автономный режим. В худшем случае задержка составляет `ttl` секунд. Выключение или перезапуск реплики вручную не приводит к нарушению работы сервиса репликации коммитов. Репликa передает сигнал лидеру о необходимости освободить её от синхронизации, до того как PostgreSQL завершит работу.

Чтобы гарантировать, что каждая запись хранится как минимум на двух нодах, включите режим `synchronous_mode_strict` в дополнение к `synchronous_mode`. Вы можете активировать эту опцию на вкладке ***Primary configuration*** сервиса ADPG. Этот параметр запрещает Patroni отключать синхронную репликацию на лидере, когда реплики недоступны. В результате лидер перестает предоставлять доступ на запись и блокирует все клиентские запросы на запись до тех пор, пока хотя бы одна реплика не станет доступной.

ПРИМЕЧАНИЕ

Из-за того, как синхронная репликация реализована в PostgreSQL, по-прежнему возможна потеря транзакций, даже если используется `synchronous_mode_strict`. Если операции на сервере PostgreSQL отмечаются во время ожидания подтверждения репликации (в результате пакетной отмены из-за превышения тайм-аута или сбоя сервера), изменения транзакций становятся видимыми для других серверов. Такие изменения еще не реплицируются и могут быть потеряны в случае смены лидера.

Patroni использует параметр `synchronous_node_count` для управления количеством синхронных резервных баз данных. Его значение по умолчанию равно `1`. Этот параметр не применяется, когда `synchronous_mode` отключен. Если включен режим `synchronous_mode`, Patroni управляет точным количеством синхронных резервных баз данных на основе `synchronous_node_count`. За дополнительной информацией обратитесь к статье Replication modes.

Можно хранить файлы логов Patroni в произвольной директории. Для того, чтобы настроить кастомное логирование, откройте вкладку ***Clusters → Кластер ADPG → Services → ADPG → Primary configuration***, активируйте переключатель ***Show advanced*** и разверните ноду ***Patroni ADPG Configurations***. Установите для параметра ***use_custom_patroni_log_dir*** значение `true` и укажите путь к директории.

Кластер etcd

Patroni сохраняет конфигурацию кластера в распределенном хранилище конфигурации (Distributed Configuration Storage, DCS). ADPG требует наличия кластер etcd, чтобы использовать его в качестве DCS для Patroni. Настройки etcd `listen_peer_urls_port` и `listen_client_urls_port` необходимо указать при установке, после установки их изменить нельзя.

Вы можете использовать встроенный ADPG-сервис Etcd или внешний кластер etcd. Если вы хотите использовать встроенный сервис Etcd, обратитесь к статье Добавление сервисов, в которой описывается, как добавить Etcd-сервис. Для Etcd необходимо нечетное количество нод. Рекомендуется использовать три или более ноды. На нодах с Etcd не должно быть ADPG-сервиса. Обратите внимание, что после добавления нод для расширения уже работающего сервиса Etcd необходимо выполнить действие ***Reconfigure & Restart*** сервисов ADPG и Balancer.

Etcd сохраняет весь лог на диске до тех пор, пока не будет выполнена процедура сжатия, что может привести к нехватке места. Чтобы избежать этой проблемы, в ADPG включено автосжатие. Значения по умолчанию: `Autocompaction mode` является `periodic`, а `Autocompaction retention` — `10h`. Максимальное количество байт, которое может использовать файл базы данных etcd (параметр `Space quota`), установлено в `2147483648` байт. Вы можете изменить эти настройки на странице ***Clusters → Кластер ADPG → Services → Etcd → Primary configuration***.

Для возврата дискового пространства от etcd в файловую систему хоста необходима дефрагментация. Рекомендуется выполнять дефрагментацию по расписанию (например, с помощью cron). Обратите внимание, что при дефрагментации ноды кластера блокируются и становятся недоступными.

Команда дефрагментации всего кластера (блокирует весь кластер на некоторое время):

```
$ etcdctl defrag --cluster
```

Для локальной ноды (на определенный период блокируется только локальная нода):

```
$ etcdctl defrag
```

Если включен SSL:

```
$ /usr/lib/adpg14/usr/bin/etcdctl defrag --cluster --cacert="/etc/adpg/ssl/etcd/ca.crt" \
--cert="/etc/adpg/ssl/etcd/client.crt" --key="/etc/adpg/ssl/etcd/client.key"
```

Если вы хотите использовать внешний кластер etcd, установите настройки внешнего etcd на вкладке ***Primary configuration*** сервиса ADPG. Чтобы указать их, откройте ***Clusters → Кластер ADPG → Services → ADPG → Primary configuration***, как описано в статье Configure services.

РЕКОМЕНДАЦИЯ

Вы можете использовать действие ***Expand*** для добавления новых нод в кластер ADPG и встроенный Etcd-сервис.

Логическая репликация

Евгений Кузина

Содержание

Назад к содержанию

Обзор
Параметры конфигурации
Публикация
Подписка
Конфликты
Безопасность

Обзор

Логическая репликация — это метод репликации объектов данных и их изменений, использующий репликационные идентификаторы (обычно это первичный ключ). Термин "логическая репликация" используется, чтобы отличать такую репликацию от физической, которая построена на адресах блоков и побайтовом копировании. Логическая репликация позволяет контролировать то только репликацию данных, но и аспекты безопасности.

Логическая репликация использует модель **публикаций/подписок (publish/subscribe)** с одним или несколькими **подписчиками (subscribers)**, которые подписываются на одну или несколько **публикаций (publications)** на **ноде публикации (publishing node)**. Подписчики получают данные из публикаций и могут повторно публиковать данные для каскадной репликации или более сложных конфигураций логической репликации.

Логическая репликация таблицы начинается с создания снимка данных на публикующей ноде и копирования его подписчику. Для получения дополнительной информации обратитесь в статье Architecture. После этого изменения на публикующей ноде отправляются подписчику в режиме реального времени по мере их возникновения. Подписчик применяет изменения в том же порядке, что и публикующая нода. Публикации в рамках одной подписки гарантируют целостность транзакций. Подписчик применяет изменения данных в том же порядке, что и нода публикации, чтобы гарантировать согласованность транзакции публикации в рамках одной подписки. Этот метод репликации данных также называется репликацией транзакций.

Типичные сценарии использования логической репликации:

- передача подписчикам инкрементальных изменений в одной базе данных или подмножестве базы данных;
- срабатывание триггеров для отдельных изменений, когда их получает подписчик;
- объединение нескольких баз данных в одну, например, для целей анализа;
- репликация данных между разными мажорными версиями PostgreSQL;
- репликация данных между экземплярами PostgreSQL на разных платформах;
- предоставление доступа к реплицированным данным различным группам пользователей;
- совместное использование данных в нескольких базах данных.

Ed2 поддержит ведет себя так же, как и любой другой экземпляр PostgreSQL, и может быть публикатором, если создать в нем публикации. Когда подписчик действует как приложение с доступом только на чтение, конфликтов не возникает. Если какие-либо приложения или другие подписчики пишут в один и тот же набор таблиц, могут возникнуть конфликты.

Для получения подробной информации об ограничениях логической репликации обратитесь к статье Restrictions.

Чтобы настроить логическую репликацию, выполните следующие действия:

- Сконфигурируйте ноды публикации и подписки.
- Создайте публикацию.
- Создайте подписку.

Параметры конфигурации

Логическая репликация требует установки нескольких параметров конфигурации. Используйте раздел **ADPG configurations** на вкладке **Clusters → Кластер ADPG → Services → ADPG → Primary configuration** пользовательского интерфейса ADCM, чтобы указать эти параметры. Получить дополнительную информацию по настройке параметров можно в статье Конфигурационные параметры.

Установите следующие параметры на ноде публикации.

- Параметр wal_level должен иметь значение `logical`.
- Параметр max_replication_slots должен иметь значение не меньше ожидаемого числа подписчиков плюс некоторый резерв для синхронизации таблиц.
- Параметр max_wal_senders должен быть не меньше чем `max_replication_slots` плюс количество физических реплик, подключаемых одновременно.

На ноде подписчика необходимо указать параметры, перечисленные ниже:

- max_replication_slots должен быть не меньше числа подписок, которые добавлены подписчику, плюс некоторый резерв для синхронизации таблиц.
- max_logical_replication_workers должен быть также не меньше числа подписок, которые добавлены подписчику, плюс некоторый резерв для синхронизации таблиц.
- max_worker_processes должен включать дополнительные рабочие процессы для репликации, как минимум `max_logical_replication_workers` плюс 1. Обратите внимание, что некоторые расширения и параллельные запросы занимают слоты из числа `max_worker_processes`. В отличие от остальных параметров, вы можете установить значение `max_worker_processes` в разделе **ADPG configurations** на вкладке **Primary configuration**. За дополнительной информацией обратитесь к статье Конфигурационные параметры.

Также необходимо проверить, не препятствуют ли политики *pg_hba.conf* на вкладке **Clusters → Кластер ADPG → Services → ADPG → Primary configuration** пользовательского интерфейса ADCM. Эти настройки зависят от конфигурации вашей сети и пользователя, от имени которого будет осуществляться подключение.

Публикация

Публикация может быть определена на любом ведущем физическом сервере репликации. Сервер, содержащий публикацию, называется **публикующим (publisher)**. **Публикация** — это набор изменений, сгенерированных из таблицы или группы таблиц. Она также может называться **набором репликации (replication set)**. Каждая публикация существует только в одной базе данных.

Публикации отличаются от схем и не влияют на доступ к таблице. Каждую таблицу можно включить в несколько публикаций. В настоящее время публикаций могут содержать только таблицы. Объекты должны быть добавлены в них явно, если только публикация не создается для всех таблиц с модификатором `ALL TABLES`.

Публикации могут ограничивать набор содержащихся в них изменений. Можно выбрать любую комбинацию операций `INSERT`, `UPDATE`, `DELETE` и `TRUNCATE`, подобно тому, как триггеры могут срабатывать при различных типах событий. По умолчанию репликируются все типы операций.

Чтобы реплицировать операции `UPDATE` и `DELETE`, опубликованная таблица должна иметь **идентификатор реплики (replica identity)**. Идентификатор реплики позволяет найти соответствующие строки для их обновления или удаления на стороне подписчика. По умолчанию первичный ключ является идентификатором реплики. Можно указать другой уникальный идентификатор для реплики. Если в таблице нет подходящего ключа, в качестве ключа можно использовать всю строку целиком. Для этого укажите значение идентификатора реплики `FULL`. Однако такой подход очень неэффективен и его следует использовать, если нет другого решения. Если на стороне публикующей ноды выбран идентификатор, отличный от `FULL` идентификатор, состоящий из такого же или меньшего количества столбцов, также должен быть определен на стороне подписчика. За дополнительными сведениями о назначении идентификаторов реплики обратитесь к статье REPLICA IDENTITY. Если таблица без идентификатора реплики добавляется к публикации, которая реплицирует операции `UPDATE` или `DELETE`, эти операции вызовут ошибку на ноде публикации. Операции `INSERT` могут быть обработаны без указания идентификатора реплики.

У каждой публикации может быть несколько подписчиков.

Используйте команду CREATE PUBLICATION, чтобы создать публикацию. Следующая команда создает публикацию, которая публикует все изменения в таблицах `books` и `orders`:

```
CREATE PUBLICATION publication1 FOR TABLE books, orders;
```

Код ниже создает публикацию, которая публикует все изменения во всех таблицах:

```
CREATE PUBLICATION all_tables FOR ALL TABLES;
```

Следующая публикация публикует операции `DELETE`, выполненные в таблице `books`:

```
CREATE PUBLICATION delete_publication FOR TABLE books WITH (publish = 'delete');
```

Вы можете использовать команду ALTER PUBLICATION, чтобы изменить определение публикации.

Приведенная ниже команда изменяет публикацию `delete_publication`, чтобы она публиковала не только удаления, но и обновления:

```
ALTER PUBLICATION delete_publication SET (publish = 'update, delete');
```

Код ниже добавляет таблицу `authors` в `publication1`:

```
ALTER PUBLICATION publication1 ADD TABLE authors;
```

Операции `ADD TABLE` и `DROP TABLE` являются транзакционными. Таким образом, репликация таблицы будет начата или закончена только после фиксации транзакции.

Чтобы удалить публикацию, вызовите команду DROP PUBLICATION:

```
DROP PUBLICATION all_tables;
```

Каталог pg_publication содержит все публикации, созданные в базе данных. Представление `pg_publication_tables` сопоставляет публикации и содержащиеся в них таблицы. Для получения данных из `pg_publication` и `pg_publication_tables` используйте следующие команды:

```
SELECT * FROM pg_publication;

SELECT * FROM pg_publication_tables;
```

Подписка

Подписка (subscription) определяется на принимающей стороне логической репликации. Нода, на которой создана подписка, называется **подписчиком**. Подписка определяет подключение к базе данных с публикациями и набор публикаций, на которые подписчик хочет подписаться.

База данных подписчика работает как любой другой экземпляр PostgreSQL и может использоваться в качестве ноды публикации для других баз данных, если определены её собственные публикации.

У подписчика может быть несколько подписок. Можно определить несколько подписок между одной парой нода публикации/подписчик. При этом необходимо следить за тем, чтобы объекты публикации не пересекались.

Каждая подписка получает изменения через один слот репликации. Для начальной синхронизации данных могут потребоваться дополнительные слоты репликации. Они удаляются после завершения синхронизации данных. За дополнительной информацией обратитесь к статье Replication slot management.

Подписка логической репликации может быть резервным сервером для синхронной репликации.

Подписки могут быть заархивированы во время создания дамп-файла базы данных. Укажите `pg_dump`, добавляет подписки в файл SQL dump, если они загружаются от имени суперпользователя. В противном случае подписки пропускаются, так как пользователи без прав суперпользователя не могут прочитать информацию о подписке из каталога pg_subscription.

Используйте команду CREATE SUBSCRIPTION, чтобы добавить подписку:

```
CREATE SUBSCRIPTION subscription1
CONNECTION 'host=10.92.6.36 port=5432 user=postgres dbname=book_store password=postgres'
PUBLICATION publication1, delete_publication;
```

Используйте команду ALTER SUBSCRIPTION, чтобы изменить подписку. Следующий код изменяет `subscription1`, делая её подписанной на публикацию `all_tables`:

```
ALTER SUBSCRIPTION subscription1 SET PUBLICATION all_tables;
```

Команда ниже отключает (останавливает) подписку:

```
ALTER SUBSCRIPTION subscription1 DISABLE;
```

Чтобы удалить подписку, используйте DROP SUBSCRIPTION:

```
DROP SUBSCRIPTION subscription1;
```

При удалении и повторном создании подписки информация о синхронизации теряется. Вам необходимо повторно синхронизировать данные после воссоздания подписки.

Только таблицы могут быть целью репликации. Например, вы не можете реплицировать данные в представление (view).

Таблицы сопоставляются между нодой публикации и подписчиком с использованием полного имени таблицы. Репликация в таблицу с другим именем не поддерживается.

Столбцы таблицы также сопоставляются по имени. Порядок столбцов может не совпадать. Типы данных столбцов также могут не совпадать, если текстовое представление данных может быть преобразовано в целевой тип. Например, вы можете реплицировать столбец типа `integer` в столбец типа `bigint`. В целевой таблице также могут быть дополнительные столбцы, которых нет в опубликованной таблице. Эти столбцы заполняются значениями по умолчанию, указанными в определении целевой таблицы.

Конфликты

Логическая репликация ведет себя аналогично операциям DML: данные обновляются, даже если они были изменены локально на ноде подписчика. Если входящие данные нарушают какие-либо ограничения, репликация останавливается. Это считается **конфликтом**.

Конфликт вызывает ошибку и останавливает репликацию. Необходимо разрешить конфликт вручную. Подробности о конфликте можно найти в журнале сервера подписчика (subscriber server log). Путь по умолчанию — */pg_data1/adp1/log*.

Пример:

```
2023-01-26 17:29:06.262 UTC [16525] LOG:  background worker "logical replication worker" (PID 1389) exited with exit code 1
2023-01-26 17:29:06.274 UTC [1398] ERROR:  duplicate key value violates unique constraint "test1_pkey"
2023-01-26 17:29:06.274 UTC [1398] DETAIL:  Key (employee_id)=(1) already exists.
2023-01-26 17:29:06.274 UTC [1398] CONTEXT:  COPY test1, line 1
```

Разрешение конфликта может заключаться либо в изменении данных на стороне подписчика, чтобы они не конфликтовали с приходящим изменением, либо в пропуске транзакции, конфликтующей с существующими данными. Пропустить транзакцию можно, вызвав функцию pg_replication_origin_advance, которой передается соответствующее имя подписки, а также позиция в качестве параметров. Текущую позицию источников можно найти в системном представлении pg_replication_origin_status.

При репликации операций `UPDATE` или `DELETE` отсутствие данных не вызывает конфликта, так что такие операции просто пропускаются.

Безопасность

Пользователь, имеющий право изменения схемы таблиц на стороне подписчика, может выполнить произвольный код как суперпользователь. Ограничьте круг владельцев и ролей, имеющих право `TRIGGER` для таких таблиц, доверенными ролями. Кроме того, если ненадежные пользователи могут создавать таблицы, используйте публикацию, которая явно перечисляет таблицы. Создавайте подписку `FOR ALL TABLES`, только когда доверяете всем пользователям, имеющим право создавать таблицы на стороне публикации или подписки.

Роль, используемая для подключения репликации, должна иметь атрибут `REPLICATION` или быть суперпользователем. Если у роли нет атрибутов `SUPERUSER` и `BYPASSRLS`, при репликации могут выполняться политики защиты строк, определенные на стороне публикации. Если эта роль не может доверять владельцам всех таблиц, добавьте в строку подключения `options=-crow_security=off`. Если владелец таблицы добавит политику защиты строк позже, при таком значении параметра репликация остановится, но политика выполняться не будет. Используйте раздел *pg_hba* на вкладке **Clusters → Кластер ADPG → Services → ADPG → Primary configuration** пользовательского интерфейса ADCM, чтобы настроить доступ для роли. Роль для репликации должна иметь атрибут `LOGIN`.

Чтобы скопировать исходные данные таблицы, роль, используемая для подключения репликации, должна иметь привилегию `SELECT` для опубликованной таблицы или быть суперпользователем.

Чтобы создать публикацию, пользователь должен иметь привилегию `CREATE` в базе данных.

Чтобы добавлять таблицы в публикацию, пользователь должен иметь права владельца для этих таблиц. Создавать публикации, публикующие все таблицы, разрешено только суперпользователям.

Создавать подписки разрешено только суперпользователям.

Процесс применения изменений подписки выполняется в локальной базе данных с правами суперпользователя.

Настройка балансировки нагрузки

Евгения Кузина

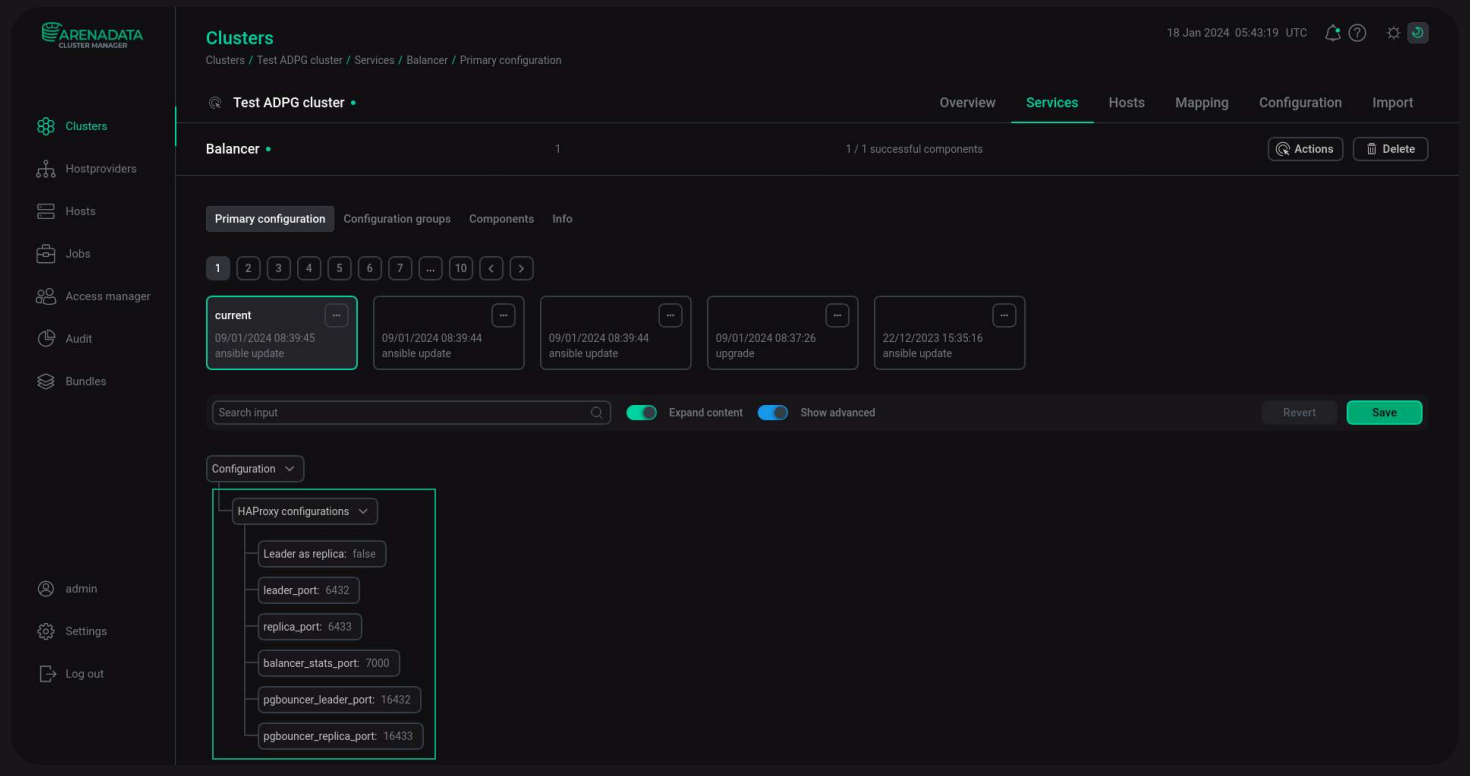
Назад к содержанию

ПРИМЕЧАНИЕ

Балансировка нагрузки (load balancing) доступна в версии ADPG Enterprise.

Кластер ADPG реализует балансировку нагрузки, которая заключается в распределении клиентских запросов по хостам базы данных. За дополнительной информацией об этом функционале обратитесь к статье Балансировка нагрузки.

Сервис, который отвечает за балансировку, называется **Balancer**. Для того чтобы его настроить, перейдите на страницу ***Clusters***, выберите кластер ADPG и переключитесь на вкладку ***Services***. Кликните сервис ***Balancer*** и раскройте ноду ***HAProxy configurations*** на вкладке ***Primary configuration*** (см. Настройка сервисов).



Настройки сервиса Balancer

С настройками по умолчанию запросы на запись обрабатываются только нодой-лидером, а на чтение передаются репликам. Вы можете разрешить лидеру также обрабатывать запросы на чтение. Для этого активируйте флажок ***leader as replica***.

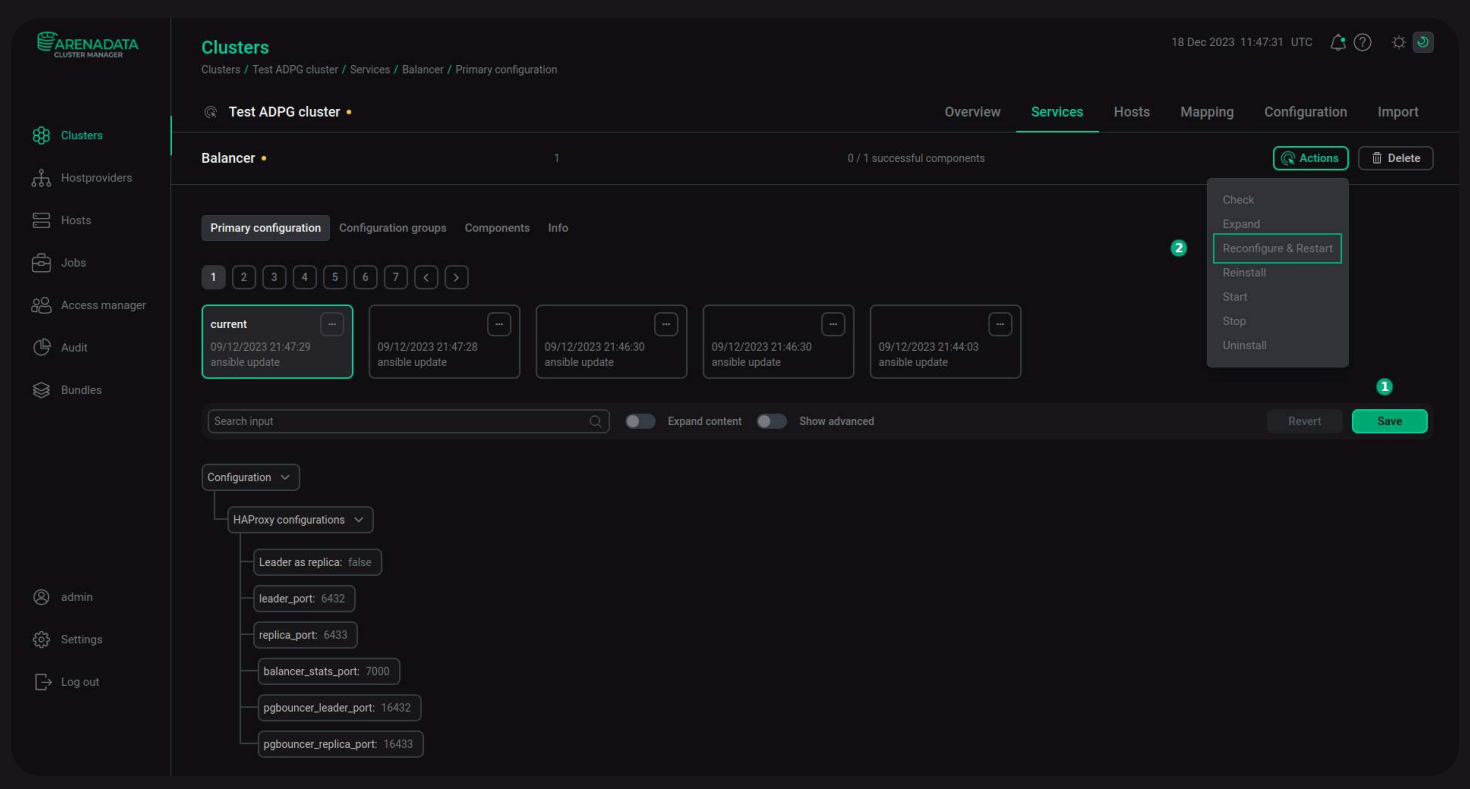
Опция ***leader_port*** назначает порт на хосте с компонентом HAProxy, на который должны приходить транзакции на запись. Значение по умолчанию `6432`.

Опция ***replica_port*** определяет порт на хосте с компонентом HAProxy, на который должны приходить транзакции на чтение. Значение по умолчанию `6433`.

Опция ***balancer_stats_port*** определяет порт, на котором доступна веб-страница с отчетом статистики HAProxy. Значение по умолчанию `7000`.

Опции ***pgbouncer_leader_port*** и ***pgbouncer_replica_port*** используются, если PgBouncer включен. Все подключения к порту `pgbouncer_leader_port` передаются в экземпляр PgBouncer ноды-лидера ADPG. Значение `pgbouncer_leader_port` по умолчанию — `16432`. Все подключения к порту `pgbouncer_replica_port` передаются экземплярам PgBouncer нод-реплик ADPG. Значение `pgbouncer_replica_port` по умолчанию — `16433`. См. Настройка PgBouncer.

После того как вы изменили настройки, нажмите ***Save*** и выполните действие Reconfigure & Restart. Действие ***Reconfigure & Restart*** обновляет конфигурационные файлы сервиса в соответствии с параметрами конфигурации и перезапускает сервис.



Выполнение действия Reconfigure & Restart

Настройка PgBouncer

Евгений Кузина

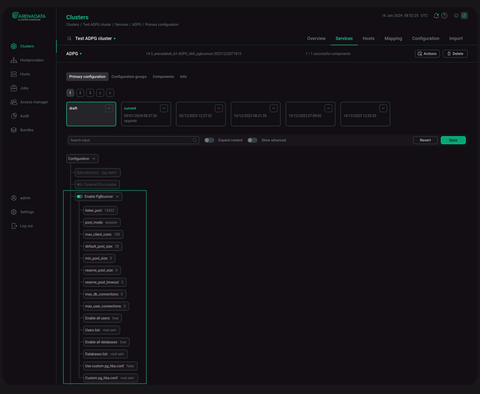
Назад к содержанию

ПРИМЕЧАНИЕ

PgBouncer доступен в версии ADPG Enterprise.

ADPG включает утилиту PgBouncer, которая управляет пулом соединений. Стороннее приложение может подключиться к PgBouncer как к серверу ADPG, и PgBouncer создаст соединение с сервером ADPG или повторно использует одно из существующих подключений. PgBouncer предназначен для снижения издержек, связанных с установкой новых подключений к ADPG. За дополнительной информацией обратитесь к статье Обзор PgBouncer.

Для того чтобы включить PgBouncer и настроить параметры его конфигурации, откройте вкладку ***Services*** кластера ADPG и выберите сервис ***ADPG***. На вкладке ***Primary Configuration*** переведите в активное состояние переключатель ***Enable PgBouncer***. Когда этот переключатель активен, вы можете развернуть соответствующую ноду и указать необходимые настройки PgBouncer.



Настройки PgBouncer

Все доступные настройки перечислены в таблице ниже.

Параметр	Описание	Значение по умолчанию
listen_port	Порт, который прослушивает PgBouncer. Этот порт нельзя поменять у работающего кластера	15432
pool_mode	Указывает, когда соединение с сервером может быть повторно использовано другими клиентами. Может содержать следующие значения: ▪ `session` — сервер возвращается в пул после отключения клиента. ▪ `transaction` — сервер возвращается в пул после завершения транзакции. ▪ `statement` — сервер возвращается в пул после завершения запроса. В этом режиме запрещены транзакции, включающие несколько операторов.	session
max_client_conn	Максимально разрешенное количество клиентских подключений	100
default_pool_size	Количество подключений к серверу, разрешенное для каждой пары пользователь/база данных	20
min_pool_size	Минимальное количество подключений к серверу, которое должно находиться в пуле	0
reserve_pool_size	Количество дополнительных подключений, разрешенных для пула, если клиентское соединение превышает время, указанное в опции `reserve_pool_timeout`	0
reserve_pool_timeout	Если клиенту необходимо ждать дольше указанного времени в секундах, используются дополнительные соединения из `reserve_pool_size`	5
max_db_connections	Максимальное количество подключений к серверу для базы данных	0
max_user_connections	Максимальное количество подключений к серверу для пользователя	0
Enable all users	Если эта опция включена, разрешается доступ всем пользователям с аутентификацией по паролю. Если опция отключена, необходимо указать имена пользователей в параметре `Users list`, чтобы разрешить этим пользователям доступ к ADPG с помощью PgBouncer	Enabled
Users list	Список пользователей, которым разрешен доступ к ADPG с помощью PgBouncer, когда опция `Enable all users` отключена. Например: `postgres,user1`	Empty
Enable all databases	Если эта опция включена, разрешается доступ ко всем существующим базам данных через PgBouncer. Если опция отключена, необходимо указать имена баз данных в параметре `Databases list`, чтобы разрешить доступ к этим базам данных с помощью PgBouncer	Enabled
Databases list	Список баз данных, к которым разрешен доступ с помощью PgBouncer, когда опция `Enable all databases` отключена. Имеет тот же синтаксис, что и секция PgBouncer databases. Например, `postgres = host=localhost port=5432`	Empty
Use custom pg_hba.conf	Если эта опция отключена, для PgBouncer используется основная секция PG_HBA сервиса ADPG. Если опция включена, необходимо указать настройки для PgBouncer в параметре ***Custom pg_hba.conf***. В этом случае PgBouncer будет использовать настройки из ***Custom pg_hba.conf***	Disabled
Custom pg_hba.conf	Пользовательская секция pg_hba.conf для PgBouncer	Empty

Для того чтобы применить новые значения, кликните ***Save*** и выполните действие Reconfigure & Restart сервиса ***ADPG***.

В ADPG PgBouncer использует клиентскую аутентификацию `hba`. Если опция ***Use custom pg_hba.conf*** отключена, фактический тип аутентификации загружается из файла *pg_hba.conf*. Если включен параметр ***Use custom pg_hba.conf***, используются настройки из ***Custom pg_hba.conf***.

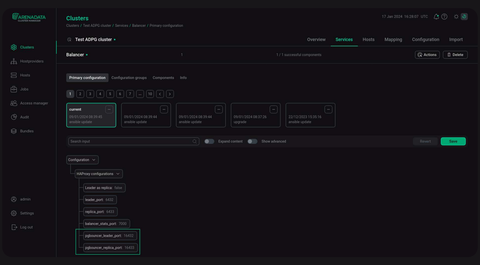
По умолчанию опция ***Enable all users*** включена. Она обеспечивает доступ всем пользователям с аутентификацией по паролю. Вы можете отключить эту опцию и указать список пользователей, которым разрешен доступ к ADPG с помощью PgBouncer, в параметре ***Users list***.

Опция ***Enable all databases*** позволяет получить доступ ко всем существующим базам данных с помощью PgBouncer. Вы можете отключить эту опцию и указать список баз данных, к которым разрешен доступ с помощью PgBouncer, в параметре ***Databases list***.

Если сервис ***Balancer*** не добавлен в кластер, все клиентские запросы должны отправляться на `listen_port` утилиты PgBouncer как на нодах-лидерах, так и на нодах-репликах.

Если кластер ADPG использует сервис ***Balancer***, соединения с нодой-лидером должны отправляться на порт балансировщика, указанный в параметре `pgbouncer_leader_port` сервиса ***Balancer***. Все подключения к этому порту передаются в экземпляр PgBouncer ноды-лидера ADPG. Значение по умолчанию `pgbouncer_leader_port` — `16432`. Соединения с нодами-репликами должны отправляться на порт балансировщика, указанный в параметре `pgbouncer_replica_port` сервиса ***Balancer***. Все подключения к этому порту передаются экземплярам PgBouncer нод-реплик ADPG. Значение по умолчанию `pgbouncer_replica_port` — `16433`.

Чтобы изменить параметры `pgbouncer_leader_port` и `pgbouncer_replica_port`, откройте вкладку ***Services*** кластера ADPG и выберите сервис ***Balancer***. Укажите параметры на вкладке ***Primary configuration***.



Настройки сервиса Balancer

После указания настроек кликните ***Save*** и выполните действие Reconfigure & Restart сервиса ***Balancer***, чтобы применить изменения.

Переключение с лидера на реплику в ADPG

Евгения Кузина

Назад к содержанию

Как упоминалось в статье Обзор архитектуры ADPG, ADPG использует Patroni для создания кластера на основе потоковой репликации PostgreSQL и реализации отказоустойчивости (High Availability,HA). Если необходимо переключиться с ноды-лидера кластера ADPG на реплику, можно сделать это вручную, с помощью команды patronictl switchover.

Эту команду можно использовать только в полностью исправном кластере, в котором присутствует:

- нода-лидер;
- синхронизированная реплика.

Выполните следующие шаги для переключения лидера:

1. Запустите действие кластера Check, чтобы убедиться в его рабочем состоянии. Если ошибок не обнаружено, можно перейти к следующим пунктам.
2. Подключитесь к одной из нод сервиса ADPG под пользователем `postgres`:

```
$ sudo su - postgres
```

3. Выполните команду patronictl list для отображения членов кластера (cluster members):

```
$ patronictl -c /etc/adpg16-patroni/adpg16-patroni.yml list
```

где `/etc/adpg16-patroni/adpg16-patroni.yml` — путь к файлу конфигурации Patroni.

Результат:

```
+ Cluster: test_adpg_cluster (7397708120470393282) ---------+---------+----+-----------+
| Member                           | Host         | Role    | State   | TL | Lag in MB |
+----------------------------------+--------------+---------+---------+----+-----------+
| test_adpg_cluster_eek-adpg-adpg1 | 10.92.40.49  | Leader  | running | 10 |           |
| test_adpg_cluster_eek-adpg-adpg2 | 10.92.42.110 | Replica | running | 10 |         0 |
| test_adpg_cluster_eek-adpg-adpg3 | 10.92.42.217 | Replica | running | 10 |         0 |
+----------------------------------+--------------+---------+---------+----+-----------+
```

4. Выполните команду `patronictl switchover`:

```
$ patronictl -c <путь к patroni.yml> switchover --leader <текущий лидер> --candidate <новый лидер>
```

Используйте в качестве параметров имена членов кластера из списка выше:

```
$ patronictl -c /etc/adpg16-patroni/adpg16-patroni.yml switchover --leader test_adpg_cluster_eek-adpg-adpg1 --candidate
test_adpg_cluster_eek-adpg-adpg3
```

Следующий текст будет выведен:

```
When should the switchover take place (e.g. 2024-08-02T15:28 )  [now]:
```

Нажмите `Enter`, чтобы подтвердить, что лидер должен быть переключен немедленно.

Напечатайте `y` в ответ на следующее сообщение:

```
Are you sure you want to switchover cluster test_adpg_cluster, demoting current master test_adpg_cluster_eek-adpg-adpg1? [y/N]:
```

Результат:

```
2024-08-02 14:29:02.22919 Successfully switched over to "test_adpg_cluster_eek-adpg-adpg3"
+ Cluster: test_adpg_cluster (7397708120470393282) ---------+---------+----+-----------+
| Member                           | Host         | Role    | State   | TL | Lag in MB |
+----------------------------------+--------------+---------+---------+----+-----------+
| test_adpg_cluster_eek-adpg-adpg1 | 10.92.40.49  | Replica | stopped |    |   unknown |
| test_adpg_cluster_eek-adpg-adpg2 | 10.92.42.110 | Replica | running | 10 |         0 |
| test_adpg_cluster_eek-adpg-adpg3 | 10.92.42.217 | Leader  | running | 10 |           |
+----------------------------------+--------------+---------+---------+----+-----------+
```

5. Выполните следующую команду, чтобы убедиться, что лидер был изменен:

```
$ patronictl -c /etc/adpg16-patroni/adpg16-patroni.yml list
```

В столбце `Role` отображается, является ли узел лидером или репликой:

```
+ Cluster: test_adpg_cluster (7397708120470393282) ---------+---------+----+-----------+
| Member                           | Host         | Role    | State   | TL | Lag in MB |
+----------------------------------+--------------+---------+---------+----+-----------+
| test_adpg_cluster_eek-adpg-adpg1 | 10.92.40.49  | Replica | running | 11 |         0 |
| test_adpg_cluster_eek-adpg-adpg2 | 10.92.42.110 | Replica | running | 11 |         0 |
| test_adpg_cluster_eek-adpg-adpg3 | 10.92.42.217 | Leader  | running | 11 |           |
+----------------------------------+--------------+---------+---------+----+-----------+
```

При выполнении команды switchover увеличивается timeline (столбец `TL`). Его увеличение на всех нодах — признак того, что все прошло успешно.

6. Выполните действие кластера Check, чтобы убедиться, что кластер работает корректно.

SQL-дамп

Евгений Кузина

AD.PG

Содержание

Назад к содержанию

- Создание SQL-дампа
- Восстановление SQL-дампа
- Создание дампа кластера
- Резервное копирование больших баз данных
  - Использование сжатых дамп-файлов
  - Использование split
  - Использование дамп-формата custom
  - Использование параллельной выгрузки в pg_dump

Техника резервного копирования с помощью SQL-дампа создает файл с командами SQL, который воссоздаст базу данных в том же состоянии, в каком она была на момент создания дампа.

Создание SQL-дампа

PostgreSQL включает утилиту pg_dump, создающую дамп-файлы. Утилита `pg_dump` не требует блокировки доступа других пользователей и может создавать SQL-дампы во время использования базы данных.

Базовый синтаксис: `pg_dump`:

```
$ pg_dump <имя_базы_данных> > <имя_файла>
```

Где:

- `<имя_базы_данных>` — имя базы данных, для которой создается резервная копия;
- `<имя_файла>` — имя создаваемого дамп-файла.

Утилита `pg_dump` записывает результат в стандартный вывод. Дамп может быть выведен в виде текстового файла, содержащего скрипт, или архивного файла (в формате `custom`, `directory` или `tar`). За дополнительной информацией обратитесь к секции Options статьи pg_dump. Приведенная выше команда создает текстовый файл со скриптом.

Дамп-файлы, содержащие скрипт, представляют собой текстовые файлы с SQL-командами. Чтобы восстановить базу данных из такого файла, используется утилита `psql`. Дамп-файлы можно использовать для создания резервных копий и для переноса базы данных на машины с другой архитектурой. Также можно вручную отредактировать файл со скриптом для восстановления базы данных в другой СУБД.

Утилита `pg_dump` выполняет резервное копирование одной базы данных. Для резервного копирования всего кластера или глобальных объектов, общих для всех баз данных в кластере, таких как роли и табличные пространства, используйте pg_dumpall.

Поскольку `pg_dump` является клиентским приложением, резервную копию можно создать с любого удаленного хоста, имеющего доступ к базе данных. У него должен быть доступ на чтение ко всем таблицам, резервные копии которых нужно создать. Чтобы создать резервную копию всей базы данных, практически всегда необходимо запускать `pg_dump` от имени суперпользователя. Если у вас нет таких привилегии, можно создать резервные копии частей базы данных, к которым у вас есть доступ на чтение. Для этого используйте параметры `-n` (схема) или `-t` (таблица), как описано в статье pg_dump.

В приведенном ниже примере выполняется резервное копирование базы данных `books_store` в файл *books_store20221215.sql*.

```
$ pg_dump books_store > books_store20221215.sql
```

В следующем примере создается резервная копия таблицы `books` базы данных `books_store`.

```
$ pg_dump -d books_store -t books > books_20221215.sql
```

Чтобы указать, к какому серверу базы данных должен обращаться `pg_dump`, используйте параметры командной строки `-h` (хост) и `-p` (порт). Хост по умолчанию — localhost или хост, указанный в переменной окружения `PGHOST`. Порт по умолчанию задается в переменной окружения `PGPORT`. Если `PGPORT` не назначен, `pg_dump` использует значение, указанное во время компиляции. Значение порта по умолчанию для сервиса ADPG — `5432`. Приложение `pg_dump` подключается с именем пользователя базы данных, которое равно текущему имени пользователя операционной системы. Чтобы переопределить это, укажите параметр `-U` или установите переменную среды `PGUSER`.

В приведенном ниже примере выполняется резервное копирование базы данных `books_store`, которая находится на хосте `10.92.6.26`. Подключение осуществляется от имени пользователя `postgres`.

```
$ pg_dump -h 10.92.6.26 -U postgres books_store > books_store20221215.sql
```

Архивные файлы утилиты `pg_dump` предназначены для переноса баз данных между различными архитектурами. За подробностями обратитесь к статье pg_dump.

> **ПРИМЕЧАНИЕ**
>
> Дамп-файлы, созданные `pg_dump`, можно загрузить в более новые версии PostgreSQL, тогда как резервное копирование на уровне файлов и непрерывное архивирование (PITR) зависят от версии PostgreSQL.

Восстановление SQL-дампа

Для восстановления данных из дамп-файлов используется утилита psql. Базовый синтаксис команды для восстановления дампа:

```
$ psql <имя_базы_данных> < <имя_файла>
```

Где:

- `<имя_базы_данных>` — имя базы данных, которая должна быть восстановлена;
- `<имя_файла>` — имя дамп-файла.

Приведенная выше команда не создает указанную базу данных. Необходимо создать её из шаблона template0 перед выполнением команды.

```
$ createdb -T template0 books_store
```

> **ПРИМЕЧАНИЕ**
>
> Утилита `pg_dump` создает дампы относительно `template0`. Языки, процедуры и другие элементы, добавленные через шаблон `template1`, также будут помещены в дамп-файл. Если вы используете измененный `template1`, создайте пустую базу данных из шаблона `template0`, как показано в приведенном выше примере.

Утилита `psql` продолжает выполнять скрипт после возникновения SQL-ошибок. Вы можете использовать переменную `ON_ERROR_STOP`, чтобы изменить это поведение и обеспечить выход из `psql` со статусом `3` в случае ошибок.

```
$ psql --set ON_ERROR_STOP=on books_store < books_store20221215.sql
```

Перед восстановлением SQL-дампа все пользователи, которые владели объектами или имели права на объекты в выгруженной базе данных, должны существовать. Если они отсутствуют, при восстановлении базы данных будут ошибки переподания объектов с первоначальными владельцами или правами.

После восстановления резервной копии также необходимо запустить ANALYZE для каждой базы данных, чтобы предоставить оптимизатору запросов правильную статистику.

Дампы, выгруженные в архивном формате, восстанавливаются утилитой pg_restore.

> **РЕКОМЕНДАЦИЯ**
>
> Для улучшения производительности восстановления дамп-файлов, обратите внимание на рекомендации из раздела Улучшение производительности pg_dump.

Создание дампа кластера

Утилита `pg_dump` создает резервную копию базы данных. Она не выводит информацию о ролях или табличных пространствах, поскольку они указаны для всего кластера, а не для отдельной базы данных. Утилита pg_dumpall предназначена для создания дампа всего кластера. Она создает резервную копию каждой базы данных в кластере и сохраняет общие данные для всего кластера, такие как определения ролей и табличных пространств.

```
$ pg_dumpall > <имя_файла>
```

Где `<имя_файла>` — это имя дамп-файла, который будет создан.

Базу данных из полученного дамп можно восстановить с помощью `psql`:

```
$ psql -f dumpfile postgres
```

Опция `-f` означает, что `psql` читает команды из указанного файла `dumpfile` `postgres` — имя базы данных. Вы можете указать имя любой существующей базы данных вместо `postgres`, но если вы восстанавливаете резервную копию, содержащую несколько баз данных, в пустой кластер, следует использовать `postgres`. Также необходимо иметь права суперпользователя для восстановления дампа `pg_dumpall`. Если вы используете табличные пространства, убедитесь, что пути к табличным пространствам в дампе соответствуют новой среде.

Утилита `pg_dumpall` добавляет в файл дампа команды для пересоздания ролей, табличных пространств и пустых баз данных, после чего вызывает `pg_dump` для каждой базы данных. Это означает, что хотя каждая база данных внутренне согласована, состояние разных баз данных не синхронизируются.

Чтобы выгрузить только глобальные данные кластера, используйте опцию `pg_dumpall --globals-only`.

Резервное копирование больших баз данных

Операционные системы могут иметь ограничения на максимальный размер файла, это приводит к проблемам при создании больших выходных файлов `pg_dump`. Так как `pg_dump` пишет в стандартный вывод, можно использовать стандартные инструменты Unix, чтобы решить эту проблему.

Использование сжатых дамп-файлов

Вы можете использовать программу сжатия, например gzip:

```
$ pg_dump <имя_базы_данных> | gzip > <имя_файла.gz>
```

Восстановить дамп можно следующим образом:

```
$ gunzip -c <имя_файла.gz> | psql <имя_базы_данных>
```

или

```
$ cat <имя_файла.gz> | gunzip | psql <имя_базы_данных>
```

Использование split

Команда split позволяет разделить вывод на файлы меньшего размера. В следующем примере файл разбивается на фрагменты по 2 ГБ:

```
$ pg_dump <имя_базы_данных> | split -b 2G - <имя_файла>
```

Восстановить дамп можно следующим образом:

```
$ cat <имя_файла>* | psql <имя_базы_данных>
```

Если вы используете GNU split, можно использовать его и gzip вместе:

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

Воспользуйтесь утилитой zcat для восстановления такой резервной копии.

Использование дамп-формата custom

Поскольку ADPG включает в себя библиотеку zlib, формат `custom` утилиты pg_dump сжимает данные. В этом формате размер файла дампа близок к размеру, полученному с помощью gzip, но позволяет выборочно восстанавливать таблицы. Следующая команда выгружает базу данных в формате `custom`.

```
$ pg_dump -Fc <имя_базы_данных> > <имя_файла>
```

Дамп-файл формата `custom` не является скриптом, для его восстановления используется утилита pg_restore:

```
$ pg_restore -d <имя_базы_данных> <имя_файла>
```

Для очень больших баз данных вам может понадобиться объединить использование утилиты split с форматом `custom` или сжатыми дамп-файлами.

Использование параллельной выгрузки в pg_dump

Чтобы ускорить выгрузку большой базы данных, вы можете использовать режим параллельной выгрузки `pg_dump`. При этом одновременно будут выгружаться несколько таблиц. Управлять числом параллельных заданий позволяет параметр `-j`. Параллельная выгрузка поддерживается только для формата `directory`.

```
$ pg_dump -j <число> -F d -f <выходной_каталог> <имя_базы_данных>
```

Вы также можете использовать утилиту `pg_restore` с опцией `-j`, чтобы восстановить дамп в параллельном режиме. Это поддерживается для любого архива в формате `directory` или `custom`, даже если архив создавался не командой `pg_dump -j`.

Резервное копирование на уровне файловой системы

Евгения Кузина

Содержание

Назад к содержанию

Утилита pg_basebackup

Создание архивного файла

Создание снепшота

Использование утилиты rsync

Альтернативная стратегия резервного копирования — прямое копирование файлов, которые PostgreSQL использует для хранения данных в базе данных. По умолчанию ADPG размещает файлы базы данных по следующему пути: */pg_data1/adpg16*.

Для резервного копирования файловой системы вы можете использовать любой удобный для вас метод. Возможные варианты описаны ниже.

Утилита pg_basebackup

Утилита pg_basebackup позволяет создать базовую резервную копию работающего кластера PostgreSQL. Она создает резервные копии, не затрагивая других клиентов базы данных.

Утилита `pg_basebackup` создает точную копию файлов кластера, автоматически включая и отключая режим резервного копирования. Резервные копии создаются для всего кластера. Невозможно сделать копию отдельных баз данных или объектов базы данных. Для выборочного резервного копирования используйте утилиту `pg_dump`, как описано в статье SQL-дамп.

Утилита `pg_basebackup` создает резервные копии через стандартное соединение PostgreSQL, использующее протокол репликации. Соединение должно быть установлено от имени суперпользователя или пользователя с правами запускать репликацию, также *pg_hba.conf* должен разрешать подключения для репликации.

В следующем примере создается резервная копия сервера, находящегося на хосте `10.92.6.36`, и сохраняется в локальном каталоге `/usr/local/pgsql/data`:

```
$ pg_basebackup -h 10.92.6.36 -D /usr/local/pgsql/data
```

Код, приведенный ниже, создает резервную копию локального сервера с одним tar-файлом для каждого табличного пространства и сохраняет его в каталоге `backup2022-12-22`:

```
$ pg_basebackup -D backup2022-12-22 -Ft
```

За дополнительной информацией о параметрах утилиты `pg_basebackup` обратитесь к статье pg_basebackup.

Создание архивного файла

Вы можете создать архивный файл, используя утилиту tar:

```
$ tar -cf backup20221215.tar /pg_data1/adpg16
```

Этот метод имеет два ограничения:

- Сервер базы данных должен быть выключен. Используйте для этого действие Stop. Также необходимо выключить сервер перед восстановлением данных.
- Невозможно создать резервную копию и восстановить только определенные таблицы или базы данных из соответствующих файлов или каталогов. Это не работает, потому что информацию, содержащуюся в этих файлах, нельзя использовать без файлов журналов транзакций (*pg_xact/**), которые содержат состояние всех транзакций. Файл таблицы можно использовать только с этой информацией. Резервные копии файловой системы должны использоваться только для полного резервного копирования и восстановления всего кластера базы данных.

Создание снепшота

Другой подход к резервному копированию файловой системы заключается в создании снепшота каталога данных, если файловая система поддерживает эту функцию. Типичная процедура состоит в том, чтобы сделать "замороженный" снепшот тома, содержащего базу данных, затем скопировать каталог данных из снепшота на устройство резервного копирования, а затем освободить "замороженный" снепшот. Это подход можно применять во время работы сервера базы данных. Однако резервная копия, созданная таким образом, сохраняет файлы базы данных в состоянии, как если бы сервер базы данных не был выключен должным образом. Когда запустится сервер базы данных с резервными копиями, он будет считать, что предыдущий процесс сервера аварийно завершил работу, и накатит журналы WAL. Поэтому необходимо включать файлы WAL в резервную копию. Вы можете выполнить команду CHECKPOINT перед созданием снепшота, чтобы сократить время восстановления.

Если ваша база данных распределена по нескольким файловым системам, может не быть способа получить одновременные "замороженные" снепшоты всех томов во время работы базы данных. Например, если ваши файлы данных и журнал WAL находятся на разных дисках или если табличные пространства находятся в разных файловых системах, невозможно использовать снепшоты для резервного копирования, поскольку снепшоты должны быть одновременными. В этом случае можно остановить сервер базы данных и сделать все снепшоты.

Использование утилиты rsync

Утилиту rsync также можно использовать для резервного копирования файловой системы. Запустите `rsync` во время работы сервера базы данных, затем выключите сервер базы данных, чтобы выполнить `rsync --checksum` (`--checksum` необходим, потому что `rsync` различает время только с точностью до секунд). Второй `rsync` быстрее, чем первый, потому что ему нужно передавать меньше данных. Итоговый результат будет согласован, потому что сервер был остановлен. Этот метод позволяет создавать резервные копии файловой системы с минимальным временем простоя.

ПРИМЕЧАНИЕ

Резервная копия файловой системы обычно больше, чем дамп SQL. Например, `pg_dump` не нужно выгружать содержимое индексов, а только команды для их пересоздания. Однако резервное копирование файловой системы может быть быстрее.

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

Евгений Кузина

Содержание

Назад к содержанию

Обзор

Настройка архивирования WAL

Создание базовой резервной копии

Использование низкоуровневого API для создания базовой резервной копии

Восстановление непрерывной архивной копии

Обзор

PostgreSQL записывает каждое изменение, внесенное в файлы базы данных, в журнал предзаписи WAL (write-ahead log). Файлы WAL находятся в каталоге *pg_wal*. По умолчанию ADPG размещает этот каталог по следующему пути */pg_data1/adpg16/pg_wal*, где */pg_data1* — значение параметра `data directory`. За дополнительной информацией об этом параметре можно обратиться к статье Конфигурационные параметры. В случае сбоя системы PostgreSQL воспроизведет записи из записей WAL, сделанных с момента последней контрольной точки.

Для создания резервной копии данных можно комбинировать резервное копирование на уровне файловой системы с файлами WAL. Сначала нужно восстановить резервную копию файловой системы, а затем применить записи WAL к базе данных, чтобы привести систему в текущее состояние. Этот подход имеет следующие преимущества:

- В качестве начальной точки для восстановления необязательно иметь полностью согласованную копию на уровне файлов. Когда PostgreSQL применяет записи WAL, он исправляет любые несоответствия в резервной копии. Для него не нужна возможность делать снепшоты файловой системы, и можно использовать tar или аналогичный инструмент архивации.
- Поскольку при восстановлении можно обрабатывать неограниченную последовательность файлов WAL, непрерывную резервную копию можно получить просто продолжая архивировать файлы WAL. Это полезно для больших баз данных, где требуется значительный объем ресурсов для частого создания полных резервных копий.
- Нет необходимости повторять операции из всех записей WAL. Можно остановиться в любой момент и получить непротиворечивый моментальный снимок базы данных на определенный момент времени. Таким образом, можно восстановить базу данных на любой момент времени с момента создания базовой резервной копии.
- Если сервер постоянно передает файлы WAL на другую машину, которая загружена из той же базовой резервной копии, эта машина имеет текущую копию базы данных и может заменить сервер в случае сбоя.

ПРИМЕЧАНИЕ

Утилиты `pg_dump` и `pg_dumpall` не создают резервные копии на уровне файловой системы и не могут использоваться как часть решения для непрерывного архивирования.

Обратите внимание, что PITR поддерживает восстановление всего кластера базы данных, а не его части. Также PITR требует дополнительное хранилище для базовой резервной копии и архивных файлов WAL.

Для успешного использования PITR необходима непрерывная последовательность архивируемых файлов WAL, которая начинается не позднее момента запуска резервного копирования. Вы должны настроить и протестировать процедуру архивирования файлов WAL до того, как сделаете первую базовую резервную копию.

Настройка архивирования WAL

PostgreSQL создает бесконечно длинную последовательность записей WAL. Система делит эту последовательность на файлы сегменты WAL, размер которых обычно составляет 16 МБ. Файлы сегментов имеют числовые имена, отражающие их положение в абстрактной последовательности WAL. Если сервер не использует архивирование WAL, система создает столько файлов сегментов, сколько нужно, а затем "перерабатывает" (перезаписывает) неужелые файлы сегментов в сегменты с большими порядковыми номерами. Предполагается, что файлы сегментов, содержимое которых предшествует последней контрольной точке, могут быть переработаны.

Чтобы заархивировать данные WAL, необходимо где-нибудь сохранять содержимое каждого файла сегмента после его заполнения до того, как файл сегмента будет переработан. Например, можно скопировать файлы сегментов в каталог, подключенный к NFS, на другом компьютере или записать их на внешний диск. PostgreSQL позволяет администратору указать команду оболочки для создания копии завершенного файла сегмента. Команда может быть простой или вызывать shell-скрипт.

Чтобы включить архивирование WAL, установите значение `replica` или выше для конфигурационного параметра wal_level, для параметра archive_mode — значение `on` и укажите команду оболочки для archive_command. Используйте скрипты ***ADPG configurations*** на вкладке ***Clusters → ADPG cluster → Services → ADPG → Primary configuration*** в пользовательском интерфейсе ADCM, чтобы задать эти параметры. Описание ***ADPG configurations*** можно найти в статье Конфигурационные параметры.

В `archive_command` используйте `%p` вместо пути к архивируемому файлу и `%f` вместо имени файла. Путь к архивируемому файлу задается относительно каталога данных кластера. Например, следующая команда `archive_command` копирует сегменты WAL в каталог */mnt/server/archivedir*:

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

Во время выполнения сервер заменяет параметры `%p` и `%f` значениями, и команда может выглядеть следующим образом:

```
test ! -f /mnt/server/archivedir/000000010000000000000A0A && cp pg_wal/000000010000000000000A0A
/mnt/server/archivedir/000000010000000000000A0A
```

Аналогичная команда генерируется для каждого нового архивируемого файла. Команда архивирования выполняется от имени того же пользователя, который запускает ADPG.

Команда архивации не должна перезаписывать существующие архивные файлы. Это важная функция безопасности для сохранения целостности архива в случае ошибок администратора (например, при отправке вывода двух разных серверов в один и тот же каталог архива). В приведенном выше примере команда архивации включает команду `test` для этой цели.

Пример определения параметра `archive_command` с shell-скриптом

```
archive_command : 'local_backup_script.sh "%p" "%f"'
```

Поскольку архивные файлы WAL содержат информацию из вашей базы данных, убедитесь, что каталог с этими файлами имеет права доступа, гарантирующие безопасность данных.

Команда архивации должна возвращать нулевой код завершения только в случае успеха. Когда PostgreSQL получает нулевой результат, это означает, что файл заархивирован и может быть переработан. Если команда возвращает ненулевой статус, PostgreSQL предполагает, что файл не заархивирован, пока не добьется успеха.

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

PostgreSQL вызывает команду архивации для завершенных сегментов WAL. Если сервер генерирует небольшой WAL-трафик, это может привести к большой задержке между завершением транзакции и её записью в архивное хранилище. Вы можете установить параметр archive_timeout, чтобы ограничить эту задержку. Этот параметр заставляет сервер переключаться на новый файл сегмента WAL по истечении указанного периода времени.

Вы также можете заставить сервер переключиться на новый файл WAL с помощью функции `pg_switch_wal`, если хотите заархивировать завершенную транзакцию как можно скорее. Другие функции, связанные с управлением WAL, перечислены в разделе Backup Control Functions.

Создание базовой резервной копии

Вы можете использовать утилиту pg_basebackup, как описано в статье Резервное копирование на уровне файловой системы, чтобы создать базовую резервную копию.

Чтобы использовать резервную копию, вам необходимо сохранить все файлы сегментов WAL, созданные во время и после резервного копирования файловой системы. Процесс базового резервного копирования создает файл истории резервного копирования, который хранится в каталоге архива WAL. Этот файл назван так же, как первый файл сегмента WAL, необходимый для резервного копирования файловой системы. Например, если начальный файл WAL имеет имя 000000010000123400000053D, файл истории резервного копирования будет называться 000000010000123400000053D.00709330.backup. Вторая часть имени файла указывает точную позицию в файле WAL и может быть проигнорирована.

Необходимо хранить все заархивированные файлы WAL с момента последнего базового резервного копирования. Поэтому интервал резервного копирования обычно выбирается исходя из размера хранилища файлового архива WAL. Также следует учитывать время, необходимое для восстановления данных. Система должна воспроизвести все сегменты WAL, заархивированные с момента последнего базового резервного копирования, для восстановления.

Использование низкоуровневого API для создания базовой резервной копии

Базовые резервные копии низкого уровня могут быть сделаны немонопольным или монопольным способом. Рекомендуется использовать немонопольный метод, так как монопольный метод считается устаревшим. Немонопольное резервное копирование позволяет запускать другие параллельные процессы резервного копирования, которые могут использовать те же API резервного копирования или утилиту `pg_basebackup`.

Создание базовой резервной копии с помощью низкоуровневого API содержит больше шагов, чем метод с использованием `pg_basebackup`. Очень важно, чтобы эти шаги выполнялись по порядку, и следующий шаг выполнялся только в случае успеха предыдущего.

1. Убедитесь, что архивирование WAL включено и работает.
2. Подключитесь к серверу как пользователь, у которого есть права на запуск функции pg_start_backup, и выполните команду:

```
SELECT pg_start_backup('label', false, false);
```

Функция `pg_start_backup` имеет следующий синтаксис:

```
pg_start_backup(<уникальный_идентификатор>, <быстрая_архивная_копия>, <монопольное_копирование>)
```

Где:

- `<уникальный_идентификатор>` — строка, являющаяся уникальным идентификатором операции резервного копирования.
- `<быстрая_архивная_копия>` — по умолчанию, `pg_start_backup` может работать длительное время, так как `pg_start_backup` выполняет контрольную точку, а операции ввода-вывода, необходимые для контрольной точки, распределяются за значительный период времени, равный половине интервала между контрольными точками. Обратитесь к статье checkpoint_completion_target за дополнительной информацией. Это создает к минимуму влияние на обработку запросов. Если вы хотите начать резервное копирование как можно скорее, измените второй параметр на `true`. В этом случае контрольная точка будет выполнена немедленно без ограничения объема операций ввода-вывода.
- `<монопольное_копирование>` — определяет, будет ли `pg_start_backup` инициировать монопольное базовое резервное копирование. В нашем примере производится немонопольное копирование.
  Соединение, которое вызывает `pg_start_backup`, должно поддерживаться до тех пор, пока резервное копирование не будет завершено, иначе процесс будет автоматически прерван.
3. Создайте резервную копию, применив любой способ резервного копирования файловой системы. Обратитесь к статье Backing up the data directory за дополнительной информацией о том, какие каталоги должны быть скопированы.
4. Используя то же соединение, что на шаге 2, вызовите функцию pg_stop_backup:

```
SELECT * FROM pg_stop_backup(false, true);
```

Функция `pg_stop_backup` имеет следующий синтаксис:

```
pg_stop_backup(<монопольное_копирование>, <ожидание_архивирования_wal>)
```

Где:

- `<монопольное_копирование>` — указывает, используется ли монопольный режим при создании резервной копии. Этот параметр должен соответствовать значению из вызова `pg_start_backup`.
- `<ожидание_архивирования_wal>` — определяет, должна ли `pg_stop_backup` ждать, пока файлы WAL будут заархивированы.
  Эта команда завершает режим резервного копирования. Ведущий сервер автоматически переключается на следующий сегмент WAL. На резервном сервере автоматическое переключение сегментов WAL невозможно, поэтому вы можете выполнить `pg_switch_wal` на ведущем, чтобы произвести переключение вручную. Такое переключение позволяет получить готовый к архивированию последний сегмент WAL, записанный в процессе резервного копирования.
  Функция `pg_stop_backup` возвратит одну строку с тремя значениями. Второе из них нужно записать в файл *backup_label* в корневом каталоге резервной копии. Третье значение, если оно не пустое, должно быть записано в файл *tablespace_map*. Эти файлы являются критическими для восстановления копии и должны быть записаны байт за байтом без изменений, для чего может потребоваться открыть файл в двоичном редакторе.
5. Необходимо заархивировать файлы сегментов WAL, активные во время создания резервной копии. Функция `pg_stop_backup` в первом возвращаемом значении указывает, какой последний сегмент WAL требуется для формирования полного набора файлов резервной копии. На ведущем сервере, если включен режим архивации (параметр `archive_mode` равен `on`) и аргумент `wait_for_archive` функции `pg_stop_backup` равен `true`, функция `pg_stop_backup` не завершится, пока не будет заархивирован последний сегмент. На резервном сервере значением `archive_mode` должно быть `always`, чтобы `pg_stop_backup` ожидала архивации. Так как параметр `archive_command` задан, эти файлы будут заархивированы автоматически.

Восстановление непрерывной архивной копии

Для восстановления из резервной копии выполните следующие действия:

1. Остановите сервер.
2. Скопируйте каталог данных кластера и все табличные пространства во временное хранилище. Эта операция требует, чтобы в вашей системе было достаточно свободного места для хранения двух копий базы данных. Если у вас недостаточно места для сохранения всех данных, необходимо сохранить содержимое каталога *pg_wal*, потому что в нем могут быть файлы, которые не были заархивированы до системного сбоя.
3. Удалите все существующие файлы и подкаталоги в каталоге данных кластера и корневых каталогах всех табличных пространств.
4. Восстановите файлы базы данных из резервной копии файловой системы. Убедитесь, что они восстановлены с правильным владельцем (пользователь, запускающий сервер, а не `root`) и с правильными разрешениями. Если вы используете табличные пространства, проверьте правильность восстановления символьных ссылок в *pg_tblspc/*.
5. Удалите все файлы, присутствующие в *pg_wal/*. Эти файлы взяты из резервной копии файловой системы и, поэтому, скорее всего, они будет более старыми, чем текущие. Если вы не использовали *pg_wal/*, создайте этот каталог заново с соответствующими правами, а если вы использовали символьную ссылку, восстановите ее.
6. Если у вас есть незаархивированные файлы сегментов WAL, которые вы сохранили на шаге 2, скопируйте их в *pg_wal/*. Не перемещайте их, чтобы у вас остались неизменные файлы, если возникнет проблема, и все придется начинать сначала.
7. Установите параметры восстановления в разделе ***ADPG configurations*** и создайте файл *recovery.signal* в каталоге данных кластера (каталог данных в ADPG по умолчанию — */pg_data1/adpg16*). Вы также можете временно изменить скрипт ***PG_HBA*** в пользовательском интерфейсе ADCM, чтобы запретить подключение пользователей до тех пор, пока система не будет восстановлена.
8. Запустите сервер. Сервер перейдет в режим восстановления и считает необходимые архивные файлы WAL. Если восстановление прервалось из-за внешней ошибки, сервер можно перезапустить, и восстановление продолжится. По завершении процесса восстановления сервер удалит *recovery.signal*, чтобы предотвратить случайный повторный вход в режим восстановления, и начнет стандартные операции с базой данных.
9. Проверьте содержимое базы данных и убедитесь, что желаемое состояние восстановлено. Если это так, отредактируйте раздел ***PG_HBA***, чтобы разрешить пользователям подключаться к восстановленной базе данных. В противном случае вернитесь к шагу 1.

На шаге 7 необходимо задать конфигурационные параметры, которые вы будете использовать для восстановления и до какой точки. Используйте параметр restore_command, указывающий как извлекать файлы WAL из архива. Этот параметр подобен `archive_command`. Он должен содержать команду оболочки или `%p` вместо пути, куда будут копироваться файлы WAL и `%f` вместо имени файла журнала. Путь задается относительно каталога данных кластера.

Пример

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

Параметр `restore_command` также может содержать shell-скрипт.

Если сервер не может найти сегменты WAL в архиве, он ищет их в каталоге *pg_wal/*. Это позволяет использовать последние незаархивированные сегменты. Сегменты из архива имеют более высокий приоритет, чем файлы из *pg_wal/*.

Процесс восстановления обрабатывает все доступные сегменты WAL и восстанавливает базу данных на текущий момент времени или как можно ближе к этому моменту с учетом доступных сегментов WAL. Поэтому восстановление заканчивается сообщением "файл не найден", точный текст сообщения об ошибке зависит от текущей команды `restore_command`. Вы также можете увидеть сообщение об ошибке в начале восстановления для файла с именем, похожим на *00000001.history*. Это также не указывает на проблему в большинстве случаев. За дополнительной информацией обратитесь к разделу Timelines.

Если нужно восстановиться до какого-то предыдущего момента времени, укажите точку остановки. Вы можете задать точку остановки (recovery target) по дате/времени, именованной точке восстановления или по завершению транзакции с определенным идентификатором. На данный момент полезным может быть только указание даты/времени или именованной точки восстановления, пока нет никаких средств, позволяющих точно определить, какой идентификатор транзакции нужно выбрать.

Если процесс восстановления обнаруживает повреждение данных WAL, восстановление останавливается на этом этапе и сервер не запускается. В этом случае запустите процесс восстановления с самого начала и укажите точку остановки (recovery target) перед точкой повреждения. Восстановление сможет завершиться удачно. Если восстановление завершается сбоем по внешней причине, такой как системный сбой или недоступность архива WAL, восстановление можно запустить заново, и оно начнется с точки сбоя. Перезапуск восстановления работает так же, как контрольная точка, сервер периодически сохраняет текущее состояние на диск и обновляет файл *pg_control*. Таким образом, уже обработанные файлы WAL не сканируются повторно.

Резервное копирование и восстановление с использованием pgBackRest

Last Updated: 2025-04-26

Содержание

Назад к содержанию

Обзор
Настройка резервного копирования
- Хранилища бэкапов
- Сроки хранения бэкапов
- Сжатие бэкапов
- Логирование
- Расширенная конфигурация pgBackRest

Создание бэкапа
Восстановление кластера ADPG
Получение списка бэкапов
Ручное PITR-восстановление ADPG-кластера

Обзор

ВАЖНО

- Этот функционал устарел и будет удален после минорного обновления ADPG, использующего PostgreSQL 17, используйте вместо него менеджер бэкапов кластера ADPG Control.
- Обратите внимание, что скрипты внешних сервисов хранят и архивируют менеджеры бэкапов ADPG Control не поддерживаются.

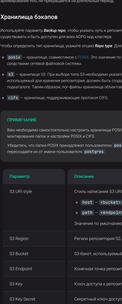
ADPG использует функциональное возвращается утилиты `pgBackRest` для создания резервных копий кластера (бэкапов) и восстановления из его копий. Это предоставляет гибкие возможности для больших баз данных и сценариев непрерывного архивирования. Все действия по резервному копированию выполняются от имени пользователя `postgres`.

Бэкап — это согласованная копия кластера баз данных, которую можно использовать для восстановления после сбоя оборудования, для восстановления из определенной временной точки (Point In Time Recovery, PITR) или для запуска нового резервного сервера. Можно создавать следующие типы бэкапов:

- **Full backup** — `pgBackRest` копирует все файлы данных кластера. Самый надежный, но самый медленный и ресурсоемкий тип бэкапа (full backup). `pgBackRest` может восстановить кластер, используя только полный бэкап. Такой бэкап не зависит от каких-либо других, он автономен и цел.
- **Differential backup** — `pgBackRest` копирует только те файлы базы данных, которые изменились с момента последнего успешного полного резервного копирования (full backup). Чтобы восстановить кластер из дифференциального бэкапа, `pgBackRest` потребуется также файл из текущего дифференциального бэкапа и файл из полного бэкапа (full backup), от которого был сделан данный дифференциальный бэкап. Инкрементный бэкап зависит от других резервных копий. Все резервные копии должны быть целостными, чтобы данные были восстановлены. Если какие-либо восстановленные из инкрементного бэкапа. Последовательность резервных копий может включать и дифференциальные бэкапы.

Настройка резервного копирования

Чтобы включить создание бэкапов с помощью утилиты `pgBackRest`, активируйте переключатель **Enable backups**, отображаемый на вкладке **Primary Configuration** сервиса **ADPG Clusters** > **Кластер ADPG** > **Services** > **ADPG** > **Primary configuration**. После этого вы сможете развернуть вид **Enable backup** и указать настройки `pgBackRest` описанные ниже:



Включение резервного копирования

ПРИМЕЧАНИЕ

После установки всех необходимых параметров не забудьте **Primary Configuration** нажмите **Save**, чтобы сохранить текущую конфигурацию, и запустите действие сервиса ADPG Reconfigure & Restart.

В настройках `pgBackRest` каждый кластер PostgreSQL ADPG для которого создается бэкап, имеет свою конфигурацию, называемую **стэнза (stanza)**. Параметры стэнзы определяют, где расположены файлы данных кластера, параметры архивирования и другие настройки. Значения по умолчанию и в 8 ADPG стэнза создается с именем, равным `backup-stanza` для дополнительных параметров `adpgconfig`. Например, если указано имя `mystanza`, то результат `adpgbr-mystanza`.

`bit` — количество бит, которое нужно сдвинуть хэш-функции, чтобы определить время удаления (в секундах) для каждого файла. Значение по умолчанию — `10`.

`hdd` — количество бит, которое нужно сдвинуть хэш-функции, чтобы определить время удаления (в секундах) для каждого файла. Значение по умолчанию — `15`.

`ssd` — количество бит, которое нужно сдвинуть хэш-функции, чтобы определить время удаления (в секундах) для каждого файла. Значение по умолчанию — `13`.

`nfs` — количество бит, которое нужно сдвинуть хэш-функции, чтобы определить время удаления (в секундах) для каждого файла. Значение по умолчанию — `18`.

`cifs` — количество бит, которое нужно сдвинуть хэш-функции, чтобы определить время удаления (в секундах) для каждого файла. Значение по умолчанию — `18`.

`extreme` — количество бит, которое нужно сдвинуть хэш-функции, чтобы определить время удаления (в секундах) для каждого файла. Значение по умолчанию — `20`.

ПРИМЕЧАНИЕ

Вам необходимо самостоятельно настроить хранилище PDSS и ОТП для корректной работы кластера. ADPG не предоставляет инструменты для настройки PDSS и ОТП.

Убедитесь, что только PDSS имеет разрешение пользователя `postgres` или доступны для него на чтение и запись на каждом хосте. Если это не так, переходите их от имени пользователя `postgres`.

Параметр	Описание
S3 URI style	Стиль написания S3 URI. Поддерживаются следующие стили: - `host` — `<bucket>.<endpoint>.<domain>`. Например, `<bucket>.s3-east-1.amazonaws.com`. - `path` — `<endpoint>.<host>/<bucket>`. Например, `us-east-1.amazonaws.com/mybucket`. Значение по умолчанию — `path`.
S3 Region	Регион репозитория S3, в котором был создан бакет.
S3 Bucket	S3-бакет, используемый для хранения репозиториев с резервными копиями
S3 Endpoint	Конечная точка репозитория S3. Конечная точка должна быть валидна для указанного региона
S3 Key	Ключ доступа к репозиториям S3, используемый для доступа к бакету
S3 Key Secret	Секретный ключ доступа к репозиториям S3, используемый для доступа к бакету

Хранилища бэкапов

Используйте параметр **Backup repo**, чтобы указать путь к репозиторию, в котором `pgBackRest` хранит бэкапы и архивирует сегменты WAL. Репозиторий должен существовать и быть доступен для всех ADPG-нод кластера.

Вы можете настроить несколько репозиториев. Конечная точка бэкапов можно использовать следующие типы хранилищ:

- `posix` — хранилище, совместимое с POSIX. Это значение по умолчанию. Директории на всех нодах должны быть синхронизированы между собой, например, смонтированы сетевой файловой системой.
- `s3` — хранилище S3. Три бэкапа типа S3 необходимо указать настройки, находящиеся в разделе **S3 Configuration**. Настройки S3 подчеркнуты в таблице ниже. Бакет, используемый для хранения репозиториев, должен быть создан заранее. Репозиторий может располагаться в корне бакета (`/`), но обычно его размещают в подпапке. Таким образом, вам потребуется создать только один бакет для нескольких репозиториев.
- `cifs` — хранилище, поддерживающее протокол CIFS.

ПРИМЕЧАНИЕ

Вам необходимо самостоятельно настроить хранилище PDSS и ОТП для корректной работы кластера. ADPG не предоставляет инструменты для настройки PDSS и ОТП.

Убедитесь, что только PDSS имеет разрешение пользователя `postgres` или доступны для него на чтение и запись на каждом хосте. Если это не так, переходите их от имени пользователя `postgres`.

Параметр	Описание
S3 URI style	Стиль написания S3 URI. Поддерживаются следующие стили: - `host` — `<bucket>.<endpoint>.<domain>`. Например, `<bucket>.s3-east-1.amazonaws.com`. - `path` — `<endpoint>.<host>/<bucket>`. Например, `us-east-1.amazonaws.com/mybucket`. Значение по умолчанию — `path`.
S3 Region	Регион репозитория S3, в котором был создан бакет.
S3 Bucket	S3-бакет, используемый для хранения репозиториев с резервными копиями
S3 Endpoint	Конечная точка репозитория S3. Конечная точка должна быть валидна для указанного региона
S3 Key	Ключ доступа к репозиториям S3, используемый для доступа к бакету
S3 Key Secret	Секретный ключ доступа к репозиториям S3, используемый для доступа к бакету

Сроки хранения бэкапов

Вы можете определить параметры хранения бэкапов. **Retention Full type** определяет, представляет ли параметр **Retention Full** период времени (дни) или количество полных бэкапов (full backup), которые нужно сохранить. Доступны следующие значения **Retention Full type**:

- `count` — задается количество бэкапов для хранения, по умолчанию. Значение по умолчанию.
- `time` — полные бэкапы старше значения **Retention full** (в днях) будут удалены из репозитория, если существует хотя бы один бэкап. срок действия которой истекает позднее, чем самое это произошло. Значение **Retention full** определяет, если **Retention full type** (в днях) есть день полных бэкапов. срок (5 дневный и срок 35 дневный. — бэкапы от будут удалены. Это гарантирует, что количество срока хранения 35 дневного 5 дней будет сохранено только полного бэкапа (full backup), что хорошо полного бэкапа (full backup), что может потребоваться для восстановления. Если вы хотите иметь возможность восстановления на любой день в течение 35 дней до истечения срока хранения полного бэкапа (full backup), вам необходимо сохранить все полные бэкапы (full backup) и инкрементальные бэкапы, связанные с ними, также истекает.

Вы можете использовать опцию **Retention diff**, чтобы указать количество сохраняемых дифференциальных бэкапов. По истечении срока хранения дифференциального бэкапа срок хранения всех инкрементальных бэкапов, связанных с ним, также истекает. Если этот параметр не указан, все дифференциальные бэкапы будут сохранены до тех пор, пока не истечет срок хранения полных бэкапов, от которых они зависят.

Сжатие бэкапов

Вы можете настроить сжатие бэкапов, установив флаги **Enable compression** и указав параметр **Compression type**. Используйте один из следующих типов: `bz2`, `lz4`, `lz4 - 1`, `zst`. Значение по умолчанию — `lz`.

Вы также можете установить **Compress level** — уровень сжатия файлов (`1` до `9`), используемый, когда **Compression type** не равно `none`. Если это опция не указана, используется следующие уровни по умолчанию для каждого значения **Compression type**:

- `bz2` — 5;
- `gz` — 6;
- `lz4` — 1;
- `zst` — 3

Логирование

По умолчанию ADPG размещает лог файлы `pgBackRest` в следующую путь: `/var/log/adpg10-pgbackrest`. Используйте опцию **Log path**, чтобы изменить каталог для хранения лог файлов.

Вы также можете установить уровень логирования `pgBackRest` с помощью **Log level**. Возможные значения: `off`, `error`, `warn`, `info`, `detail`, `debug` и `trace`. Значение по умолчанию — `info`.

Расширенная конфигурация pgBackRest

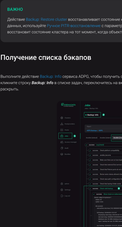
ADPG позволяет задавать настройки `pgBackRest` недоступные в пользовательском интерфейсе. Для этого установите флажок **Use custom config** и укажите пользовательскую конфигурацию в поле **Global options: Custom options**. В этом случае нет настройки `pgBackRest` кроме **Backup stanza** (применяется ко всем нодам). Параметры из **Global options** добавляются в раздел `[global]` конфигурационного файла `pgBackRest`. Конфигурационный файл находится по следующему пути: `/etc/adpg10-pgbackrest/adpg10-pgbackrest.conf`.

ПРИМЕЧАНИЕ

Если вы укажете флаг **Use custom config** и задаете настройки хранилища `pgBackRest`, ADPG запишет эти настройки в конфигурационный файл `pgBackRest`, без проверки. В случае ошибочных настроек сообщения об ошибке не будет выведено в интерфейсе ADCM. Вам необходимо самостоятельно убедиться, что настройки верные, а затем для запуска бэкапов существует и доступна.

Создание бэкапа

После включения резервного копирования и установки всех необходимых параметров, как описано выше, вы можете выполнить действие Backup. Open сервиса ADPG, чтобы создать бэкап. При выполнении этого действия ADCM отобразит диалоговое окно с несколькими параметрами, которые можно настроить:



Изменение типа бэкапа

В открывшемся окне выберите тип бэкапа:

Выбор типа бэкапа

По умолчанию `pgBackRest` создает инкрементные бэкапы. Однако инкрементный бэкап должен основываться на полном бэкапе. Если полного бэкапа (full backup) не существует, `pgBackRest` создаст полный бэкап (full backup) вместо инкрементного, даже если вы выбрали инкрементный бэкап.

Чтобы выбрать тип бэкапа **Backup type**, выбрав подходящее значение. Доступные значения: `full`, `differential` и `incremental`. Значение по умолчанию — `incremental`.

Если вы выбрали тип бэкапа **Backup type**, отличный от `full`, укажите **Delta option**, чтобы выполнить дельта-восстановление. Это позволяет `pgBackRest` восстановить кластер, используя существующие файлы данных, а не восстанавливать все файлы с нуля. Это может значительно ускорить процесс восстановления, поскольку `pgBackRest` может пропустить многие файлы, которые уже существуют и не изменились.

Вы можете указать **Backup stanza**, чтобы создать бэкап для определенной стэнзы. Если вы не укажете стэнзу, `pgBackRest` создаст бэкап для стэнзы по умолчанию, которая называется `backup-stanza`.

Вы можете указать **Backup priority**, чтобы установить приоритет для бэкапа. Приоритет может быть числом от `1` до `999`, где `1` — самый высокий приоритет. Если вы не укажете приоритет, `pgBackRest` использует значение по умолчанию `100`.

Вы можете указать **Backup annotations**, чтобы добавить произвольные аннотации к бэкапу. Аннотации должны быть в формате `key=value`. Вы можете указать несколько аннотаций, разделенных запятыми.

Вы можете указать **Backup start fast**, чтобы принудительно выполнить контрольную точку немедленно, вместо ожидания запланированной контрольной точки. Это может ускорить начало резервного копирования, но может повлиять на производительность базы данных во время выполнения контрольной точки.

Вы можете указать **Backup stop auto**, чтобы автоматически остановить любой другой бэкап, который выполняется в данный момент. Это полезно, если вы хотите убедиться, что ваш бэкап выполняется немедленно, и не хотите ждать завершения другого бэкапа.

Вы можете указать **Backup archive check**, чтобы проверить, что все файлы WAL, необходимые для согласованности бэкапа, были заархивированы. Это гарантирует, что бэкап можно будет восстановить.

Вы можете указать **Backup archive copy**, чтобы скопировать файлы WAL, необходимые для согласованности бэкапа, в каталог бэкапа. Это может увеличить размер бэкапа, но гарантирует, что файлы WAL будут доступны для восстановления.

Вы можете указать **Backup standby**, чтобы выполнить бэкап с резервного сервера. Это может снизить нагрузку на основной сервер, но может привести к задержке в создании бэкапа.

Вы можете указать **Backup manual**, чтобы пометить бэкап как ручной. Ручные бэкапы не удаляются автоматически политикой хранения.

Вы можете указать **Backup block**, чтобы включить инкрементное резервное копирование на уровне блоков. Это может уменьшить размер бэкапа, но может увеличить время, необходимое для создания бэкапа.

Вы можете указать **Backup bundle**, чтобы объединить несколько файлов в один файл. Это может уменьшить количество файлов в бэкапе, но может увеличить время, необходимое для создания бэкапа.

Вы можете указать **Backup bundle limit**, чтобы установить максимальный размер файла, который может быть объединен. Это полезно, если вы хотите ограничить размер файлов в бэкапе.

Вы можете указать **Backup bundle size**, чтобы установить максимальный размер объединенного файла. Это полезно, если вы хотите ограничить размер файлов в бэкапе.

Вы можете указать **Backup compress**, чтобы включить сжатие бэкапа. Это может уменьшить размер бэкапа, но может увеличить время, необходимое для создания бэкапа.

Вы можете указать **Backup compress type**, чтобы установить тип сжатия, используемый для бэкапа. Доступные значения: `gz`, `bz2`, `lz4`, `zst`. Значение по умолчанию — `gz`.

Вы можете указать **Backup compress level**, чтобы установить уровень сжатия, используемый для бэкапа. Доступные значения: `1` до `9`. Значение по умолчанию — `6`.

Вы можете указать **Backup repo**, чтобы выполнить бэкап в определенный репозиторий. Если вы не укажете репозиторий, `pgBackRest` выполнит бэкап в репозиторий по умолчанию.

Вы можете указать **Backup archive timeout**, чтобы установить тайм-аут для архивации файлов WAL. Если архивация файлов WAL занимает больше времени, чем указанный тайм-аут, `pgBackRest` завершит с ошибкой.

Вы можете указать **Backup archive mode**, чтобы установить режим архивации файлов WAL. Доступные значения: `off`, `on`, `always`. Значение по умолчанию — `on`.

Вы можете указать **Backup archive queue**, чтобы установить максимальный размер очереди архивации файлов WAL. Если очередь архивации файлов WAL превышает указанный размер, `pgBackRest` завершит с ошибкой.

Вы можете указать **Backup archive retry**, чтобы установить количество попыток архивации файлов WAL. Если архивация файлов WAL завершается с ошибкой, `pgBackRest` повторит попытку указанное количество раз.

Вы можете указать **Backup archive retry delay**, чтобы установить задержку между попытками архивации файлов WAL. Если архивация файлов WAL завершается с ошибкой, `pgBackRest` подождет указанное количество секунд перед повторной попыткой.

Вы можете указать **Backup archive retry max**, чтобы установить максимальное количество попыток архивации файлов WAL. Если архивация файлов WAL завершается с ошибкой, `pgBackRest` повторит попытку не более указанного количества раз.

Вы можете указать **Backup archive retry max delay**, чтобы установить максимальную задержку между попытками архивации файлов WAL. Если архивация файлов WAL завершается с ошибкой, `pgBackRest` подождет не более указанного количества секунд перед повторной попыткой

Vacuuming

© Антон Бобров

Содержание

- Обзор
- Команды VACUUM и VACUUM FULL
- Освобождение дискового пространства
- Обновление статистики планировщика
- Обновление карты видимости
- Предотвращение сбоя переноса (wraparound) идентификаторов транзакций
- Мультистранзакции и перенос идентификаторов транзакций
- Autovacuum

Обзор

Цель [MVCC](https://postgrespro.ru/docs/postgrespro/current/mvcc) — обеспечить эффективный доступ к данным при одновременном выполнении транзакций. Но у этого подхода есть и обратная сторона: устаревшие версии строк, которые больше не нужны ни одной транзакции, продолжают занимать место на диске. Они называются [мёртвыми строками](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-SPACE-RECOVERY) (dead tuples).

Команда **VACUUM** и выполняющий ту же работу процесс [autovacuum](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#AUTOVACUUM) удаляют мёртвые строки, освобождая занятое ими пространство для повторного использования. Кроме того, они обновляют [статистику планировщика](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-STATISTICS), [карту видимости](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-VISIBILITY-MAP) и [карту свободного пространства](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-FREE-SPACE-MAP), а также [предотвращают сбой переноса идентификаторов транзакций](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-WRAPAROUND).

Обычно **VACUUM** работает в фоновом режиме, не мешая другим операциям. Однако при [определённых условиях](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-WRAPAROUND) он может начать [агрессивно](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-WRAPAROUND) расходовать ресурсы, что приведёт к заметному снижению производительности. Поэтому важно настроить **VACUUM** так, чтобы он успевал справляться с нагрузкой.

Команды VACUUM и VACUUM FULL

Как отмечалось выше, удаление устаревших версий строк выполняется командами VACUUM и [autovacuum](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#AUTOVACUUM). Команда VACUUM FULL выполняет более агрессивную очистку: она полностью перестраивает содержимое таблицы, записывая данные в новый файл и удаляя старый. Это позволяет немедленно освободить неиспользуемое пространство, но требует [монопольной блокировки](https://postgrespro.ru/docs/postgrespro/current/explicit-locking) таблицы.

```
CREATE TABLE test(
  id integer NOT NULL,
  val text
);
INSERT INTO test SELECT id, md5(id) FROM generate_series(1,1000) id;
```

Создается тестовая таблица и заполняется данными.

```
UPDATE test SET val = 'x' WHERE id <= 200;
DELETE FROM test WHERE id > 800;
```

Производятся операции обновления и удаления строк.

```
SELECT * FROM heap_page('test',0);
```

Посмотрим данные о первой странице с помощью [расширения pageinspect](https://postgrespro.ru/docs/postgrespro/current/pageinspect).

```
 ctid | state  | xmin | xmax | hhu | phu | pru | all | nul
------+--------+------+------+-----+-----+-----+-----+-----
 (0,1) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,2) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,3) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,4) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,5) | normal |  496 |  497 | f   | f   | f   | f   | f
(5 rows)
```

Выполняется обычная очистка таблицы.

```
VACUUM test;
```

Посмотрим данные о первой странице.

```
 ctid | state  | xmin | xmax | hhu | phu | pru | all | nul
------+--------+------+------+-----+-----+-----+-----+-----
 (0,1) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,2) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,3) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,4) | normal |  496 |  497 | f   | f   | f   | f   | f
 (0,5) | normal |  496 |  497 | f   | f   | f   | f   | f
(5 rows)
```

Теперь **VACUUM** сможет для некоторых строк установить [признак](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-VISIBILITY-MAP) **t** в **hhu**.

Выполняется [полная очистка](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#VACUUM-FOR-SPACE-RECOVERY) таблицы командой VACUUM FULL.

```
VACUUM FULL test;
```

Посмотрим данные о первой странице.

```
 ctid | state  | xmin | xmax | hhu | phu | pru | all | nul
------+--------+------+------+-----+-----+-----+-----+-----
 (0,1) | normal |    0 |    0 | f   | f   | f   | f   | t
 (0,2) | normal |    0 |    0 | f   | f   | f   | f   | t
 (0,3) | normal |    0 |    0 | f   | f   | f   | f   | t
 (0,4) | normal |    0 |    0 | f   | f   | f   | f   | t
 (0,5) | normal |    0 |    0 | f   | f   | f   | f   | t
(5 rows)
```

Если таблица устойчиво не изменяется, то команда **VACUUM** или обработчик [autovacuum](https://postgrespro.ru/docs/postgrespro/current/routine-vacuuming#AUTOVACUUM) «замораживает» старые версии строк. Функция [heap_page](https://postgrespro.ru/docs/postgrespro/current/pageinspect#PAGEINSPECT-HEAP-FUNCTIONS) показывает, что для замороженных строк установлены признаки **all** и **nul**.

Команда **VACUUM** принимает значительное число [параметров](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-PARAMS), которые можно перечислять через запятую. Например, **VACUUM (FULL, ANALYZE)** выполнит полную очистку и обновление статистики планировщика. Параметр **VERBOSE** выводит подробную информацию о работе команды. Параметр **SKIP_LOCKED** позволяет [пропустить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-SKIP-LOCKED) таблицы, которые в данный момент заблокированы другими транзакциями.

Параметр **INDEX_CLEANUP** управляет [очисткой индексов](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-INDEX-CLEANUP), а параметр **PROCESS_TOAST** — [очисткой TOAST-таблиц](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-PROCESS-TOAST).

Параметр **TRUNCATE** управляет [усечением](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-TRUNCATE) пустых страниц в конце таблицы.

Параметр **PARALLEL** задает [степень параллелизма](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-PARALLEL) при очистке индексов.

Параметр **SKIP_DATABASE_STATS** позволяет [пропустить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-SKIP-DATABASE-STATS) обновление статистики базы данных, а параметр **ONLY_DATABASE_STATS** — выполнить только обновление статистики базы данных.

Параметр **BUFFER_USAGE_LIMIT** задает [размер кольцевого буфера](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-BUFFER-USAGE-LIMIT), используемого для очистки.

Параметр **ONLY_DATABASE_STATS** позволяет [выполнить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-ONLY-DATABASE-STATS) только обновление статистики базы данных.

Параметр **INCLUDE_DISABLED** позволяет [включить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-INCLUDE-DISABLED) в обработку отключенные таблицы.

Параметр **SKIP_TOAST** позволяет [пропустить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-SKIP-TOAST) TOAST-таблицы.

Параметр **SKIP_LOCKED** позволяет [пропустить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-SKIP-LOCKED) заблокированные таблицы.

Параметр **INDEX_CLEANUP** управляет [очисткой индексов](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-INDEX-CLEANUP).

Параметр **PROCESS_TOAST** позволяет [выполнить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-PROCESS-TOAST) очистку TOAST-таблиц.

Параметр **TRUNCATE** управляет [усечением](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-TRUNCATE) пустых страниц в конце таблицы.

Параметр **PARALLEL** задает [степень параллелизма](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-PARALLEL) при очистке индексов.

Параметр **SKIP_DATABASE_STATS** позволяет [пропустить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-SKIP-DATABASE-STATS) обновление статистики базы данных.

Параметр **ONLY_DATABASE_STATS** позволяет [выполнить](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-ONLY-DATABASE-STATS) только обновление статистики базы данных.

Параметр **BUFFER_USAGE_LIMIT** задает [размер кольцевого буфера](https://postgrespro.ru/docs/postgrespro/current/sql-vacuum#SQL-VACUUM-BUFFER-USAGE-LIMIT), используемого для очистки

Сбор статистики с помощью ANALYZE

Д. Есупов Кусина

Содержание

Назад к содержанию

- Команда ANALYZE
  - Обзор
  - Примеры
  - Конфигурационные параметры, влияющие на ANALYZE
  - ANALYZE для секционированных и дочерних таблиц
- Статистика, используемая планировщиком
  - Статистика по одному столбцу
  - Расширенная статистика

Статистика баз данных играет важную роль в работе планировщика. Команда ANALYZE собирает статистику о содержимом таблиц базы данных. Планировщик запросов использует эту статистику для создания эффективных планов запросов.

Команда ANALYZE

Обзор

Команда ANALYZE собирает статистику о содержимом таблицы базы данных и сохраняет результаты в системном каталоге pg_statistic. В дальнейшем планировщик запросов использует эту статистику для определения стоимости различных выполнения запросов. ANALYZE имеет следующий синтаксис:

```
ANALYZE [ <параметры> [, ...] ] [ <таблица_и_столбцы> [, ...] ]
ANALYZE [ VERBOSE ] [ <таблица_и_столбцы> [, ...] ]
```

где <параметры> может быть одним из:

```
    VERBOSE
    SKIP_LOCKED [ <логическое_значение> ]
```

и <таблица_и_столбцы> это:

```
    <имя_таблицы> [ ( <имя_столбца> [, ...] ) ]
```

Параметры команды ANALYZE	
VERBOSE	Включает отображение сообщений о ходе выполнения. Если указано VERBOSE, ANALYZE выдаст сообщения о ходе выполнения каждой таблицы и данный момент обрабатывается, даже если это не является различимо статистику
SKIP_LOCKED	Указывает, что ANALYZE не должен ждать освобождения конфигурации блокировок при обработке каждой таблицы. Если отключение не может быть заблокировано без ожидания, она пропускается. Обратите внимание, что даже с этой опцией ANALYZE может ждать при открытии индексов связанной или при получении выборки строк из них, даже данные таблиц в индексе находящемся в настоящее типов внешних таблиц. Эта опция также указывает ANALYZE пропускать все опции, если в секционированной таблице является конфигурации блокировки
<логическое_значение>	Определяет, следует ли включить или выключить выбранную опцию. Можно написать TRUE, ON или 1, чтобы включить эту опцию, и FALSE, OFF или 0, чтобы отключить ее. Если логическое значение опущено, предполагается TRUE.
<имя_таблицы>	Имя таблицы для анализа (может включать имя схемы). Если этот параметр опущен, анализируются все обычные таблицы, секционированные таблицы и материализованные представления в текущей базе данных, но не внешние таблицы. Если указана таблица является секционированной, обозначается как статистика этой секционированной таблицы, так и статистика отдельных секций
<имя_столбца>	Имя определенного столбца для анализа. По умолчанию анализируются все столбцы

Чтобы проанализировать таблицу, пользователь должен быть владельцем таблицы или суперпользователем. Владельцам баз данных разрешено анализировать все таблицы в своих базах данных, кроме каталогов. ANALYZE пропускает любые таблицы, на анализ которых у пользователя нет разрешения.

Примеры

ANALYZE VERBOSE для базы данных

Чтобы запустить ANALYZE в базе данных, сначала подключитесь к этой базе данных. Например, вы можете сделать это, используя psql:

```
\c example
```

Запустите команду ANALYZE VERBOSE:

```
ANALYZE VERBOSE;
```

Результат:

```
INFO:  analyzing "public.pgbench_accounts"
INFO:  "pgbench_accounts": scanned 30000 of 81960 pages, containing 1830000 live rows and 0 dead rows; 30000 rows in sample, 5000040 estimated total rows
INFO:  analyzing "public.pgbench_branches"
INFO:  "pgbench_branches": scanned 1 of 1 pages, containing 50 live rows and 0 dead rows; 50 rows in sample, 50 estimated total rows
INFO:  analyzing "public.pgbench_history"
INFO:  "pgbench_history": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows
INFO:  analyzing "public.pgbench_tellers"
INFO:  "pgbench_tellers": scanned 1 of 1 pages, containing 500 live rows and 0 dead rows; 500 rows in sample, 500 estimated total rows
INFO:  analyzing "pg_catalog.pg_type"
INFO:  "pg_type": scanned 15 of 15 pages, containing 409 live rows and 0 dead rows; 409 rows in sample, 409 estimated total rows
INFO:  analyzing "pg_catalog.pg_foreign_table"
INFO:  "pg_foreign_table": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows
INFO:  analyzing "pg_catalog.pg_authid"
INFO:  "pg_authid": scanned 1 of 1 pages, containing 16 live rows and 3 dead rows; 16 rows in sample, 16 estimated total rows
INFO:  analyzing "pg_catalog.pg_statistic_ext_data"
INFO:  "pg_statistic_ext_data": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated total rows
```

Поскольку используется опция VERBOSE, выходные данные содержат подробную информацию об анализируемых отношениях.

ANALYZE для таблицы

Для анализа таблицы укажите ее имя в команде:

```
ANALYZE pgbench_accounts;
```

ANALYZE для определенных столбцов

Для анализа столбцу укажите имя таблицы и имя столбца в скобках:

```
ANALYZE pgbench_accounts (bid, abalance);
```

Внешние таблицы анализируются только при явном указании. На все сборки внешних данных поддерживают ANALYZE. Если обертка не поддерживает ANALYZE, команда выдаст предупреждение и больше ничего не происходит.

Автовакуум запускает ANALYZE для таблиц, когда они впервые заполняются данными, а также если они изменяются во время обычных операций. Если автовакуум отключен для таблицы, ANALYZE также рекомендуется периодически вызывать для нее, если только дополнительные после окончания изменений в содержимом таблицы. Точная статистика помогает планировщику выбирать наиболее подходящий план запроса и повысить скорость обработки запросов. Общая стратегия для баз данных, предназначенных в основном для чтения, — запускать команды VACUUM и ANALYZE один раз в день, во время малой загрузки. Этого недостаточно, если база данных активно обновляется.

Для ANALYZE требуется только блокировка на чтение целевой таблицы, поэтому он может выполняться параллельно с другими операциями над таблицей.

Если таблица секционирована, ANALYZE собирает и хранит статистику для всей таблицы. Существующая статистика секционирована.

Каждый сервер, на котором выполняется ANALYZE, будет записывать информацию о прогрессе выполнения команды в представление pg_stat_progress_analyze. Для получения дополнительной информации об этом обратитесь к статье ANALYZE progress reporting.

Конфигурационные параметры, влияющие на ANALYZE

Статистика, собранная с помощью ANALYZE, обычно включает список наиболее распространенных значений и гистограмму, показывающую приблизительное распределение данных в каждом столбце. Один или оба из этих статистических показателей могут быть опущены, если они бесполезны или если тип данных столбца не поддерживает стандартный оператор сортировки. Например, в столбце, для которого включен тип общих наиболее распространенных значений.

Для большой таблицы ANALYZE берет случайную выборку содержимого таблицы, а не проверяет каждую строку. Это позволяет анализировать даже очень большие таблицы за короткое время. Обратите внимание, что статистика является приблизительной и будет немного меняться каждый раз при запуске ANALYZE, даже если ничего фактически содержимое таблицы не изменилось. Это приводит к небольшим изменениям в предполагаемых затратах планировщика.

Иногда ANALYZE будет принимать решение не хранить статистику для столбцов, включенных в список столбцов, для которых была указана ANALYZE. Чтобы избежать этого, укажите общий список столбцов, отдельный ANALYZE для этой таблицы.

Объем статистики можно управлять конфигурацией параметр default_statistics_target. Используйте раздел **ASPG configurations** на вкладке **Clusters** → **Кластер ASPG** → **Primary configuration** → **Primary configuration** и отредактируйте ключ default_statistics_target. Для более детального анализа столбца используйте ALTER TABLE SET STATISTICS. Также можно установить целевое значение для каждого столбца с помощью ALTER TABLE. ALTER TABLE, ALTER COLUMN, SET STATISTICS См. ALTER TABLE.

```
ALTER TABLE table1 ALTER COLUMN column1 SET STATISTICS 0;
```

Целевое значение может максимальное количество записей в списке наиболее распространенных значений и максимальное количество интервалов в гистограмме. Целевое значение по умолчанию - 100, но если значение этого умеющаяся, чтобы найти баланс между скоростью планировщика и временем, необходимым для запуска ANALYZE. Точное значение можно установить default_statistics_target. Нулевое целевого значения данных 0 отключает сбор статистики для этого столбца. Это может быть полезно для столбца, который никогда не используется как часть выражений WHERE, GROUP BY или ORDER BY, поскольку планировщик не использует статистику по таким столбцам.

Самый большой целевой показатель статистики среди столбцов определяет количество строк таблицы, выбранных для подготовки статистики. Это увеличение приводит к пропорциональному увеличению времени и места на диске, необходимого для выполнения ANALYZE.

Вторая значение, оценивающее с помощью ANALYZE, — это количество различных значений, которые присутствуют в каждом столбце. Поскольку количество строк выборки ограничено, оценка может быть весьма неточной для столбцов с большим количеством различных значений. Если это приводит к неправильным планам запросов, более точное значение можно определить вручную и использовать с помощью команды ALTER TABLE. ALTER COLUMN, SET (n_distinct =); VACUUM ANALYZE.

```
ALTER TABLE table2 ALTER COLUMN column2 SET (n_distinct = 500);
```

ANALYZE для секционированных и дочерних таблиц

Если у таблицы есть наследующие дочерние элементы, ANALYZE собирает две набора статистики: один только по строкам родительской таблицы, а второй включает строки родительской таблицы и всех ее дочерних элементов. Планировщик использует второй набор статистики при обработке запросов, оперирующих всем деревом наследования, поэтому для таблицы только по родительским элементам нет необходимости. Последнего автоматически учитывает только вставки или обновления дочерней таблицы при принятии решения об автоматическом запуске ANALYZE для этой таблицы. Когда таблицы редко обновляются, статистика будет устаревшей, если не запускать ANALYZE вручную.

Для секционированных таблиц, ANALYZE собирает статистику, выборки строки из всех секций. Кроме того, он будет рекурсивно посещать каждую секцию и обновлять ее статистику. ANALYZE также собирает и хранит статистику для данных для многоуровневых секционированных. ANALYZE — собирает статистику по секционированной таблице (без данных из ее секций), так как при секционированном они пусто.

Автовакуум не обрабатывает секционированные таблицы и родительские таблицы, если были изменены только дочерние таблицы или секции. Обычно необходимо периодически запускать ANALYZE для родительской таблицы, чтобы поддерживать актуальную статистику этих корневых таблиц.

Если какие-либо дочерние таблицы или секции являются внешними таблицами и их обертки внешних данных не поддерживают ANALYZE, эти таблицы игнорируются при сборе статистики.

Статистика, используемая планировщиком

Как упоминалось выше, планировщику запросов необходимо оценить количество строк, полученных запросом, чтобы составить эффективный план запроса. В этом разделе предоставлен краткий обзор статистики, которую системе используют для этих оценок.

Статистика по одному столбцу

Одним из компонентов статистики является общее количество записей и количество дисковых блоков, занимаемых каждой таблицей и индексом. Таблица pg_class содержит эту информацию в таблице relpages и reltuples.

```
SELECT relname, relkind, reltuples, relpages
FROM pg_class
WHERE relname LIKE 'pgbench_accounts';
```

Результат:

```
     relname      | relkind | reltuples | relpages
------------------+---------+-----------+----------
 pgbench_accounts | r       |  5.000e+06 |    81960
 pgbench_accounts_pkey | i     |  5.000e+06 |    13711
```

Таблица pgbench_accounts и ее индекс содержат 5.000e+06 строку, но индекс меньше таблицы.

Это соображение учитывается планировщиком, но планировщик основан с помощью VACUUM ANALYZE и несколько команд DDL, таких как CREATE INDEX. Операция VACUUM или ANALYZE могут не сканировать всю таблицу, а вычислить приближение reltuples на основе тех частей таблицы, которые они уже сканировали, что приводит к приближению. Планировщик масштабирует значение из pg_class, чтобы оно соответствовало текущему физическому размеру таблицы, и получает более точное приближение.

Большинство запросов извлекают только часть строк таблицы, ограничивающих выражением WHERE. Планировщику необходимо оценить избирательность выражений WHERE, то есть долю строк, соответствующих каждому условию в WHERE. Для этого планировщик использует информацию, извлеченную в системном каталоге pg_statistic. Команды ANALYZE и VACUUM ANALYZE обновляют записи в pg_statistic, но они всегда приблизительны, даже сразу после обновления.

Вместо того чтобы использовать непосредственно pg_statistic, где изучение статистических строк обратиться к представлению pg_stats. pg_stats доступно для чтения всем, когда pg_statistic — только суперпользователю. Это не позволяет непривилегированным пользователям узнать что либо из статистики о содержимом таблиц других пользователей. Представление pg_stats организовано отображением только строки с информацией о тех таблицах, которые текущий пользователь может прочитать.

Можно выполнить следующий запрос к pg_stats, где terminals — это имя таблицы:

```
SELECT attname, n_distinct,
array_to_string(most_common_vals, E'\n') as most_common_vals
FROM pg_stats
WHERE tablename = 'terminals';
```

Результат:

```
  attname    | n_distinct |  most_common_vals
-------------+------------+---------------------
 terminal_code |         -1 |
 timezone    |  -0.1454154 | Europe/Moscow      +
             |            | Asia/Yekaterinburg+
             |            | Asia/Krasnoyarsk   +
             |            | Asia/Irkutsk       +
             |            | Asia/Vanutsk       +
             |            | Europe/Samara      +
             |            | Asia/Vladivostok   +
             |            | Asia/Novokuznetsk  +
             |            | Europe/Volgograd
 terminal_name |         -1 |
 city        |  -0.9711536 | Moscow             +
             |            | Ulyanovsk
```

где:

- attname – имя столбца, описываемого этой строкой
- n_distinct – если значение больше нуля, оно представляет собой предполагаемое количество различных значений в столбце. Если это число меньше нуля, это модуль представляет собой количество различных значений, разделенное на количество строк. Отрицательная форма используется, когда ANALYZE подозревает, что количество различных значений будет увеличиваться по мере роста таблицы, положительная форма — когда предполагается, что столбец имеет фиксированное количество возможных значений. -1 указывает на уникальный столбец, в котором количество различных равно количеству строк.
- most_common_vals – список наиболее распространенных значений в столбце. Null (пусто в приведенном выше примере), если никакие значения не являются более распространенными, чем другие.

Расширенная статистика

Часто можно увидеть медленные запросы с плохими планами выполнения, поскольку несколько столбцов, используемых в запросе, взаимосвязаны. Планировщик обычно предполагает, что все условия, проверяемые условий независимы друг от друга, поэтому они не учитывают зависимости между такими столбцами. Однако ASPG PostgreSQL позволяет вычислять многомерную статистику, собирающую такую информацию.

Поскольку количество возможных комбинаций столбцов очень велико, автоматически вычислять многомерную статистику нерационально. Вместо этого ASPG PostgreSQL позволяет создавать **расширенные объекты статистики** для получения статистики по выбранным наборам столбцов.

Объекты статистики создаются с помощью команды CREATE STATISTICS. При создании такого объекта просто добавляется запись, выражающий необходимую статистику. Команда ANALYZE обрабатывает таблицы статистической данных, которые можно просмотреть в каталоге pg_statistic_ext_data.

Самый простой вид расширенной статистики — отслеживание функциональных зависимостей. Эта концепция используется в определениях нормальных форм базы данных. Столбец B функционально зависит от столбцов A, если значение значения B достаточно для определения значения A, то есть не существует двух строк с одинаковыми значениями A, но разными значениями B.

Наличие функциональных зависимостей напрямую влияет на точность оценок в тех или иных запросов. Чтобы собрать информацию о функциональных зависимостях, ANALYZE может собирать измерения зависимости между столбцами. Данные оцениваются только для тех групп столбцов, которые указаны в объекте расширенной статистики, определенного с помощью опции dependencies. Рекомендуется создавать статистику зависимостей только для групп столбцов, сильно связанных между собой, например столбцы корреспонденции. Чтобы избежать ненужных накладных расходов при анализе, такая статистика создается только при последующем запросе.

Код ниже создает объект расширенной статистики stat1 для сбора статистики функциональных зависимостей в столбцах terminal_name и city таблицы terminals:

```
CREATE STATISTICS stat1 (dependencies) ON terminal_name, city FROM terminals;

ANALYZE terminals;
```

Проверьте результат:

```
SELECT stxname, stxkeys, stxddependencies
FROM pg_statistic_ext JOIN pg_statistic_ext_data ON (oid = stxoid)
WHERE stxname = 'stat1';
```

Результат:

```
 stxname | stxkeys |             stxddependencies
---------+---------+------------------------------------------
 stat1   | 2 3     | {"2 => 3": 1.000000, "3 => 2": 0.847619}
```

Столбцы 2, terminal_name полностью определяет столбец 3, city – коэффициент равен 1.0, тогда как city определяет terminal_name только примерно в 87% всех случаев, то есть существуют города, в которых находятся несколько терминалов.

Дополнительную информацию и примеры многовариантной статистики можно найти в статье How the planner uses statistics.

Логирование

Евгения Кузина

Содержание

Назад к содержанию

Настройка параметров логирования

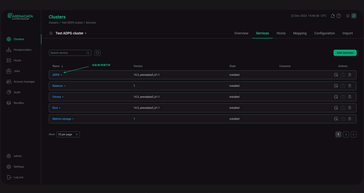
Просмотр лог-файлов

В этой статье описывается как настроить логирование и просмотреть лог-файлы сервиса ADPG.

Настройка параметров логирования

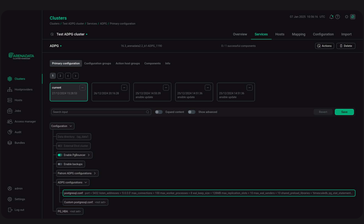
Используйте ADCM, чтобы настроить параметры логирования.

Откройте страницу ***Clusters*** и кликните кластер ADPG. Перейдите на вкладку ***Services*** и кликните сервис ***ADPG***.



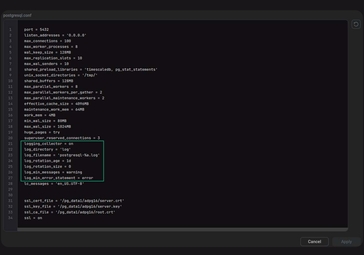
Сервисы кластера ADPG

Откроется вкладка ***Primary configuration*** сервиса ADPG. Раскройте ноду ***ADPG configurations*** и кликните ***postgresql.conf***.



Параметры сервиса ADPG

Откроется текстовое поле с параметрами конфигурации. В этом поле можно указать необходимые параметры логирования.



Поле "postgresql.conf"

Наиболее часто используемые настройки логирования перечислены в таблице.

Параметры логирования сервиса ADPG

Наименование параметра	Описание	Значение по умолчанию
logging_collector	Включает сборщик сообщений логирования (*logging collector*). Сборщик сообщений логирования — это фоновый процесс, который перехватывает сообщения логирования, отправленные в stderr, и перенаправляет их в файлы логов	Сборщик сообщений логирования включен (`true`)
log_directory	Определяет каталог, в который записываются лог-файлы. Может содержать абсолютный путь или путь относительно каталога с данными кластера ADPG	log (абсолютный путь — /pg_data1/adpg16/log)
log_filename	Задает шаблон имени лог-файла, может содержать спецификаторы % strftime для включения времени и даты создания в имя файла. Если спецификаторы % не заданы, используйте утилиты ротации лог-файлов, чтобы избежать переполнения диска	postgresql-%a.log
log_rotation_age	Определяет максимальный период использования лог-файла, после которого создается новый лог-файл. Если значение задано без единиц измерения, оно берется в минутах. Чтобы отключить создание лог-файлов по времени, установите значение `0`	1d
log_rotation_size	Определяет максимальный размер лог-файла. После того как лог-файл достигает указанного размера, создается новый лог-файл. Если значение задано без единиц измерения, оно берется в килобайтах. Чтобы отключить создание лог-файлов при превышении определенного размера, установите значение `0`	0
log_min_messages	Определяет минимальный уровень важности сообщений, которые записываются в лог-файл. Допустимые значения: `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal` и `panic` (см. таблицу Уровни важности лог-сообщений). Сообщения с заданной важностью и более важные записываются в лог-файл. Например, если вы установили значение `warning`, лог-файл будет включать сообщения с важностью `warning`, `error`, `log`, `fatal` и `panic`.	warning
log_min_error_statement:	Определяет какие SQL-операторы, завершившиеся ошибкой, записываются в лог-файл. Допустимые значения: `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal` и `panic` (см. таблицу Уровни важности лог-сообщений). SQL-оператор будет записан в лог-файл, если он завершится ошибкой с указанным уровнем важности или выше. Чтобы отключить логирование SQL-операторов, установите значение `panic`	error

Ниже перечислены уровни важности сообщений, которые могут быть записаны в лог-файл.

Уровни важности лог-сообщений

Значение	Описание
Debug1..Debug5	Детальная информация для разработчиков. Чем больше номер, тем подробнее информация
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	Сообщение об ошибке, из-за которой прервана текущая команда
Log	Информация, полезная для администраторов. Например, выполнение контрольных точек
Fatal	Сообщение об ошибке, из-за которой прервана текущая сессия
Panic	Сообщение об ошибке, из-за которой прерваны все сессии

Если вы изменили настройки логирования, выполните действие Reconfigure & Restart, чтобы применить изменения.

Все настройки, перечисленные выше — стандартные параметры конфигурации PostgreSQL сервера. Вы можете найти дополнительную информацию на официальном сайте: Error Reporting and Logging.

ВНИМАНИЕ

Когда сервис ADPG выполняет Reconfigure & Restart, файл *postgresql.conf* перезаписывается с настройками, указанными в ***Services → ADPG → Configuration***. Если вы изменили файл *postgresql.conf* напрямую, эти изменения будут потеряны.

Просмотр лог-файлов

Вы можете найти лог-файлы на хосте, на котором установлен ADPG.

1. Подключитесь к хосту (например, по SSH) и выполните команду ls, передав путь к каталогу с лог-файлами как параметр (в этом примере путь – */pg_data1/adpg16/log*):

```
$ sudo ls -la /pg_data1/adpg16/log
```

Вы получите список лог-файлов в каталоге:

```
total 10708
drwx------  2 postgres postgres     136 May 26 00:00 .
drwx------ 20 postgres postgres    4096 May 26 08:54 ..
-rw-------  1 postgres postgres    1009 May 20 13:31 postgresql-Fri.log
-rw-------  1 postgres postgres     594 May 23 06:30 postgresql-Mon.log
-rw-------  1 postgres postgres 3803094 May 26 10:04 postgresql-Thu.log
-rw-------  1 postgres postgres     988 May 24 07:37 postgresql-Tue.log
-rw-------  1 postgres postgres 3343041 May 25 23:59 postgresql-Wed.log
```

2. Вы можете использовать команду tail, чтобы просмотреть лог-файл:

```
$ sudo tail /pg_data1/adpg16/log/postgresql-Mon.log
```

Результат:

```
2022-05-23 06:30:40.804 GMT [951] LOG:  starting PostgreSQL 14.2 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red
Hat 4.8.5-44), 64-bit
2022-05-23 06:30:40.804 GMT [951] LOG:  listening on IPv6 address "::1", port 5432
2022-05-23 06:30:40.804 GMT [951] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2022-05-23 06:30:40.813 GMT [951] LOG:  listening on Unix socket "/tmp/.s.PGSQL.5432"
2022-05-23 06:30:40.845 GMT [1079] LOG:  database system was shut down at 2022-05-20 13:31:40 GMT
2022-05-23 06:30:40.972 GMT [951] LOG:  database system is ready to accept connections
```

Оптимизация производительности

Антон Ларин

ADPG

Назад к содержанию

Содержание

Оптимизация потребления ресурсов
- Настройки памяти
- Настройки, связанные с операциями записи вывода
Мониторинг загрузки диска
- Использование SQL-функции для определения размеров объектов базы данных
- Просмотр системных каталогов
- Использование модуля oid2name
Наполнение базы данных
- Отключение автофиксации транзакций
- Использование COPY
- Удаление индексов
- Удаление ограничений внешних ключей
- Увеличение maintenance_work_mem
- Увеличение max_wal_size
- Отключение архивации WAL и создания бэкапа
- Выполнение ANALYZE по завершении
- Улучшение производительности pg_dump

Оптимизация потребления ресурсов

Настройки памяти

В каталоге initdb создаются файлы postgresql.conf, pg_hba.conf и pg_ident.conf. Первый из них содержит настройки сервера, второй управляет доступом, третий используется для сопоставления внешних имен пользователей именам пользователей PostgreSQL.

Основные параметры	Описание	Рекомендованное значение
effective_cache_size	Оценка объема памяти, доступного для кэширования данных одним запросом. Учитывается планировщиком при выборе между последовательным и индексным сканированием. Не влияет на реальное выделение памяти.	2/3 RAM
shared_buffers	Объем памяти, выделяемый для общего кэша страниц данных. Основной параметр производительности. Слишком большое значение может вызвать конкуренцию за память с ОС.	8MB-8
temp_buffers	Максимальный объем памяти, выделяемый для временных таблиц в каждой сессии. Временные таблицы не журналируются.	256 MB
work_mem	Объем памяти, используемый для операций сортировки и хеширования перед записью во временные файлы. Выделяется для каждой операции, а не для всего запроса.	16MB-32
maintenance_work_mem	Объем памяти для операций обслуживания: VACUUM, CREATE INDEX и ALTER TABLE ADD FOREIGN KEY.	1GB-2
autovacuum_work_mem	Объем памяти для автоматического VACUUM. Если не задан, используется maintenance_work_mem.	1GB-2

Все основные параметры, кроме autovacuum_work_mem и max_stack_depth, можно изменить в разделе **ADPG configurations** раздела **Configuration** на странице **Clusters** → **Cluster Actions** → **ADPG** → **Primary configurations**.

Настройки, связанные с операциями записи вывода

В каталоге wal хранятся журналы транзакций. Они используются для восстановления данных после сбоев и для репликации. Настройки журналирования влияют на надежность и производительность записи.

Основные параметры	Описание
fsync	Принудительная синхронизация данных с диском после каждой операции записи. Гарантирует сохранность данных при сбое ОС или питания. Отключение ускоряет запись, но может привести к повреждению данных. Рекомендуется оставить включенным.
synchronous_commit	Определяет, будет ли клиент ожидать записи данных на диск перед получением подтверждения о фиксации транзакции. Значение off ускоряет работу, но при сбое последние транзакции могут быть потеряны.
checkpoint_completion_target	Доля времени между контрольными точками, отведенная на запись данных. Значение 0.9 означает, что запись будет распределена равномерно, чтобы избежать резких скачков нагрузки.
effective_io_concurrency	Количество параллельных операций ввода-вывода, которые может выполнять подсистема хранения. Для SSD можно установить 200 и выше, для HDD — 2–4.
random_page_cost	Оценка стоимости произвольного чтения страницы относительно последовательного. Для SSD рекомендуется 1.1, для HDD — 4.0.

Использование SQL-функции для определения размеров объектов базы данных

Начиная с версии 9.0 в PostgreSQL появилась возможность получать информацию о размерах объектов базы данных. Для этого используется функция pg_relation_size(), которая возвращает размер указанного объекта в байтах. Вы можете использовать имя объекта или его OID (идентификатор объекта):

```
SELECT pg_relation_size('updated_accounts');
```

Результат:

```
 pg_relation_size
------------------
            57344
```

Функция pg_size_pretty() преобразует размер в удобную для чтения форму с единицами измерения (байты, КБ, МБ, ГБ или ТБ):

```
SELECT pg_size_pretty(pg_relation_size('updated_accounts'));
```

Результат:

```
 pg_size_pretty
----------------
 56 kB
```

Можно получить размер объекта с помощью функции pg_total_relation_size(), которая также учитывает индексы и данные TOAST:

```
SELECT pg_size_pretty (pg_database_size('events'));
```

Результат:

```
 pg_size_pretty
----------------
 7821 MB
```

Просмотр системных каталогов

Для просмотра системных каталогов можно использовать psql. Для его правильного отображения часто требуется предварительно включить расширенный режим вывода с помощью команды \x. Эта команда переключает режим отображения:

```
SELECT pg_relation_filepath(oid), relpages FROM pg_class WHERE relname = 'updated_accounts';
```

Результат:

```
-[ RECORD 1 ]+-----
pg_relation_filepath | base/16384/17334
relpages             | 7
```

Чтобы получить список всех баз данных, можно обратиться к системному каталогу pg_database. Следующий запрос выведет имена баз данных и их OID:

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC LIMIT 5;
```

Результат:

```
       relname       | relpages
---------------------+----------
 pg_depend_reference |      113
 pg_depend           |       57
 pg_attribute        |       55
 pg_proc             |       54
 pg_description      |       46
```

Чтобы найти самые большие таблицы и индексы, используйте следующий запрос:

```
SELECT relname, relpages
FROM pg_class
ORDER BY relpages DESC LIMIT 5;
```

Результат:

```
       relname       | relpages
---------------------+----------
 pg_depend_reference |      113
 pg_depend           |       57
 pg_attribute        |       55
 pg_proc             |       54
 pg_description      |       46
```

Запрос ниже отображает размер индекса для таблицы updated_accounts:

```
SELECT pg_relation_oid(oid)
FROM pg_class
WHERE relname = 'updated_accounts';
```

Результат:

```
 pg_relation_oid
-----------------
           16384
```

Использование модуля oid2name

Модуль oid2name позволяет получить имя объекта по его идентификатору (OID). Сначала создайте каталог для примеров — временный каталог /tmp подойдет:

```
mkdir /tmp/pg_oid_test
```

Теперь создайте тестовую базу данных и несколько таблиц для проверки:

```
CREATE DATABASE oid_test;
\c oid_test
CREATE TABLE test1 (id integer);
CREATE TABLE test2 (id integer);
```

Запустите команду oid2name без параметров, чтобы увидеть список доступных опций:

```
oid2name
```

Результат:

```
All databases:
    Oid  Database Name  Tablespace
----------------------------------
  16385        oid_test  pg_default
  16384        postgres  pg_default
  16383       template1  pg_default
  16382       template0  pg_default
```

Попробуем найти расположение файла. Функция pg_relation_filepath() возвращает путь к файлу относительно каталога данных PostgreSQL:

```
SELECT pg_relation_filepath('test1');
```

Результат:

```
 pg_relation_filepath
----------------------
 base/16385/16386
```

Теперь используем oid2name с параметром -f, чтобы найти таблицу по OID файла:

```
$ oid2name -f 16386
```

Результат:

```
From database "oid_test":
  Filenode  Table Name
----------------------
     16386        test1
```

Для списка tablespaces используется параметр -s. Он покажет все табличные пространства и их OID:

```
$ oid2name -s
```

Результат:

```
All tablespaces:
     Oid  Tablespace Name
-------------------------
    1663        pg_default
    1664         pg_global
```

Параметр -x покажет и имена баз данных, и имена табличных пространств. Для этого укажите конкретную базу данных:

```
$ oid2name -x -d oid_test
```

Результат:

```
From database "oid_test":
  Filenode  Table Name  Tablespace
-----------------------------------
     16386        test1  pg_default
```

Чтобы найти все файлы, принадлежащие конкретной базе данных, используйте параметр -d. Укажите имя базы данных:

```
$ oid2name -d oid_test
```

Результат:

```
From database "oid_test":
  Filenode  Table Name
----------------------
     16386        test1
     16387        test2
```

Чтобы найти все файлы, принадлежащие конкретной таблице, используйте параметр -t. Укажите имя таблицы:

```
$ oid2name -d oid_test -t test1
```

Результат:

```
From database "oid_test":
  Filenode  Table Name
----------------------
     16386        test1
```

Параметр -q позволяет скрыть заголовки. Это удобно при автоматической обработке вывода:

```
$ oid2name -q -d oid_test -t test1
```

Результат:

```
     16386        test1
```

Параметр -o покажет OID объекта в дополнение к имени файла:

```
$ oid2name -o -d oid_test -t test1
```

Результат:

```
From database "oid_test":
  Filenode  Table Name  OID
---------------------------
     16386        test1  16386
```

Параметр -S позволяет работать с системными объектами. Без него они не отображаются:

```
$ oid2name -S -d oid_test
```

Результат:

```
From database "oid_test":
  Filenode  Table Name
----------------------
     16386        test1
     16387        test2
     16388        test3
     16389        test4
     16390        test5
     16391        test6
     16392        test7
     16393        test8
     16394        test9
     16395        test10
```

Параметр -i исключает индексы из вывода. Без него они показываются:

```
$ oid2name -i -d oid_test
```

Результат:

```
From database "oid_test":
  Filenode  Table Name
----------------------
     16386        test1
     16387        test2
     16388        test3
     16389        test4
     16390        test5
     16391        test6
     16392        test7
     16393        test8
     16394        test9
     16395        test10
```

Параметр -T выводит тип объекта. Например, table или index:

```
$ oid2name -T -d oid_test
```

Результат:

```
From database "oid_test":
  Filenode  Table Name  Object Type
---------------------------------
     16386        test1  table
     16387        test2  table
     16388        test3  table
     16389        test4  table
     16390        test5  table
     16391        test6  table
     16392        test7  table
     16393        test8  table
     16394        test9  table
     16395        test10  table
```

Параметр -P покажет путь к файлу относительно каталога данных PostgreSQL:

```
$ oid2name -P -d oid_test
```

Результат:

```
From database "oid_test":
  Filenode  Table Name  Path
---------------------------
     16386        test1  base/16385/16386
     16387        test2  base/16385/16387
     16388        test3  base/16385/16388
     16389        test4  base/16385/16389
     16390        test5  base/16385/16390
     16391        test6  base/16385/16391
     16392        test7  base/16385/16392
     16393        test8  base/16385/16393
     16394        test9  base/16385/16394
     16395        test10  base/16385/16395
```

Параметр -p позволяет указать порт, если сервер работает не на стандартном. Параметр -U задает имя пользователя для подключения. Параметр -H задает хост сервера. Параметр -P запрашивает пароль перед подключением. Параметр -V выводит версию утилиты.

Наполнение базы данных

Этот раздел содержит рекомендации о том, как быстро добавлять большие объемы данных или в процессе их использования.

Отключение автофиксации транзакций

Если вы используете psql, отключите автофиксацию транзакций командой \set AUTOCOMMIT off. В остальных SQL-API это включено по умолчанию. Команда BEGIN начинает транзакцию, а COMMIT фиксирует ее. Все операции между ними будут выполнены как единое целое.

```
BEGIN;
INSERT INTO users (id, name) VALUES (1, 'Alice');
INSERT INTO users (id, name) VALUES (2, 'Bob');
COMMIT;
```

Использование COPY

Команда COPY загружает данные из файла в таблицу. Она работает быстрее, чем серия INSERT. Синтаксис COPY имя_таблицы FROM 'путь_к_файлу' WITH (FORMAT csv, HEADER true);. Параметр FORMAT задает формат файла. Параметр HEADER указывает, что первая строка содержит заголовки столбцов. Если файл не имеет заголовков, этот параметр можно опустить.

```
COPY users FROM '/path/to/data.csv' WITH (FORMAT csv, HEADER true);
```

Если файл находится на клиентской машине, используйте \copy вместо COPY. Синтаксис команды \copy такой же, как у COPY. Она читает файл с клиента и передает данные серверу.

```
\copy users FROM 'data.csv' WITH (FORMAT csv, HEADER true);
```

Удаление индексов

Перед массовой загрузкой данных можно временно удалить индексы. Это ускорит операции INSERT, так как индексы не будут обновляться при каждой вставке. После загрузки индексы можно создать заново. Команда DROP INDEX имя_индекса; удаляет индекс. Команда CREATE INDEX имя_индекса ON имя_таблицы (столбец1, столбец2); создает его заново.

```
DROP INDEX idx_users_email;
-- загрузка данных
CREATE INDEX idx_users_email ON users (email);
```

Удаление ограничений внешних ключей

Ограничения внешних ключей также замедляют вставку. Их можно временно удалить. Команда ALTER TABLE имя_таблицы DROP CONSTRAINT имя_ограничения; удаляет ограничение. Команда ALTER TABLE имя_таблицы ADD CONSTRAINT имя_ограничения FOREIGN KEY (столбец) REFERENCES другая_таблица (столбец); восстанавливает его.

```
ALTER TABLE orders DROP CONSTRAINT fk_orders_user_id;
-- загрузка данных
ALTER TABLE orders ADD CONSTRAINT fk_orders_user_id FOREIGN KEY (user_id) REFERENCES users (id);
```

Увеличение maintenance_work_mem

Параметр maintenance_work_mem задает объем памяти для операций обслуживания: VACUUM, CREATE INDEX и ALTER TABLE ADD FOREIGN KEY. Значение по умолчанию — 64 МБ. Для больших таблиц его стоит увеличить. Это можно сделать в разделе **ADPG configurations** раздела **Configuration** кластера **Clusters** → **Cluster Actions** → **ADPG** → **Primary configurations** или выполнить команду SET maintenance_work_mem = '1GB'; в текущей сессии.

```
SET maintenance_work_mem = '1GB';
CREATE INDEX idx_users_email ON users (email);
```

Увеличение max_wal_size

Параметр max_wal_size задает максимальный размер WAL-файлов между контрольными точками. По умолчанию — 1 ГБ. Увеличение значения уменьшает частоту контрольных точек и снижает нагрузку на диск. Изменить параметр можно в разделе **ADPG configurations** раздела **Configuration** кластера **Clusters** → **Cluster Actions** → **ADPG** → **Primary configurations** или выполнив команду ALTER SYSTEM SET max_wal_size = '4GB'; и последующий SELECT pg_reload_conf();.

Отключение архивации WAL и создания бэкапа

Внимание!

Отключение резервного копирования WAL может привести к потере данных при сбое. Используйте эту настройку только для временных задач, таких как массовая загрузка данных.

Если вы выполняете массовую загрузку данных, можно временно отключить архивацию WAL и создание бэкапов. Для этого установите параметр wal_level в значение minimal. Также установите archive_mode в off и max_wal_senders в 0. Эти настройки можно изменить в разделе **Primary configurations** кластера **ADPG Clusters** → **Cluster Actions** → **ADPG** → **Primary configurations**.

Выполнение ANALYZE по завершении

После загрузки данных выполните команду ANALYZE имя_таблицы;. Она обновит статистику, которую использует планировщик запросов. Без актуальной статистики PostgreSQL может выбрать неэффективный план выполнения запроса. Команда ANALYZE без имени таблицы обновит статистику для всей базы данных.

Улучшение производительности pg_dump

Если вы используете pg_dump для создания резервных копий, можно ускорить процесс восстановления. Для этого используйте параметры --jobs и --format=directory. Параметр --jobs задает количество параллельных потоков. Параметр --format=directory сохраняет данные в формате, который поддерживает параллельное восстановление.

```
pg_dump --format=directory --jobs=4 --file=/backup/mydb_dump mydb
```

Для восстановления используйте pg_restore с теми же параметрами --jobs и --format=directory.

```
pg_restore --format=directory --jobs=4 --dbname=mydb /backup/mydb_dump
```

Настройка использования huge pages в Linux

Евгений Кузина

AD.PG

Содержание

Назад к содержанию

Включение использования huge pages

Расчет необходимого количества huge pages на работающем кластере

Включение использования huge pages

Применение huge pages — больших страниц памяти — снижает накладные расходы, когда ADPG/PostgreSQL использует большие смежные фрагменты памяти. В частности, это происходит, если параметр shared_buffers имеет большое значение.

В ADPG значение по умолчанию параметра конфигурации `huge_pages` — `try`, но обычно поддержка huge pages выключена в операционной системе. Чтобы эффективно использовать huge pages, необходимо включить их поддержку на стороне операционной системы. Для этого выполните следующие шаги:

1. Убедитесь, что ядро Linux имеет настройки `CONFIG_HUGETLBFS=y` и `CONFIG_HUGETLB_PAGE=y`, необходимые для использования huge pages. Выполните следующую команду:

```
$ grep HUGETLB /boot/config-$(uname -r)
```

   Пример результата выполнения команды:

```
CONFIG_ARCH_WANT_GENERAL_HUGETLB=y
CONFIG_CGROUP_HUGETLB=y
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
```

2. Проверьте, что в настоящее время huge pages не используются. Можно получить информацию из подсистемы виртуальной памяти:

```
$ cat /proc/sys/vm/nr_hugepages
```

   Вывод:

```
0
```

   Кроме того, информацию о huge pages можно получить из /proc/meminfo — файла, который содержит статистику использования памяти:

```
$ grep ^HugePages /proc/meminfo
```

   Вывод:

```
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
```

3. Убедитесь, что параметр ADPG `huge_pages` имеет значение `try`, например, это можно сделать с помощью `psql`.

```
SHOW huge_pages;
```

   Результат:

```
 huge_pages
------------
 try
```

4. Определите количество huge pages для резервирования.
   Остановите кластер:

```
$ sudo systemctl stop adpg14-patroni.service
```

   Получите размер huge pages, установленный в системе:

```
$ grep ^Hugepagesize /proc/meminfo
```

```
Hugepagesize:       2048 kB
```

   Вычисляемый runtime-параметр PostgreSQL `shared_memory_size_in_huge_pages` определяет количество требуемых huge pages. Значение этого параметра можно получить с помощью команды `postgres`, где `/pg_data1/adpg14` — путь к *postgresql.conf*, который может быть заменен на переменную окружения `$PGDATA`, если она установлена.

```
$ postgres -D /pg_data1/adpg14 -C shared_memory_size_in_huge_pages
```

   Вывод:

```
73
```

   В этом примере требуется не менее 73 huge pages. Большее значение можно использовать, если и другим программам тоже необходимы huge pages.

   Команда ниже отображает подкаталоги в формате `hugepages-<размер>kB`, где `<размер>` — размер huge pages, поддерживаемых комбинацией ядра и процессора.

```
$ ls /sys/kernel/mm/hugepages
```

   Вывод:

```
hugepages-1048576kB  hugepages-2048kB
```

   Размер страницы huge pages по умолчанию составляет 2 МБ, но также можно явно запросить либо 2 МБ, либо 1 ГБ с помощью конфигурационного параметра huge_page_size. Для установки этого параметра используйте раздел **ADPG configurations** на странице ***Primary configuration*** сервиса ADPG (см. «Конфигурационные параметры»). Количество страниц, рассчитанное с помощью `shared_memory_size_in_huge_pages`, будет адаптировано к размеру страницы. Чтобы определить оптимальный размер страницы huge pages, протестируйте разные размеры страниц со своей базой данных.

5. Установите параметр ядра ОС `vm.nr_hugepages`, который определяет количество huge pages для резервирования. Необходимо выполнить следующую команду от имени пользователя `root`:

```
$ sudo su -

$ echo 'vm.nr_hugepages = 73' >> /etc/sysctl.d/30-postgresql.conf
```

   Загрузите параметры ядра из всех файлов:

```
$ sysctl -p --system
```

   Проверьте результат, используя следующие команды:

```
$ grep ^HugePages /proc/meminfo
```

   Вывод:

```
HugePages_Total:      73
HugePages_Free:       73
HugePages_Rsvd:        0
HugePages_Surp:        0
```

```
$ sudo sysctl -a | grep 'huge+page'
```

   Вывод должен содержать указанное значение `vm.nr_hugepages`:

```
...
vm.nr_hugepages = 73
...
```

6. Запустите кластер:

```
$ sudo systemctl start adpg14-patroni.service
```

   Посмотрите, как изменилась статистика:

```
$ grep ^HugePages /proc/meminfo
```

   Вывод:

```
HugePages_Total:      73
HugePages_Free:       63
HugePages_Rsvd:       63
HugePages_Surp:        0
```

Расчет необходимого количества huge pages на работающем кластере

Оптимальный размер и количество huge pages определяется в результате тестирования, и если вам не подошел первый вариант расчета, приведенный выше, можно попробовать рассчитать количество huge pages на запущенном кластере. Полученные результаты также необходимо протестировать.

1. Определите pid процесса postmaster:

```
$ sudo head -1 /pg_data1/adpg14/postmaster.pid
```

   Вывод:

```
895
```

2. Получите значение пикового использования памяти процессом postmaster с помощью pid:

```
$ grep ^VmPeak /proc/895/status
```

   Вывод:

```
VmPeak:   403440 kB
```

3. Рассчитайте количество huge pages как значение `VmPeak`, разделенное на размер страницы huge pages плюс `1` — `403440/2048+1`. В этом примере необходимое количество huge pages — 197.

Поскольку этот метод не требует остановки кластера, вам необходимо перезапустить кластер вместо запуска после указания параметра `vm.nr_hugepages` (шаг 6, описанный выше). Нажмите `y` и `Enter` для подтверждения операции, когда это необходимо:

```
$ /usr/lib/adpg14/usr/bin/patronictl -c /etc/adpg14-patroni/adpg14-patroni.yml \
   restart a80befc-755f-4ccb-8414-989f8ddda0e7 test_adpg_cluster_eek1-adpg-adpg3
```

Где:

- `a80befc-755f-4ccb-8414-989f8ddda0e7` – UUID кластера. Его можно найти в пользовательском интерфейсе ADCM на вкладке кластера **Configuration**
- `test_adpg_cluster_eek1-adpg-adpg3` – имя текущей ноды в Patroni. В пользовательском интерфейсе ADCM это имя можно найти на вкладке **Info** сервиса ADPG. Также содержит это имя вывод команды Patroni `list` `/usr/lib/adpg14/usr/bin/patronictl -c /etc/adpg14-patroni/adpg14-patroni.yml list`

ПРИМЕЧАНИЕ

Huge pages настраиваются индивидуально для каждого хоста. Повторите настройку huge pages на всех хостах, где это необходимо.

Структура каталога данных

Евгения Кузина

Содержание

Назад к содержанию

Содержимое каталога данных кластера

Файлы отношений

Файлы табличных пространств

Получение пути к файлу отношения

В этой статье описывается структура директории с данными кластера ADPG.

ADPG сохраняет файлы данных в каталоге, указанном в параметре конфигурации ***Data directory***. Значение по умолчанию — `/pg_data1`. Можно изменить это значение во время установки кластера ADPG, до установки ADPG-сервиса.

Содержимое каталога данных кластера

В каталоге, указанном в ***Data directory***, находится подкаталог для текущего кластера. Для ADPG v16.x.x это adpg16. Его содержимое приведено в таблице ниже.

Элемент	Описание
PG_VERSION	Файл, содержащий номер мажорной версии ADPG/PostgreSQL
base	Каталог, содержащий подкаталоги для каждой базы данных
current_logfiles	Файл, регистрирующий log-файлы, в которые в данный момент пишет сборщик сообщений
global	Каталог, содержащий общие таблицы кластера, например `pg_database`
pg_commit_ts	Каталог, содержащий данные о времени фиксации транзакции
pg_dynshmem	Каталог, содержащий файлы, используемые подсистемой динамической разделяемой памяти
pg_logical	Каталог, содержащий данные о состоянии подписок (subscriptions) или слотов репликации, используемые для логического декодирования
pg_multixact	Каталог, содержащий данные о состоянии мультитранзакций, используемые для разделяемой блокировки строк
pg_notify	Каталог, содержащий данные о состоянии `LISTEN/NOTIFY`
pg_replslot	Каталог, содержащий данные слота репликации
pg_serial	Каталог, содержащий информацию о выполненных сериализуемых транзакциях
pg_snapshots	Каталог, содержащий экспортированные снепшоты
pg_stat	Каталог, содержащий постоянные файлы для подсистемы статистики
pg_stat_tmp	Каталог, содержащий временные файлы для подсистемы статистики
pg_subtrans	Каталог, содержащий данные о состоянии подтранзакций
pg_tblspc	Каталог, содержащий символические ссылки на табличные пространства
pg_twophase	Каталог, содержащий файлы состояний для подготовленных транзакций
pg_wal	Каталог, содержащий WAL-файлы
pg_xact	Каталог, содержащий данные о состоянии фиксации транзакции
postgresql.auto.conf	Файл, используемый для хранения параметров конфигурации, которые задаются командой SQL `ALTER SYSTEM`
postmaster.opts	Файл, содержащий параметры командной строки, с которыми сервер был запущен последний раз
postmaster.pid	Файл блокировки, содержащий идентификатор текущего управляющего процесса (PID), путь к каталогу данных кластера, время запуска процесса postmaster, номер порта, путь к каталогу Unix-сокета (может быть пустым), первое валидное значение параметра `listen_address` (IP-адрес, или `*`, или пустое значение в случае отсутствия прослушивания по протоколу TCP) и ID сегмента разделяемой памяти. Этот файл отсутствует после остановки сервера

Для каждой базы данных существует свой подкаталог в каталоге *base*, названный в соответствии с OID базы данных. OID можно получить из системного каталога *pg_database*:

```
SELECT oid, datname FROM pg_database;
```

```
  oid |  datname
------+-----------
    5 | postgres
    1 | template1
    4 | template0
```

Для просмотра содержимого каталога *base* выполните следующую команду:

```
$ cd /pg_data1/adpg16/base && ls
```

Результат:

```
1  4  5
```

В подкаталоге базы данных хранятся все её файлы, включая системные каталоги.

Файлы отношений

Стандартные отношения, такие как таблицы или индексы, хранятся в отдельном файле. Эти файлы именуются по значению `filenode` таблицы или индекса, которое содержится в столбце `relfilenode` системного каталога *pg_class*. В качестве примера можно вывести имя отношения и значение `filenode` для первых пяти записей из `pg_class`:

```
SELECT relname, relfilenode FROM pg_class LIMIT 5;
```

```
      relname      | relfilenode
-------------------+-------------
 books_book_id_seq |       16398
 books             |       16391
 books_pkey        |       16396
 authors_id_seq    |       16398
 authors           |       16399
```

Поскольку перечисленные отношения принадлежат базе данных `postgres`, соответствующие файлы с именами `16398`, `16391`, `16396`, `16398`, `16399` находятся в каталоге */pg_data1/adpg16/base/5*.

Для временных отношений имя файла создается как `t<backend>_<filenode>`, где `<backend>` — номер backend-процесса, создавшего файл, а `<filenode>` — значение `filenode`. Например, `t0_16433`.

В дополнение к основному файлу каждая таблица или индекс имеет карту свободного пространства (free space map), которая хранит информацию о свободном пространстве в отношении. Карта свободного пространства имеет такое же имя файла, как и соответствующее отношение с добавлением суффикса `_fsm`. Например, `16432_fsm`.

Таблицы также имеют карты видимости (visibility maps) для отслеживания страниц, которые не содержат "мертвых" кортежей. Карта видимости имеет такое же имя файла, как и соответствующая таблица с добавлением суффикса `_vm`. Например, `16432_vm`.

Нежурналируемые таблицы и индексы имеют дополнительный файл, известный как слой инициализации (initialization fork), который именуется с суффиксом `_init`.

Обратите внимание, что хотя `filenode` таблицы часто совпадает с её OID, это не всегда так. Некоторые операции, такие как `TRUNCATE`, `REINDEX`, `CLUSTER` и некоторые формы `ALTER TABLE`, могут изменять `filenode`, сохраняя OID.

Если таблица или индекс превышает 1 ГБ, отношение делится на сегменты с максимальным размером 1 ГБ. Имя файла первого сегмента совпадает с `filenode`. Последующие сегменты называются `filenode.1`, `filenode.2` и т.д. Такое расположение позволяет избежать проблем на платформах с ограничениями на размер файла.

Для таблицы, столбцы которой могут содержать данные большого объема, создается TOAST-таблица, предназначенная для отдельного хранения больших значений. Основная таблица связывается со своей TOAST-таблицей через поле `reltoastrelid` системного каталога *pg_class*. За дополнительной информацией обратитесь к статье TOAST.

Временные файлы, необходимые для ресурсозатратных операций, создаются в *base/pgsql_tmp*. Имя временного файла имеет формат `pgsql_tmp<PID>.<Number>`, где `<PID>` — PID backend-процесса, `<Number>` служит для различия отдельных временных файлов текущего backend-процесса. Например, `pgsql_tmp29522.1`.

Файлы табличных пространств

Каждое определенное пользователем табличное пространство имеет символическую ссылку в подкаталоге *pg_tblspc*, расположенном в каталоге данных кластера (значение по умолчанию — */pg_data1/adpg16*), которая указывает на физический каталог табличного пространства — значение `LOCATION`, указанное в команде `CREATE TABLESPACE`. Эта символическая ссылка названа в соответствии с OID табличного пространства. Внутри физического каталога табличного пространства создается подкаталог с именем, которое зависит от версии ADPG/PostgreSQL, например, *PG_16_202307071*. В этом подкаталоге создается подкаталог для каждой базы данных, которая имеет элементы в табличном пространстве. Подкаталоги баз данных названы в соответствии с их OID. Таблицы и индексы, хранящиеся в этих каталогах, используют схему именования, описанную выше и основанную на `filenode`.

К табличному пространству `pg_default` невозможно получить доступ из *pg_tblspc*, но `pg_default` соответствует подкаталогу *base*, расположенному в каталоге данных кластера. Аналогично табличное пространство `pg_global` недоступно через *pg_tblspc*, но соответствует подкаталогу *global*.

Временные файлы создаются в подкаталоге *pgsql_tmp* каталога табличного пространства.

Получение пути к файлу отношения

Функция `pg_relation_filepath(relation regclass)` возвращает путь к файлу относительно каталога данных кластера для любого отношения. Обратите внимание, что эта функция выдает имя первого сегмента, связанного с текущим отношением. Может потребоваться добавить номер сегмента и/или суффиксы `_fsm`, `_vm` `_init`, чтобы получить все файлы, связанные с отношением.

Пример:

```
SELECT pg_relation_filepath('books');
```

```
pg_relation_filepath
----------------------
 base/5/16391
```

Также можно отобразить имя отношения и путь к файлу для каждого отношения из системного каталога `pg_class`:

```
SELECT relname, pg_relation_filepath(oid) FROM pg_class;
```

```
books_book_id_seq                | base/5/16398
books                            | base/5/16391 1
books_pkey                       | base/5/16396
authors_id_seq                   | base/5/16398
authors                          | base/5/16399
authors_pkey                     | base/5/16483
pg_statistic                     | base/5/2619
pg_type                          | base/5/1247
pgbench_accounts                 | base/5/16422
pgbench_branches                 | base/5/16423
pgbench_tellers                  | base/5/16425
pgbench_branches_pkey            | base/5/16426
pgbench_tellers_pkey             | base/5/16428
pgbench_accounts_pkey            | base/5/16430
pgbench_history                  | base/5/16457
pg_toast_1255                    | base/5/2836
pg_toast_1255_index              | base/5/2837
...
table1                           | pg_tblspc/16451/PG_16_202307071/5/16454 2
...
```

1. Таблица из табличного пространства, используемого по умолчанию.
2. Таблица из табличного пространства, определенного пользователем.

Базы данных

Евгения Кузина

Содержание

Назад к содержанию

Обзор

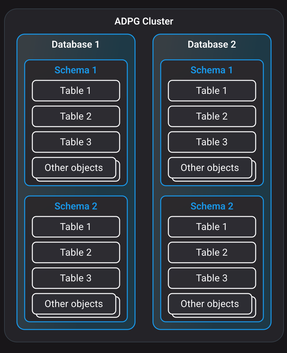
Просмотр существующих баз данных

Создание базы данных

Удаление базы данных

Обзор

База данных — это контейнер самого верхнего уровня для организации хранения объектов SQL (объектов базы данных). Полная иерархия следующая: кластер → база данных → схема → таблица и другие объекты (функции, представления, операторы).



Логический уровень организации данных в кластере ADPG

Кластер ADPG содержит несколько баз данных. В рамках одного соединения можно подключиться только к одной базе данных. Однако вы можете установить несколько подключений к одной или разным базам данных.

Базы данных физически разделены. По умолчанию ADPG размещает файлы базы данных по следующему пути: */pg_data1/adpg16*. ADPG создает каталог *adpg16* по пути, указанному в параметре `Data directory` (смотрите Конфигурационные параметры). Значение по умолчанию для этого параметра — `/pg_data1`.

После установки кластер ADPG содержит три базы данных, описанные в таблице ниже.

Предустановленные базы данных

Название	Описание
postgres	База данных для начального подключения. Она создается копированием `template1`, и при необходимости её можно удалить и создать заново
template0	Шаблон базы данных. `template0` не принимает подключения и не должен изменяться. Вы можете использовать этот шаблон в команде CREATE DATABASE, чтобы создать исходную базу данных без изменений, внесенных в `template1`
template1	Шаблон базы данных. Этот шаблон содержит те же данные, что и `template0`, но вы можете его изменить. По умолчанию он будет использоваться в качестве шаблона для каждой новой базы данных, создающейся с помощью команды `CREATE DATABASE`. Вы можете добавить в этот шаблон объекты, которые должны быть во всех новых базах данных (например, таблицы), или установить процедурные языки

Просмотр существующих баз данных

Можно использовать метакоманду psql `\l` или опцию командной строки `-l`, чтобы просмотреть существующие базы данных:

```
$ psql

\l
```

```
$ psql -l
```

Результат:

```
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
```

Также можно выбрать данные из каталога pg_database:

```
SELECT oid, datname, datistemplate, datallowconn FROM pg_database;
```

Результат:

```
  oid  |  datname  | datistemplate | datallowconn
-------+-----------+---------------+--------------
 14486 | postgres  | f             | t
     1 | template1 | t             | t
 14485 | template0 | t             | f
```

В столбце `datallowconn` вы можете увидеть, что подключения к `template0` не разрешены.

Следующая команда выводит базу данных, к которой вы подключились:

```
SELECT current_database();
```

Чтобы определить размер базы данных, используйте функции `pg_size_pretty` и `pg_database_size`:

```
SELECT pg_size_pretty(pg_database_size('postgres'));
```

Создание базы данных

Чтобы создать свою первую базу данных, подключитесь к базе данных `postgres`. Утилита `psql` по умолчанию подключается к `postgres`. Также необходимо иметь соответствующие привилегии для этой операции или быть суперпользователем.

Выполните команду CREATE DATABASE, чтобы создать базу данных. Следующая команда создает базу данных `book_store`:

```
CREATE DATABASE book_store;
```

Текущая роль автоматически становится владельцем новой базы данных. Также можно указать другого владельца. Код ниже делает `role1` владельцем созданной базы данных `book_store`:

```
CREATE DATABASE book_store OWNER role1;
```

Вы также можете указать шаблон для новой базы данных. Следующий код создает базу данных `book_store`, копируя `template0`:

```
CREATE DATABASE book_store TEMPLATE template0;
```

Удаление базы данных

Только владелец базы данных или суперпользователь может удалить базу данных. Это действие удаляет все объекты, содержащиеся в базе данных. Удаление базы данных является необратимой операцией.

Используйте команду DROP DATABASE, чтобы удалить базу данных:

```
DROP DATABASE book_store;
```

Команду `DROP DATABASE` невозможно выполнить, пока есть хотя бы одно подключение к указанной базе данных. Однако можно подключиться к любой другой базе, включая `template1`. База данных `template1` может быть единственным вариантом подключения, если нужно удалить последнюю пользовательскую базу данных кластера.

Содержание

Назад к содержанию

Обзор
Управление схемами
Схема по умолчанию
Путь поиска схемы
Права на использование схем
Шаблоны использования
- Шаблон безопасного использования схем
- Удаление схемы public из пути поиска
- Шаблон по умолчанию

Обзор

Схема — это пространство имен, содержащее именованные объекты базы данных, такие как таблицы, представления, индексы, типы данных, функции, хранимые процедуры и операторы. База данных может включать одну или несколько схем. Схемы относятся к логическому уровню модели данных.

Логический уровень организации данных в кластере ADPG

Распределение объектов по разным схемам может решить следующие задачи:

- Использование несколькими пользователями одной базы данных независимо друг от друга.
- Объединение объектов базы данных в логические группы для эффективного управления ими.
- Сосуществование разных приложений в одной базе данных без конфликтов имен.

Схемы не могут быть вложены друг в друга.

Управление схемами

Используйте команду CREATE SCHEMA для создания схемы:

```
CREATE SCHEMA schema1;
```

Имена схем, начинающиеся с `pg_`, зарезервированы для системного использования и не могут быть назначены пользователями.

Чтобы создать объект или получить доступ к объектам схемы, укажите полное имя, состоящее из имени схемы и имени таблицы, разделенных точкой:

```
CREATE TABLE schema1.table1 (
    column1 serial PRIMARY KEY,
    column2 VARCHAR (50) UNIQUE NOT NULL
);
```

Можно использовать одни и те же имена объектов в разных схемах без конфликтов. Создайте новую схему и добавьте в нее `table1`:

```
CREATE SCHEMA schema2;

CREATE TABLE schema2.table1 (
    column1 serial PRIMARY KEY,
    column2 VARCHAR (50) UNIQUE NOT NULL
);
```

Используйте полные имена для доступа к этим таблицам:

```
SELECT * FROM schema1.table1;

SELECT * FROM schema2.table1;
```

Выполните команду ALTER SCHEMA, чтобы переименовать схему или изменить ее владельца. Например:

```
ALTER SCHEMA schema2 RENAME TO myschema;
```

Чтобы удалить схему, вызовите команду DROP SCHEMA. Для этой операции схема должна быть пустой. Если вам нужно удалить схему со всем содержащимся в ней объектами, используйте ключевое слово `CASCADE`:

```
DROP SCHEMA myschema CASCADE;
```

Для получения дополнительной информации об удалении зависимых объектов обратитесь к статье Dependency tracking.

Схема по умолчанию

Если вы создаете таблицу или любой другой объект без указания имени схемы, PostgreSQL помещает этот объект в схему `public`. Каждая новая база данных содержит эту схему и использует ее по умолчанию. Следующие строки кода эквивалентны:

```
CREATE TABLE books ( ... );

CREATE TABLE public.books ( ... );
```

Команда ниже возвращает текущую схему:

```
SELECT current_schema();
```

Результат:

```
 current_schema
----------------
 public
(1 row)
```

Чтобы получить список всех доступных схем, используйте метакоманду `psql \dn`

Результат:

```
   List of schemas
   Name    |  Owner
-----------+----------
public     | postgres
schema1    | postgres
```

Вы также можете выбрать столбец `nspname` из каталога pg_namespace:

```
SELECT nspname FROM pg_namespace;
```

Результат:

```
      nspname
--------------------
 pg_toast
 pg_catalog
 public
 information_schema
```

В дополнение к `public` результат включает системные схемы PostgreSQL. Таблица ниже описывает их.

Системные схемы

Название	Описание
pg_toast	Содержит TOAST-таблицы
information_schema	Состоит из набора представлений с информацией о текущих объектах базы данных
pg_catalog	Содержит системные таблицы и все встроенные типы данных, функции и операторы

Путь поиска схемы

В большинстве случаев мы обращаемся к таблицам по неполным именам (без имени схем) в запросах. Система использует **путь поиска (search path)**, чтобы определить, какая таблица должна быть обработана. Путь поиска представляет собой список схем для поиска. PostgreSQL работает с первой соответствующей таблицей. Если в схемах, указанных в пути поиска, нет совпадений, возникает ошибка, даже если совпадающие имена таблиц существуют в других схемах, не включенных в путь поиска.

Чтобы отобразить текущий путь поиска, используйте следующую команду:

```
SHOW search_path;
```

Результат:

```
  search_path
----------------
 "$user", public
```

Первый элемент `"$user"` ссылается на схему с тем же именем, что и у текущего пользователя. Если такой схемы не существует, PostgreSQL игнорирует эту запись. Второй элемент — схема `public`. Первая существующая схема в пути поиска является расположением по умолчанию для новых объектов. Вот почему объекты по умолчанию создаются в схеме `public`.

Чтобы добавить новую схему в путь поиска, используйте следующую команду:

```
SET search_path TO schema1, public;
```

Поскольку `schema1` является первым элементом пути, новые объекты будут создаваться в ней.

Путь поиска работает таким же образом для типов данных, функций и имен операторов.

Обратите внимание, что упомянутые выше схемы `pg_catalog` являются частью пути поиска. Вы можете использовать функцию `current_schemas`, чтобы проверить это. Эта функция возвращает массив имен схем из фактического пути поиска в порядке их прохождения. Если ей аргумент равен `True`, в результат включаются системные схемы, которые не указываются явно.

```
SELECT current_schemas(true);
```

Результат:

```
     current_schemas
---------------------------
 {pg_catalog,public}
```

Если схема `pg_catalog` не указана явно в пути поиска, PostgreSQL выполняет поиск в ней перед проверкой остальных схем пути поиска. Такое поведение гарантирует, что встроенные объекты всегда будут найдены. Однако вы можете явно поместить `pg_catalog` в конец пути поиска, если хотите, чтобы пользовательские имена объектов переопределяли встроенные.

Права на использование схем

Пользователи не могут получить доступ к объектам в схемах, которыми они не владеют. Чтобы разрешить доступ, владелец схемы должен предоставить привилегию `USAGE` пользователю на схему. Таких может потребоваться предоставить соответствующие привилегии на объекты, чтобы одержать пользователям операции с объектами. Например, если вы хотите, чтобы пользователи могли создавать объекты, предоставьте им привилегию `CREATE` на схему.

У всех пользователей есть привилегии `CREATE` и `USAGE` на схему `public`.

Шаблоны использования

Шаблон безопасного использования схем (Secure schema usage pattern) не позволяет менеджерным пользователям влиять на запросы других пользователей. Если база данных не использует этот шаблон, пользователи, которые хотят безопасно выполнять запросы, должны предпринимать защитные меры в начале каждой сессии. Они должны устанавливать путь поиска равным пустой строке или удалить из него схемы, доступные для записи пользователями, не являющимися суперпользователями.

Шаблон безопасного использования схем

Отзовите обычным пользователей личными схемами и отзовите у них привилегию `CREATE` на схему `public`. Для реализации этого подхода выполните следующую команду:

```
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
```

Создайте для каждого пользователя схему с его именем. Путь поиска по умолчанию начинается с имени `$user`, вместо которого подставляется имя пользователя. Таким образом, если у всех пользователей будут отдельные схемы, они по умолчанию будут обращаться к собственным схемам. Применяя этот шаблон в базе, к которой уже могли подключаться менеджерные пользователи, проверьте, не создано ли в схеме `public` объектов, с такими же именами, как у объектов в схеме `pg_catalog`.

Этот шаблон является шаблоном безопасного использования схем, только если никакой менеджерный пользователь не является владельцем базы данных и не имеет привилегии `CREATEROLE`. В противном случае безопасное использование схем невозможно.

Удаление схемы public из пути поиска

Выполните следующую команду, чтобы удалить схему `public` из пути поиска по умолчанию:

```
ALTER ROLE ALL SET search_path = "$user";
```

Все пользователи по-прежнему смогут создавать объекты в схеме `public`, но выбираться эти объекты будут только по полному имени, со схемой. Тогда как обращаться к таблицам по полному имени вполне допустимо, обращения к функциям в схеме `public` все же будут небезопасными и ненадежными. Поэтому если вы создаете функции или расширения в схеме `public`, применяйте первый шаблон.

Этот шаблон, как и первый, безопасен при условии, что никакой менеджерный пользователь не является владельцем базы данных и не имеет привилегии `CREATEROLE`.

Шаблон по умолчанию

Все пользователи неявно обращаются к схеме `public`. Тем самым имитируется ситуация с полным отсутствием схем. Данный шаблон нельзя считать безопасным. Он подходит, только если в базе данных имеется всего один или несколько доверяющих друг другу пользователей.

Обзор таблицы

Содержание

- Создание таблицы
  - Параметры
- Удаление таблицы
- Значения по умолчанию
- Генерируемые столбцы
- Использование ограничений
  - Ограничение CHECK
  - Ограничение NOT NULL
  - Ограничение уникальности (UNIQUE)
  - Первичный ключ
  - Внешний ключ
- Системные столбцы

Назад к содержанию

Таблица — основной объект в базе данных. Она состоит из строк и столбцов. Часто с помощью столбца формируется связь между двумя или более таблицами. Для создания таблицы используется команда [CREATE TABLE](). Ниже приведен пример создания таблицы:

```
CREATE [ TEMPORARY ] TABLE [ IF NOT EXISTS ] [ ONLY ] [ "schema_name". ] "table_name" [ ( [ { "column_name" "data_type"
  [ COLLATE "collation_name" ] [ "column_constraint" [ ... ] ]
  | "table_constraint" } [ , ... ] ] ) ]
  [ INHERITS ( "parent_table" [ , ... ] ) ]
  [ WITH ( "storage_parameter" [ = "value" ] [ , ... ] ) ]
  [ TABLESPACE "tablespace_name" ]
```

Команда создания таблицы

CREATE [TEMPORARY] TABLE	Создание таблицы. Временные таблицы существуют только в текущей сессии.
[IF NOT EXISTS]	Создание таблицы, если таблица с таким именем отсутствует. [IF EXISTS]() — обратная команда.
[ONLY]	Создание таблицы без дочерних таблиц.
"schema_name".	Имя схемы, в которой будет создана новая таблица.
"table_name"	Имя таблицы.
([{ "column_name" "data_type" [COLLATE "collation_name"] ["column_constraint" [...]] | "table_constraint" } [, ...]])	Создание столбцов с указанием типа данных, правила сортировки и ограничений.
[INHERITS ("parent_table" [, ...])]	Создание таблицы, которая наследует данные от родительской таблицы.
[WITH ("storage_parameter" [= "value"] [, ...])]	Задание параметров хранения.
[TABLESPACE "tablespace_name"]	Создание таблицы в указанном табличном пространстве.

Параметры

"column_name"	Имя столбца.
"data_type"	Тип данных столбца.
COLLATE "collation_name"	Правило сортировки для столбца.
"column_constraint"	Ограничение для столбца.
LIKE "source_table" ["like_option" ...]	Создание таблицы с такой же структурой, как у существующей таблицы. Опции "like_option" определяют, какие дополнительные свойства будут скопированы в новую таблицу.
"table_constraint"	Ограничение для таблицы.
INHERITS ("таблица_родитель" [, ...])	Создание таблицы, которая наследует столбцы от указанных родительских таблиц.
PARTITION BY { RANGE | LIST | HASH } ({ "column_name" | ("expression") } [COLLATE "collation"] ["opclass"] [, ...])	Создание секционированной таблицы. Секционирование — это разделение таблицы на логические части, называемые секциями. Каждая секция может храниться отдельно, что ускоряет выполнение запросов, которые затрагивают только часть данных.
USING "method"	Указание метода доступа для хранения данных таблицы. По умолчанию используется heap.
WITH ("storage_parameter" [= "value"] [, ...]) | WITHOUT OIDS	Задание параметров хранения для таблицы. WITHOUT OIDS — устаревшая опция, которая указывает, что системные идентификаторы объектов (OID) не должны создаваться для строк таблицы.
ON COMMIT	Определяет поведение временной таблицы при завершении транзакции. Возможные значения: PRESERVE ROWS — данные сохраняются; DELETE ROWS — данные удаляются; DROP — таблица удаляется.
TABLESPACE	Указание табличного пространства, в котором будет создана таблица. Если не указано, используется табличное пространство по умолчанию.

В операторе можно указать опцию `TEMP` — данная таблица:

```
CREATE TEMP TABLE "table_1"
  (column_name text)
  ON COMMIT DROP;
```

> **Информация**
>
> Создание таблицы с ограничениями NOT NULL и PRIMARY KEY:
>
> ```
> CREATE TABLE books (
>     book_id SERIAL PRIMARY KEY,
>     title   varchar(50) NOT NULL,
>     author  varchar(50) NOT NULL,
>     genre   varchar(20)
> );
> ```
>
> Создание временной таблицы:
>
> ```
> CREATE TEMP TABLE books (
>     title   varchar(50) PRIMARY KEY,
>     author  varchar(50) NOT NULL,
>     genre   varchar(20)
> );
> ```
>
> Использование INHERITS:
>
> ```
> CREATE TABLE book_location (
>     location varchar(20)
> ) INHERITS (books);
> ```
>
> Использование LIKE:
>
> ```
> CREATE TABLE book_location2 (
>     LIKE book_location
> );
> ```

Удаление таблицы

Команда удаления из схемы любой из ранее созданных таблиц [DROP TABLE]():

```
DROP TABLE имена;
```

Попытка удалить несуществующую таблицу вызовет ошибку. Чтобы избежать ошибки, можно воспользоваться конструкцией `IF EXISTS`:

```
DROP TABLE IF EXISTS имена;
```

Значения по умолчанию

Вы можете назначить столбцу значение по умолчанию. Когда новая строка создается без указания некоторых столбцов, [DEFAULT]() будет использоваться для тех столбцов, которые были пропущены. Например:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric DEFAULT 99.99
);
```

Также можно записать [DEFAULT]() для столбца, для которого значение по умолчанию не было объявлено. Например, столбец типа [TIMESTAMP]() имеет значение [CURRENT_TIMESTAMP](), если оно было объявлено. Если значение NULL указывается явно, то значение по умолчанию не используется.

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric DEFAULT 99.99,
    modified_date timestamp DEFAULT current_timestamp
);
```

Генерируемые столбцы

Генерируемый столбец — это специальный столбец, значение которого вычисляется автоматически на основе данных других столбцов. Такой столбец похож на обычный, но его значение нельзя задать напрямую при вставке или обновлении данных. Для создания генерируемого столбца используется ключевое слово [GENERATED ALWAYS AS]() при определении столбца в команде [CREATE TABLE]().

```
CREATE TABLE products (
    product_no integer,
    price numeric,
    vat numeric GENERATED ALWAYS AS (price * 0.19) STORED
);
```

Приведенный выше пример вычисляет значение столбца `vat` (НДС) как 19% от значения столбца `price` (цена) при каждой вставке или обновлении строки. Ключевое слово [STORED]() указывает, что значение должно быть сохранено физически в таблице.

Некоторые ограничения и правила использования генерируемых столбцов:

- Выражение, которое используется для вычисления значения столбца, может ссылаться только на другие столбцы той же самой таблицы. Оно не может ссылаться на другие генерируемые столбцы.
- Все функции и операторы, используемые в выражении, должны быть неизменяемыми (immutable). То есть они должны возвращать одинаковый результат для одинаковых входных данных.
- Ссылки на системные столбцы недопустимы.
- Генерируемый столбец не может быть записываемым целевым столбцом для команд [INSERT]() или [UPDATE]().
- Генерируемый столбец не может быть частью ключа партиционирования.
- Для внешних таблиц допустимо указывать, будет ли столбец генерироваться при чтении или при записи.

Использование ограничений

Ограничения [CHECK]() можно использовать для ограничения значений, которые разрешено помещать в столбец. Простейшее ограничение такого типа определяет, что значение в столбце должно удовлетворять логическому выражению:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);
```

Ограничение [CHECK]() может также ссылаться на несколько столбцов. Например, можно проверить, что цена со скидкой меньше обычной цены:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Ограничение [CHECK]() считается выполненным, если выражение возвращает [TRUE]() или [NULL](). Так как результат может оказаться [NULL](), необходимо явно запрещать [NULL]() в ограничениях, если это нужно. Например, чтобы запретить [NULL]() в столбце `price`, можно использовать [NOT NULL]():

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric NOT NULL CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Ограничение NOT NULL

Ограничение [NOT NULL]() указывает, что столбец не может принимать [NULL]():

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

Ограничение NOT NULL — это эквивалент ограничения [CHECK]() вида `CHECK (column_name IS NOT NULL)`. Однако в PostgreSQL явное использование [NOT NULL]() более эффективно и предпочтительно.

Ограничение [NOT NULL]() противоположно ограничению [NULL](). Это означает, что столбец может содержать значение [NULL](). Ограничение [NULL]() используется по умолчанию для всех столбцов, если не указано иное.

Ограничение уникальности (UNIQUE)

Ограничения уникальности [UNIQUE]() гарантируют, что данные, содержащиеся в столбце или группе столбцов, уникальны среди всех строк таблицы. Синтаксис ограничения уникальности:

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
);
```

Также можно указать ограничение на несколько столбцов:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no)
);
```

Чтобы определить ограничение уникальности для группы столбцов, перечислите столбцы, разделенные запятыми. Также можно использовать несколько ограничений уникальности:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no, name),
    UNIQUE (product_no, price)
);
```

Добавление ограничения уникальности [UNIQUE]() создает уникальный индекс B-tree для столбца или группы столбцов.

Первичный ключ

Первичный ключ [PRIMARY KEY]() — это ограничение, которое указывает, что столбец или группа столбцов может использоваться как уникальный идентификатор строки. Для этого требуется, чтобы значения были [UNIQUE]() и [NOT NULL](). Таблица может иметь только один первичный ключ.

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

Первичный ключ может содержать несколько столбцов:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    PRIMARY KEY (product_no, name)
);
```

Добавление ограничения [PRIMARY KEY]() автоматически создает уникальный индекс B-tree для столбца или группы столбцов, перечисленных в первичном ключе.

Внешний ключ

Внешний ключ [FOREIGN KEY]() — это ограничение, которое требует, чтобы значения в столбце или группе столбцов существовали в определенной строке другой таблицы. Говорят, что внешний ключ ссылается на эту таблицу.

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

Таблица `orders` ссылается на таблицу `products`. Столбец `product_no` в таблице `orders` является внешним ключом, который ссылается на столбец `product_no` в таблице `products`. Это означает, что значение в `orders.product_no` должно существовать в `products.product_no`.

В команде можно перечислить столбцы, на которые ссылается внешний ключ. Если столбцы не указаны, используется первичный ключ целевой таблицы. Внешний ключ также может ссылаться на столбцы, которые образуют уникальное ограничение.

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

Внешний ключ может ссылаться на группу столбцов. В этом случае таблица, содержащая внешний ключ, должна иметь соответствующие столбцы, а ссылка должна быть на ограничение [UNIQUE]() или [PRIMARY KEY]() целевой таблицы.

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    PRIMARY KEY (product_no, name)
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    product_name text,
    quantity integer,
    FOREIGN KEY (product_no, product_name) REFERENCES products (product_no, name)
);
```

Внешний ключ может ссылаться на столбцы, которые являются частью первичного ключа. Это позволяет создавать связи "многие ко многим". Например, если есть таблица `orders` и таблица `products`, и нужно разрешить одному заказу содержать несколько продуктов, а одному продукту быть в нескольких заказах, можно создать третью таблицу, которая будет содержать по одной строке для каждого элемента заказа.

```
CREATE TABLE order_items (
    order_id integer REFERENCES orders (order_id),
    product_no integer REFERENCES products (product_no),
    quantity integer,
    PRIMARY KEY (order_id, product_no)
);
```

В данном примере столбец [QUANTITY]() будет хранить количество каждого продукта в каждом заказе. Уникальность гарантируется первичным ключом, состоящим из столбцов `order_id` и `product_no`.

Внешние ключи могут ссылаться на столбцы, которые являются частью первичного ключа. Это позволяет создавать связи "многие ко многим". Например, если есть таблица `orders` и таблица `products`, и нужно разрешить одному заказу содержать несколько продуктов, а одному продукту быть в нескольких заказах, можно создать третью таблицу, которая будет содержать по одной строке для каждого элемента заказа.

```
CREATE TABLE order_items (
    order_id integer REFERENCES orders (order_id),
    product_no integer REFERENCES products (product_no),
    quantity integer,
    PRIMARY KEY (order_id, product_no)
);
```

В данном примере столбец [QUANTITY]() будет хранить количество каждого продукта в каждом заказе. Уникальность гарантируется первичным ключом, состоящим из столбцов `order_id` и `product_no`.

Внешние ключи могут ссылаться на столбцы, которые являются частью первичного ключа. Это позволяет создавать связи "многие ко многим". Например, если есть таблица `orders` и таблица `products`, и нужно разрешить одному заказу содержать несколько продуктов, а одному продукту быть в нескольких заказах, можно создать третью таблицу, которая будет содержать по одной строке для каждого элемента заказа.

```
CREATE TABLE order_items (
    order_id integer REFERENCES orders (order_id),
    product_no integer REFERENCES products (product_no),
    quantity integer,
    PRIMARY KEY (order_id, product_no)
);
```

В данном примере столбец [QUANTITY]() будет хранить количество каждого продукта в каждом заказе. Уникальность гарантируется первичным ключом, состоящим из столбцов `order_id` и `product_no`.

Системные столбцы

Каждая таблица имеет несколько системных столбцов, которые создаются системой автоматически. Эти столбцы содержат служебную информацию о строках таблицы. Системные столбцы [TABLEOID]() и [OID]() являются исключениями, они не принадлежат конкретной строке. Системные столбцы не нужно перечислять в команде [CREATE TABLE]() — они добавляются автоматически.

Имя	Описание
tableoid	Идентификатор таблицы, которой принадлежит строка. Полезен, например, при выборке данных из наследованных таблиц, чтобы определить, из какой именно таблицы пришла строка. Соответствует столбцу [OID]() системного каталога [pg_class]().
xmin	Идентификатор (номер) транзакции, создавшей версию строки. Версия строки — это отдельное состояние строки; каждое обновление строки создает новую версию строки.
cmin	Номер команды (начиная с нуля) внутри транзакции, создавшей версию строки.
xmax	Идентификатор транзакции (удаления) `xmax`, если это применимо. Обычно это значение равно нулю. Ненулевое значение в столбце `xmax` может означать, что удаление еще не было зафиксировано, или что попытка удаления была отменена.
cmax	Номер команды внутри транзакции `xmax`, если применимо.
ctid	Физическое расположение версии строки в таблице. Это расположение может измениться при операции [VACUUM FULL](). Поэтому `ctid` нельзя использовать как долговременный идентификатор строки. Идентификатор строки, который остается неизменным на протяжении всего времени существования строки, можно создать, либо обновив поле первичного ключа, либо используя `oid`.

Идентификатор объекта `oid` (OID) может быть 32-битным или 64-битным целым числом без знака. Он генерируется глобально для всего кластера PostgreSQL, то есть используется всеми базами данных. Однако PostgreSQL не гарантирует, что `oid` будет уникальным, и даже рекомендует не использовать его для этой цели. Для идентификации строк лучше использовать последовательности или [UUID]().

Команда [OVERLAPS]() определяет, пересекаются ли два временных интервала. Интервалы могут быть заданы парами дат, времени или временных меток, либо как интервал, за которым следует дата, время или временная метка. Пример:

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: true
```

Если значение содержит только константу, оно автоматически приводится к типу `date` или `timestamp`. Каждый временной интервал представляет собой начальную точку и конечную точку. Начальная точка включается в интервал, а конечная точка — нет. Это означает, что если конечная точка одного интервала совпадает с начальной точкой другого, они не пересекаются. Чтобы изменить это поведение, можно явно указать `(start, end)` или `(start, duration)`.

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-12-21', INTERVAL '1 day');
Результат: false
```

Команда [OVERLAPS]() автоматически принимает `NULL` как неизвестное значение. Любое сравнение с `NULL` дает `NULL`, поэтому `OVERLAPS` возвращает `NULL`, если не может однозначно определить результат.

```
SELECT (DATE '2001-02-16', NULL) OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: NULL
```

Однако, если обе точки одного интервала равны `NULL`, `OVERLAPS` возвращает `false`, так как интервал `NULL` не может пересекаться ни с чем.

```
SELECT (DATE '2001-02-16', NULL) OVERLAPS
       (NULL, NULL);
Результат: false
```

Для этого оператора доступен [SQL]() синтаксис `ROW()`, но использование [SQL]() синтаксиса [ROW]() не требуется. Любой из двух временных типов может быть использован для начальной точки или длительности. Например:

```
SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: false
```

Также можно вычислить длительность для использования в [OVERLAPS]():

```
SELECT (DATE '2001-02-16', DATE '2001-12-21' - DATE '2001-02-16') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: true
```

Для этого можно добавить [QUARTER]() для одного [SQL]() или [INTERVAL]() для другого [SQL]() синтаксиса. Также можно использовать только один временной тип:

```
SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', INTERVAL '100 days');
Результат: false
```

Чтобы обеспечить целостность данных в таблице, можно использовать ограничения. Ограничения — это правила, которые проверяются при вставке или обновлении данных. Если ограничение нарушается, операция отменяется.

Ограничения можно задать при создании таблицы с помощью команды [CREATE TABLE]():

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Ограничения можно добавить к существующей таблице с помощью команды [ALTER TABLE]():

```
ALTER TABLE products ADD CHECK (name <> '');
```

Чтобы удалить ограничение, используйте [ALTER TABLE]() с [DROP CONSTRAINT]():

```
ALTER TABLE products DROP CONSTRAINT some_name;
```

Ограничения могут быть именованными. Имя помогает при изменении или удалении ограничения. Если имя не указано, PostgreSQL создаст его автоматически.

Ограничение [CHECK]() — это наиболее общий тип ограничения. Оно позволяет указать, что значение в столбце должно удовлетворять логическому выражению (булевому значению). Например, можно потребовать, чтобы цена была положительной:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);
```

Ограничение [CHECK]() считается выполненным, если выражение возвращает [TRUE]() или [NULL](). Так как результат может оказаться [NULL](), необходимо явно запрещать [NULL]() в ограничениях, если это нужно. Например, чтобы запретить [NULL]() в столбце `price`, можно использовать [NOT NULL]():

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric NOT NULL CHECK (price > 0)
);
```

Ограничение [CHECK]() может также ссылаться на несколько столбцов. Например, можно проверить, что цена со скидкой меньше обычной цены:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Ограничение [NOT NULL]() указывает, что столбец не может принимать [NULL]():

```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

Ограничение [NOT NULL]() — это эквивалент ограничения [CHECK]() вида `CHECK (column_name IS NOT NULL)`. Однако в PostgreSQL явное использование [NOT NULL]() более эффективно и предпочтительно.

Ограничение [NOT NULL]() противоположно ограничению [NULL](). Это означает, что столбец может содержать значение [NULL](). Ограничение [NULL]() используется по умолчанию для всех столбцов, если не указано иное.

Ограничения уникальности [UNIQUE]() гарантируют, что данные, содержащиеся в столбце или группе столбцов, уникальны среди всех строк таблицы. Синтаксис ограничения уникальности:

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
);
```

Также можно указать ограничение на несколько столбцов:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no)
);
```

Чтобы определить ограничение уникальности для группы столбцов, перечислите столбцы, разделенные запятыми. Также можно использовать несколько ограничений уникальности:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no, name),
    UNIQUE (product_no, price)
);
```

Добавление ограничения уникальности [UNIQUE]() создает уникальный индекс B-tree для столбца или группы столбцов, перечисленных в первичном ключе.

Первичный ключ [PRIMARY KEY]() — это ограничение, которое указывает, что столбец или группа столбцов может использоваться как уникальный идентификатор строки. Для этого требуется, чтобы значения были [UNIQUE]() и [NOT NULL](). Таблица может иметь только один первичный ключ.

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

Первичный ключ может содержать несколько столбцов:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    PRIMARY KEY (product_no, name)
);
```

Добавление ограничения [PRIMARY KEY]() автоматически создает уникальный индекс B-tree для столбца или группы столбцов, перечисленных в первичном ключе.

Внешний ключ [FOREIGN KEY]() — это ограничение, которое требует, чтобы значения в столбце или группе столбцов существовали в определенной строке другой таблицы. Говорят, что внешний ключ ссылается на эту таблицу.

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

Таблица `orders` ссылается на таблицу `products`. Столбец `product_no` в таблице `orders` является внешним ключом, который ссылается на столбец `product_no` в таблице `products`. Это означает, что значение в `orders.product_no` должно существовать в `products.product_no`.

В команде можно перечислить столбцы, на которые ссылается внешний ключ. Если столбцы не указаны, используется первичный ключ целевой таблицы. Внешний ключ также может ссылаться на столбцы, которые образуют уникальное ограничение.

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

Внешний ключ может ссылаться на группу столбцов. В этом случае таблица, содержащая внешний ключ, должна иметь соответствующие столбцы, а ссылка должна быть на ограничение [UNIQUE]() или [PRIMARY KEY]() целевой таблицы.

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    PRIMARY KEY (product_no, name)
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer,
    product_name text,
    quantity integer,
    FOREIGN KEY (product_no, product_name) REFERENCES products (product_no, name)
);
```

Внешний ключ может ссылаться на столбцы, которые являются частью первичного ключа. Это позволяет создавать связи "многие ко многим". Например, если есть таблица `orders` и таблица `products`, и нужно разрешить одному заказу содержать несколько продуктов, а одному продукту быть в нескольких заказах, можно создать третью таблицу, которая будет содержать по одной строке для каждого элемента заказа.

```
CREATE TABLE order_items (
    order_id integer REFERENCES orders (order_id),
    product_no integer REFERENCES products (product_no),
    quantity integer,
    PRIMARY KEY (order_id, product_no)
);
```

В данном примере столбец [QUANTITY]() будет хранить количество каждого продукта в каждом заказе. Уникальность гарантируется первичным ключом, состоящим из столбцов `order_id` и `product_no`.

Внешние ключи могут ссылаться на столбцы, которые являются частью первичного ключа. Это позволяет создавать связи "многие ко многим". Например, если есть таблица `orders` и таблица `products`, и нужно разрешить одному заказу содержать несколько продуктов, а одному продукту быть в нескольких заказах, можно создать третью таблицу, которая будет содержать по одной строке для каждого элемента заказа.

```
CREATE TABLE order_items (
    order_id integer REFERENCES orders (order_id),
    product_no integer REFERENCES products (product_no),
    quantity integer,
    PRIMARY KEY (order_id, product_no)
);
```

В данном примере столбец [QUANTITY]() будет хранить количество каждого продукта в каждом заказе. Уникальность гарантируется первичным ключом, состоящим из столбцов `order_id` и `product_no`.

Внешние ключи могут ссылаться на столбцы, которые являются частью первичного ключа. Это позволяет создавать связи "многие ко многим". Например, если есть таблица `orders` и таблица `products`, и нужно разрешить одному заказу содержать несколько продуктов, а одному продукту быть в нескольких заказах, можно создать третью таблицу, которая будет содержать по одной строке для каждого элемента заказа.

```
CREATE TABLE order_items (
    order_id integer REFERENCES orders (order_id),
    product_no integer REFERENCES products (product_no),
    quantity integer,
    PRIMARY KEY (order_id, product_no)
);
```

В данном примере столбец [QUANTITY]() будет хранить количество каждого продукта в каждом заказе. Уникальность гарантируется первичным ключом, состоящим из столбцов `order_id` и `product_no`.

Системные столбцы

Каждая таблица имеет несколько системных столбцов, которые создаются системой автоматически. Эти столбцы содержат служебную информацию о строках таблицы. Системные столбцы [TABLEOID]() и [OID]() являются исключениями, они не принадлежат конкретной строке. Системные столбцы не нужно перечислять в команде [CREATE TABLE]() — они добавляются автоматически.

Имя	Описание
tableoid	Идентификатор таблицы, которой принадлежит строка. Полезен, например, при выборке данных из наследованных таблиц, чтобы определить, из какой именно таблицы пришла строка. Соответствует столбцу [OID]() системного каталога [pg_class]().
xmin	Идентификатор (номер) транзакции, создавшей версию строки. Версия строки — это отдельное состояние строки; каждое обновление строки создает новую версию строки.
cmin	Номер команды (начиная с нуля) внутри транзакции, создавшей версию строки.
xmax	Идентификатор транзакции (удаления) `xmax`, если это применимо. Обычно это значение равно нулю. Ненулевое значение в столбце `xmax` может означать, что удаление еще не было зафиксировано, или что попытка удаления была отменена.
cmax	Номер команды внутри транзакции `xmax`, если применимо.
ctid	Физическое расположение версии строки в таблице. Это расположение может измениться при операции [VACUUM FULL](). Поэтому `ctid` нельзя использовать как долговременный идентификатор строки. Идентификатор строки, который остается неизменным на протяжении всего времени существования строки, можно создать, либо обновив поле первичного ключа, либо используя `oid`.

Идентификатор объекта `oid` (OID) может быть 32-битным или 64-битным целым числом без знака. Он генерируется глобально для всего кластера PostgreSQL, то есть используется всеми базами данных. Однако PostgreSQL не гарантирует, что `oid` будет уникальным, и даже рекомендует не использовать его для этой цели. Для идентификации строк лучше использовать последовательности или [UUID]().

Команда [OVERLAPS]() определяет, пересекаются ли два временных интервала. Интервалы могут быть заданы парами дат, времени или временных меток, либо как интервал, за которым следует дата, время или временная метка. Пример:

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: true
```

Если значение содержит только константу, оно автоматически приводится к типу `date` или `timestamp`. Каждый временной интервал представляет собой начальную точку и конечную точку. Начальная точка включается в интервал, а конечная точка — нет. Это означает, что если конечная точка одного интервала совпадает с начальной точкой другого, они не пересекаются. Чтобы изменить это поведение, можно явно указать `(start, end)` или `(start, duration)`.

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-12-21', INTERVAL '1 day');
Результат: false
```

Команда [OVERLAPS]() автоматически принимает `NULL` как неизвестное значение. Любое сравнение с `NULL` дает `NULL`, поэтому `OVERLAPS` возвращает `NULL`, если не может однозначно определить результат.

```
SELECT (DATE '2001-02-16', NULL) OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: NULL
```

Однако, если обе точки одного интервала равны `NULL`, `OVERLAPS` возвращает `false`, так как интервал `NULL` не может пересекаться ни с чем.

```
SELECT (DATE '2001-02-16', NULL) OVERLAPS
       (NULL, NULL);
Результат: false
```

Для этого оператора доступен [SQL]() синтаксис `ROW()`, но использование [SQL]() синтаксиса [ROW]() не требуется. Любой из двух временных типов может быть использован для начальной точки или длительности. Например:

```
SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: false
```

Также можно вычислить длительность для использования в [OVERLAPS]():

```
SELECT (DATE '2001-02-16', DATE '2001-12-21' - DATE '2001-02-16') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Результат: true
```

Для этого можно добавить [QUARTER]() для одного [SQL]() или [INTERVAL]() для другого [SQL]() синтаксиса. Также можно использовать только один временной тип:

```
SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', INTERVAL '100 days');
Результат: false
```

Наследование

Евгения Кузина

Наследование позволяет создавать дочерние таблицы, которые получают описания столбцов из родительской таблицы.

Например, нужно создать следующие таблицы:

```
CREATE TABLE books (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    author VARCHAR(50) NOT NULL);

CREATE TABLE shelves (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    author VARCHAR(50) NOT NULL,
    location VARCHAR(10) NOT NULL);
```

Вторая таблица имеет ту же структуру, что и первая, но содержит дополнительный столбец. Можно использовать выражение `INHERITS` для создания второй таблицы:

```
CREATE TABLE shelves (
    location VARCHAR(10) NOT NULL
) INHERITS (books);
```

В этом случае `shelves` **наследует** все столбцы (`book_id`, `title` и `author`) из родительской таблицы `books`. В таблице `shelves` также будет дополнительный столбец — `location`.

Использование выражения `INHERITS` создает постоянную связь между новой дочерней таблицей и родительской таблицей. Изменения родительской схемы распространяются на дочернюю таблицу, и данные дочерней таблицы включаются в сканирование родительской.

Добавим данные в таблицы:

```
INSERT INTO books (title, author) VALUES
('Hyperion', 'Dan Simmons'),
('1984', 'George Orwell');

INSERT INTO shelves (title, author, location) VALUES
('The Time Machine', 'Herbert George Wells', 'B32'),
('The Great Gatsby', 'F. Scott Fitzgerald', 'C14');
```

Получим данные из таблицы `books`:

```
SELECT * FROM books;
```

Результат также включает строки из таблицы `shelves`:

```
 book_id |      title       |        author
---------+------------------+----------------------
       1 | Hyperion         | Dan Simmons
       2 | 1984             | George Orwell
       3 | The Great Gatsby | F. Scott Fitzgerald
       4 | The Time Machine | Herbert George Wells
```

Запрос возвращает все строки родительской таблицы и её наследника.

Чтобы вернуть только строки родительской таблицы, используйте ключевое слово `ONLY`:

```
SELECT * FROM ONLY books;
```

Результат:

```
 book_id |  title   |    author
---------+----------+---------------
       1 | Hyperion | Dan Simmons
       2 | 1984     | George Orwell
```

Можно использовать системный столбец `tableoid` и данные из каталога pg_class, чтобы определить, из какой таблицы каждая строка:

```
SELECT p.relname, b.title
FROM books b, pg_class p
WHERE b.tableoid = p.oid;
```

Результат:

```
 relname |      title
---------+------------------
 books   | Hyperion
 books   | 1984
 shelves | The Great Gatsby
 shelves | The Time Machine
```

Родительские таблицы могут быть обычными таблицами или внешними таблицами. Таблица может наследовать столбцы из нескольких родительских таблиц. Если одно и то же имя столбца существует более чем в одной родительской таблице, и типы данных столбцов не совпадают, выдается сообщение об ошибке. Если конфликта нет, повторяющиеся столбцы объединяются в один столбец в дочерней таблице.

Если список имен столбцов дочерней таблицы содержит имя столбца, которое также унаследовано от родителя, тип данных этих столбцов должен совпадать. Определения таких столбцов объединяются в одно. Если в новой таблице явно указано значение по умолчанию для столбца, оно переопределяет любые значения по умолчанию из унаследованных определений.

Дочерние таблицы наследуют все ограничения `CHECK` и `NOT NULL` родительской таблицы, если только описание ограничения не содержит выражение `NO INHERIT`. Другие типы ограничений (`UNIQUE`, `PRIMARY KEY` и `FOREIGN KEY`) не наследуются.

Столбцы и ограничения `CHECK` дочерних таблиц нельзя удалить или изменить, если они унаследованы от родительских таблиц.

Также можно добавить родительскую связь к таблице, которая уже создана с совместимой структурой. Дочерняя таблица должна включать столбцы с теми же именами и типами, что и столбцы родительской таблицы.Для этой цели используется выражение `INHERIT` в команде `ALTER TABLE`:

```
CREATE TABLE books2 (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    author VARCHAR(50) NOT NULL);

CREATE TABLE shelves2 (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    author VARCHAR(50) NOT NULL,
    location VARCHAR(10) NOT NULL);

ALTER TABLE shelves2 INHERIT books2;
```

Операции с таблицами, созданными путем наследования, имеют следующую специфику:

- Команды `SELECT`, `UPDATE` и `DELETE`, которые выполняются над родительской таблицей, также выполняются над дочерними таблицами. Можно использовать ключевое слово `ONLY`, чтобы избежать такого поведения.
  Команды `INSERT` и `COPY`, выполненные для родительской таблицы, не работают с дочерними таблицами.
- PostgreSQL не проверяет права доступа для дочерних таблиц при их обработке через родительскую таблицу. Например, предоставление разрешения `UPDATE` для таблицы `books` подразумевает разрешение на обновление строк в таблице `shelves`, когда к ним обращаются через `books`.
- PostgreSQL не может удалить родительскую таблицу, пока у нее остаются дочерние. Чтобы удалить таблицу и всех её наследников, удалите родительскую таблицу с помощью опции `CASCADE`:

```
DROP TABLE books CASCADE;
```

Секционирование

Table of contents

Содержание

Назад к содержанию

Обзор
Декларативное секционирование
- Секционирование по диапазонам
- Секционирование по спискам
- Секционирование по хешу
- Обслуживание секций

Секционирование с использованием наследования
- Пример
- Обслуживание таблиц, секционированных через наследование

Обзор

PostgreSQL поддерживает базовое **секционирование (partitioning)** таблиц. Под секционированием понимается разбиение одной большой логической таблицы на несколько физических частей. Каждая часть содержит подмножество данных исходной таблицы. Выигрыш от таблиц со структурой, отражающей естественный характер запросов.

Секционирование может дать следующие преимущества:

- В определенных ситуациях производительность запросов может быть значительно улучшена. Часто запросы будут затрагивать только некоторые секции, а не всю таблицу. Если секции физически отражают распределение данных, которое требуется для некоторого запроса, то для выполнения этого запроса необходимо просканировать только часть индекса и таблицы.
- Когда в секцию можно загрузить или удалить целый набор строк, производительность может возрасти, если будет выполнена последовательная операция этой секции, а не поток по отдельному, связанному всю операцию по отдельности.
- Массовое загрузка и удаление данных также могут быть реализованы путем добавления или удаления секций, если [DROP TABLE](#) выполнено быстрее, чем [ALTER TABLE DETACH PARTITION](#), выполняется быстрее, чем массовая операция над таблицей.
- Редко используемые данные можно перенести на более дешевое и медленное хранилище.

Декларативное секционирование

PostgreSQL поддерживает следующие виды декларативного секционирования. Для каждого из них определяется набор допустимых ключей секционирования, которые могут состоять из одного или нескольких столбцов таблицы. Выражения не допускаются. Также должен существовать оператор принадлежности B-дерева для каждого столбца ключа секционирования, который будет использоваться в целях **секционирования**. Методы секционирования описаны в таблице ниже.

Методы секционирования

Метод	Описание
Секционирование по диапазонам	Таблица секционирована по диапазонам, определенным по ключевому столбцу или набору столбцов. Диапазоны не должны перекрываться друг с другом. Например, каждая секция может назначать диапазон для значений определенного столбца или набора столбцов, при этом не должно быть перекрывающихся значений в исключаемых диапазонах. Например, если ключ секции данных включает от 1 до 15, а другая секция — от 10 до 20, значения от 10 до 15 являются неоднозначными для обеих секций.
Секционирование по списку	Таблица разделяется с помощью списков, явно указывающих, какие значения ключа появляются в каждой секции.
Секционирование по хешу	Таблица секционируется по определенным модулям и остаткам, которые появляются для каждой секции. Каждая секция содержит строки, для которых хеш-значение ключевого столбца разделено на указанный модуль.

Если вашим приложениям требуется другая форма секционирования, можно добиться их реализации различными методами: [секционирование с использованием наследования](#). Также декларативное секционирование не обладает некоторыми возможностями, которые предоставляет наследование. Например, [PARTITION TABLE](#), описанные ниже.

Базовый синтаксис создания секций:

```
CREATE TABLE name_таблицы (
    [ FOR VALUES FROM {имя_значения_выражения} TO {имя_значения_выражения}
    | FOR VALUES IN {имя_значения_выражения}
    | FOR VALUES WITH (MODULUS remainder_модуля, REMAINDER значение_остатка) ]
)
```

Каждая секция должна быть создана командой таблицы, в которой она создана. Однако это отдельная таблица, доступная для отдельных операций. Вы можете добавить новую секцию в существующую секционированную таблицу или удалить секцию из секционированной таблицы. Массовые операции загрузки и удаления могут быть выполнены путем добавления или удаления секций.

Определенные преобразования обычных таблиц в секционированные возможны. Однако они включают добавление ограничений таблицы в качестве секции и перемещение данных между различными физическими местами хранения.

Секционирование по диапазонам

Разбиение по диапазонам часто используется по столбцам типа `timestamp` с [`date`](#).

Приведенная ниже структура может быть создана в соответствии с основным полем [`delivery_date`](#). Каждая дополнительная в функции будет создана по секции [`books_2022_01`](#), а строки с [`books_2022_02`](#) в строке — в секции [`books_2022_02`](#). Затем добавляется секция по умолчанию:

```
CREATE TABLE books (
    code        char(13) NOT NULL,
    title       varchar(40) NOT NULL,
    delivery_date  date NOT NULL,
    genre       varchar(10)
) PARTITION BY RANGE (delivery_date);

CREATE TABLE books_2022_01 PARTITION OF books FOR VALUES FROM ('2022-01-01') TO ('2022-02-01');
CREATE TABLE books_2022_02 PARTITION OF books FOR VALUES FROM ('2022-02-01') TO ('2022-03-01');
CREATE TABLE books_2022_03 PARTITION OF books FOR VALUES FROM ('2022-03-01') TO ('2022-04-01');
CREATE TABLE books_default PARTITION OF books DEFAULT;
```

Следующая строка добавляется в секцию [`books_2022_01`](#):

```
INSERT INTO books (code, title, delivery_date, genre) VALUES
    ('0-55-', 'Hyperion', '2022-01-01', 'sci-fi');
```

Ключи типа `timestamp` являются [`MINVALUE`](#) — [`MAXVALUE`](#), обозначающие минимальное и максимальное значения, содержащиеся в столбце, могут использоваться при указании диапазона. Например:

```
CREATE TABLE tabl_part1 PARTITION OF tabl FOR VALUES FROM (MINVALUE) TO (100);
CREATE TABLE tabl_part2 PARTITION OF tabl FOR VALUES FROM (100) TO (200);
CREATE TABLE tabl_part3 PARTITION OF tabl FOR VALUES FROM (200) TO (MAXVALUE);
```

Секционирование по списку

В приведенной ниже схеме, показанной [`genre`](#), используется список значений, которые появляются в каждой секции. Этот метод позволяет использовать несколько значений для одной секции. Поэтому в нем над создает секцию по умолчанию, которая для всех [`DEFAULT`](#) строк, которые для них жанров [`detective`](#) и [`mystery`](#):

```
CREATE TABLE books (
    code        char(13) NOT NULL,
    title       varchar(40) NOT NULL,
    delivery_date  date NOT NULL,
    genre       varchar(10) NOT NULL
) PARTITION BY LIST (genre);

CREATE TABLE books_sci_fi PARTITION OF books FOR VALUES IN ('sci-fi');
CREATE TABLE books_detective PARTITION OF books FOR VALUES IN ('detective', 'mystery');
CREATE TABLE books_default PARTITION OF books DEFAULT;
```

Секционирование по хешу

Вы можете использовать декларативное секционирование по хешу, если нужно равномерно распределить строки по секциям, а таблица не имеет подходящего ключа секционирования.

Пример:

```
CREATE TABLE books (
    code        char(13) NOT NULL,
    title       varchar(40) NOT NULL,
    delivery_date  date NOT NULL,
    genre       varchar(10) NOT NULL
) PARTITION BY HASH (code);

CREATE TABLE books_part1 PARTITION OF books FOR VALUES WITH (MODULUS 5, REMAINDER 1);
CREATE TABLE books_part2 PARTITION OF books FOR VALUES WITH (MODULUS 5, REMAINDER 2);
CREATE TABLE books_part3 PARTITION OF books FOR VALUES WITH (MODULUS 5, REMAINDER 3);
CREATE TABLE books_part4 PARTITION OF books FOR VALUES WITH (MODULUS 5, REMAINDER 4);
CREATE TABLE books_part5 PARTITION OF books FOR VALUES WITH (MODULUS 5, REMAINDER 5);
```

Обслуживание секций

Можно создавать индексы для столбцов ключа секционированной таблицы или любых других индексов, которые могут потребоваться. Индекс ключа секционированной таблицы не существует, но индексы могут быть созданы для отдельных секций. Вы можете создать уникальный индекс для каждой секции, а любые секции, которые должны иметь уникальные ключи. Индекс ключа секционирования создается, если [`delivery_date`](#) использовался в качестве ключа секционирования и создает следующий:

```
CREATE INDEX ON books (delivery_date);
```

> **Примечание**
>
> Для получения дополнительной информации о синтаксисе, параметрах и ограничениях [`CREATE INDEX`](#) должны быть равно [`ON`](#). Это включает использование по умолчанию в AUM). За дополнительной информацией обратитесь к разделу Полнотекстовый.

Вложенное секционирование

Таблицы секций могут быть также сами разделены. Например, можно добавить вложенную секцию в существующую секцию:

```
CREATE TABLE books_novel PARTITION OF books FOR VALUES IN ('novel');
CREATE TABLE books_detective PARTITION OF books FOR VALUES IN ('detective', 'mystery');
CREATE TABLE books_default PARTITION OF books DEFAULT PARTITION BY RANGE (delivery_date);
    CREATE TABLE books_oct21_2022_01 PARTITION OF books_oct FOR VALUES FROM ('2022-01-01') TO ('2022-01-31');
    CREATE TABLE books_oct21_2022_02 PARTITION OF books_oct FOR VALUES FROM ('2022-02-01') TO ('2022-02-31');
    CREATE TABLE books_oct21_2022_03 PARTITION OF books_oct FOR VALUES FROM ('2022-03-01') TO ('2022-03-31');
```

PostgreSQL вставит следующую строку в таблицу [`books_oct21_2022_01`](#):

```
INSERT INTO books (code, title, delivery_date, genre) VALUES
    ('0-14-', 'The time machine', date'2022-01-19', 'sci-fi');
```

Удаление секций

Любую секцию из данных можно исключить. Следующий оператор удаляет удаление секций, содержащих упорядоченные данные и обычно это самый эффективный способ удаления данных, которые больше не нужны. Иногда вы можете захотеть исключить операцию, но этот изменения структуры секций, и не физического перемещения большого объема данных.

Отключение [`PARTITION`](#) секции позволяет отключить [`INHERIT UPDATE EXCLUSIVE`](#) для отключенной таблицы. Рекомендуется создать ограничение [`CHECK`](#) для отключенной таблицы, которое будет соответствовать ограничению секции, как показано ниже. Таким образом, PostgreSQL может получить ограничение, необходимое для будущих операций с этой таблицей.

```
ALTER TABLE books ATTACH PARTITION books_2022_06
    FOR VALUES FROM ('2022-06-01') TO ('2022-07-01');
```

Включение [`PARTITION`](#) секции позволяет отключить [`INHERIT UPDATE EXCLUSIVE`](#) для отключенной таблицы. Рекомендуется создать ограничение [`CHECK`](#) для отключенной таблицы, которое будет соответствовать ограничению секции, как показано ниже. Таким образом, PostgreSQL может получить ограничение, необходимое для будущих операций с этой таблицей.

Ограничения

Использование декларативного таблиц накладывает следующие ограничения:

- Создать ограничение уникальности или первичного ключа для секционированной таблицы можно. Этого ключа столбцов не включают каждая выражения или вызовы функций и ограничение должно включать все столбцы ключа секционирования.
- Создать ограничение исключения, охватывающее всю секционированную таблицу, можно. Можно только поместить такое ограничение в отдельные секции с данными.
- Триггеры [`BEFORE ROW`](#) для [`INSERT`](#) могут изменить секцию, в которую в итоге попадет новая строка.
- Смешанное хранилище в постоянных таблицах в одном дереве секционирования не допускается.
- Секции не могут иметь столбцы, отсутствующие в родительской таблице.
- Отключение [`CHECK`](#) — [`NOT NULL`](#) ограничения в секционированную таблицу всегда игнорируются всех секциях. Также нельзя создать ограничение [`NOT NULL`](#) для столбца, если столбец не существует во всех секциях. Ограничения, которые применяются ко всей секционированной таблице, но отдельно применяются ко всем секциям, игнорируются будет очень низкой.

Секционирование с использованием наследования

Хотя декларативное секционирование покрывает для большинства задач, в некоторых случаях может использоваться полезным более гибкий подход. Секционирование может быть реализовано с помощью наследования таблиц, что позволяет реализовать такие функции, как различные определения столбцов для разных секций. Также наследование поддерживает несколько родительских таблиц и дополнительный таблиц.

Механизм наследования таблиц поддерживает множественное наследование.

Секционирование с использованием наследования полезно, например, для архивирования исторических данных. Старые записи удаляются и отключаются от основной секции, чтобы их можно было архивировать или обработать позже.

Пример

В этом примере создается структура секционирования, аналогичная приведенному выше примеру декларативного секционирования по диапазонам.

Создадим родительскую таблицу, от которой наследуют структуры для дочерних таблиц:

```
CREATE TABLE books (
    code        char(13) NOT NULL,
    title       varchar(40) NOT NULL,
    delivery_date  date NOT NULL,
    genre       varchar(10)
);
```

Создадим несколько дочерних таблиц, которые наследуются от родительской, и добавим непересекающиеся ограничения, чтобы определить допустимые значения ключей для каждой из них:

```
CREATE TABLE books_2022_01 (
CHECK ( delivery_date >= DATE '2022-01-01' AND delivery_date < DATE '2022-02-01' )
) INHERITS (books);

CREATE TABLE books_2022_02 (
CHECK ( delivery_date >= DATE '2022-02-01' AND delivery_date < DATE '2022-03-01' )
) INHERITS (books);

CREATE TABLE books_2022_03 (
CHECK ( delivery_date >= DATE '2022-03-01' AND delivery_date < DATE '2022-04-01' )
) INHERITS (books);
```

Для каждой дочерней таблицы создадим индекс для ключевого столбца, а также любые другие индексы, если они необходимы:

```
CREATE INDEX books_2022_01_delivery_date ON books_2022_01 (delivery_date);
CREATE INDEX books_2022_02_delivery_date ON books_2022_02 (delivery_date);
CREATE INDEX books_2022_03_delivery_date ON books_2022_03 (delivery_date);
```

PostgreSQL должен использовать эти таблицы для любых запросов, которые требуют данных в указанной таблице подходящих триггеров функции:

```
CREATE OR REPLACE FUNCTION books_select_part()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.delivery_date BETWEEN date'2022-01-01' AND date'2022-01-01' + 1 then
        INSERT INTO books_2022_01 VALUES (NEW.*);
    ELSIF NEW.delivery_date BETWEEN date'2022-02-01' and date'2022-02-01' + 1 then
        INSERT INTO books_2022_02 VALUES (NEW.*);
    ELSIF NEW.delivery_date BETWEEN date'2022-03-01' and date'2022-03-01' + 1 then
        INSERT INTO books_2022_03 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION 'This date not in your partitions. Add a partition.';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Создайте триггер, который вызывает функцию:

```
CREATE TRIGGER insert_books_trigger
    BEFORE INSERT ON books
    FOR EACH ROW EXECUTE FUNCTION books_select_part();
```

Теперь при попытке вставить строковые данные в дочернюю дочерней таблицы, если строка добавляется в нее еще нашел его, но для пустоты ты не реализовано [`TRIGGER`](#) для [`DELETE`](#) и [`UPDATE`](#).

> **Примечание**
>
> При использовании этого подхода, не рекомендуется отключать конфликтующий параметр [`constraint_exclusion`](#). В противном случае дочерние таблицы могут сканироваться, когда это не требуется. И AUM) начинается по умолчанию для всех запросов, включая [`partition`](#)

Обслуживание таблиц, секционированных через наследование

Удаление секции

Можно использовать команду [`DROP TABLE`](#), чтобы удалить дочернюю таблицу, которая больше не нужна:

```
DROP TABLE books_2022_01;
```

Или можно сначала отключить ее от родительской таблицы, не сохранять к ней доступ как к обычной таблице, используйте команду [`ALTER TABLE`](#):

```
ALTER TABLE books_2022_02 NO INHERIT books;
```

Добавление секции

Чтобы добавить новую дочернюю таблицу для обработки данных, создайте пустую дочернюю таблицу:

```
CREATE TABLE books_2022_04 (
CHECK ( delivery_date >= DATE '2022-04-01' AND delivery_date < DATE '2022-05-01' )
) INHERITS (books);
```

Также можно создать и заполнить новую дочернюю таблицу, прежде чем добавлять ее в иерархию таблиц. Такой подход позволяет загружать, проверять и преобразовывать данные до того, как они появятся в базе данных.

```
CREATE TABLE books_2022_04
    ELSIF books INCLUDING DEFAULTS INCLUDING CONSTRAINTS);
    CHECK ( delivery_date >= DATE '2022-04-01' AND delivery_date < DATE '2022-05-01' )
) INHERITS (books);

ALTER TABLE books_2022_04 INHERIT books;
```

После добавления новой секции обновите функцию триггера:

```
CREATE OR REPLACE FUNCTION books_select_part()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.delivery_date BETWEEN date'2022-01-01' AND date'2022-01-01' + 1 then
        INSERT INTO books_2022_01 VALUES (NEW.*);
    ELSIF NEW.delivery_date BETWEEN date'2022-02-01' and date'2022-02-01' + 1 then
        INSERT INTO books_2022_02 VALUES (NEW.*);
    ELSIF NEW.delivery_date BETWEEN date'2022-03-01' and date'2022-03-01' + 1 then
        INSERT INTO books_2022_03 VALUES (NEW.*);
    ELSIF NEW.delivery_date BETWEEN date'2022-04-01' and date'2022-04-01' + 1 then
        INSERT INTO books_2022_04 VALUES (NEW.*);
    ELSE
        RAISE EXCEPTION 'This date not in your partitions. Add a partition.';
    END IF;
    RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

Ограничения

Ограничения секционирования через наследование имеют следующие ограничения:

- PostgreSQL не может проверить автоматически, являются ли все ограничения [`CHECK`](#) взаимно исключающими. Поэтому безопаснее создать код для формирования дочерних таблиц и ограничений или изменить объекты, как долго это нужно.
- Индексы и внешние ключи относятся к родительской таблице, но не к ее версиям наследования.
- В приведенных выше примерах показана логика case, это значения изменения столбца ключа можно изменить, что приведет к перемещению строки в подходящую дочернюю таблицу. Если вы хотите избежать такого поведения, вставьте [`CHECK`](#) ограничение, которое не позволит изменить ключ.
- Если вы используете команды [`VACUUM`](#) или [`ANALYZE`](#), нужно вручную обработать каждую дочернюю таблицу.
- Операторы [`INSERT`](#) с конфликтами [`ON CONFLICT`](#) не будут работать ожидаемым образом, если для [`ON CONFLICT`](#) выполняется только в случае нарушений уникальности и указанный индекс, а не ограничения.
- Необходимую создавать триггеры или правила для перенаправления строк в подходящую дочернюю таблицу. Триггеры сложно написать и медленно, как недрогнувшие, выполняются при декларативном секционировании.

Внешние таблицы

Д. Евгений Кукша

Содержание

Назад к содержанию

Обзор
Примеры
- Доступ к данным, хранящимся на сторонних серверах PostgreSQL
- Доступ к файлам в файловой системе сервера

Обзор

PostgreSQL частично реализует спецификацию SQL/MED. Это позволяет использовать SQL запросы для доступа к данным, которые находятся за пределами кластера PostgreSQL. Эти данные называются внешними. Чтобы получить к ним доступ, выполните следующие действия:

1. Используйте **обёртку внешних данных (foreign data wrapper)**. Обёртка внешних данных подключается к внешним источникам и получает от них данные. Расширение contrib, включённое в сборку PostgreSQL содержит несколько обёрток внешних данных. За дополнительной информацией обратитесь к статье Additional Supplied Modules. Также можно использовать сторонние обёртки, если они соответствуют вам. Например, Multicorn и Foreign data wrappers.
2. Создайте **внешний сервер (foreign server)**, который инкапсулирует информацию о соединении для соответствующего сервера внешних данных. Используйте для этого команду CREATE SERVER.

```
CREATE SERVER [ IF NOT EXISTS ] name_сервера [ TYPE 'тип_сервера' ] [ VERSION 'версия_сервера' ]
FOREIGN DATA WRAPPER name_обертки
[ OPTIONS ( опция 'значение' [, ... ] ) ]
```

Параметры команды CREATE SERVER

IF NOT EXISTS	Если указан этот параметр, PostgreSQL не выдаст ошибку, если сервер с указанным именем существует. Не гарантируется, что существующий объект соответствует тому, который мог бы быть создан
name_сервера	Имя создаваемого внешнего сервера. Имя сервера должно быть уникальным в рамках базы данных
тип_сервера	Необязательный параметр, определяющий тип сервера, потенциально полезный для обёрток внешних данных
версия_сервера	Необязательный параметр, указывающий версию сервера, потенциально полезный для обёрток внешних данных
name_обертки	Имя обёртки внешних данных, которая управляет внешним сервером
OPTIONS (опция 'значение' [, ...])	Указывает параметры сервера. Эти параметры определяют детали подключения, но их имена и значения зависят от обёртки внешних данных

Пользователь, создавший сервер, становится его владельцем. Чтобы создать сервер, пользователь должен иметь привилегию USAGE для указанной обёртки внешних данных.

2. Для доступа к удалённым данным может потребоваться использовать внешние источники данных. Создайте объект **user mapping**, используемый для доступа пользователя к внешнему серверу. Он определяет, какую роль пользователя PostgreSQL использовать для подключения к внешнему серверу. Для этого воспользуйтесь командой CREATE USER MAPPING со следующим синтаксисом:

```
CREATE USER MAPPING [ IF NOT EXISTS ] FOR { имя_пользователя> | USER | CURRENT_ROLE | CURRENT_USER | PUBLIC }
  SERVER name_сервера>
  [ OPTIONS ( опция 'значение' [, ... ] ) ]
```

Параметры команды CREATE USER MAPPING

IF NOT EXISTS	Если указан этот параметр, PostgreSQL не выдаст ошибку, если объект user mapping для указанного пользователя для подключения к указанному серверу существует. Не гарантируется, что существующий объект соответствует тому, который мог бы быть создан
name_пользователя>	Имя существующего пользователя, для соответствия с пользователем, на внешнем сервере. `CURRENT_ROLE`, `CURRENT_USER` и `USER` соответствуют имени текущего пользователя. Когда указано `PUBLIC`, создается так называемое общее сопоставление, которое будет использоваться всеми пользователями системы для подключения
name_сервера	Определяет имя существующего сервера, для которого создается объект user mapping
OPTIONS (опция 'значение' [, ...])	Указывает параметры объекта user mapping. Параметры обычно определяют фактическое имя пользователя и пароль на стороннем сервере. Допустимые имена и значения параметров зависят от обёртки внешних данных

Владелец внешнего сервера может создавать объекты user mapping для этого сервера для любого пользователя. Пользователь также может создать user mapping для себя, если ему предоставлена привилегия USAGE на внешний сервер.

3. Создайте одну или несколько **внешних таблиц (foreign tables)**, определяющих структуру внешних данных. Внешняя таблица не хранит данные на сервере PostgreSQL, но к ней можно обращаться с помощью запросов из кластера. PostgreSQL использует обёртку внешних данных для извлечения данных из сторонних источников и передачи данных в этот источник в случае команд обновления.

Выполните команду CREATE FOREIGN TABLE, чтобы создать внешнюю таблицу:

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] имя_таблицы> ( [
  { имя_столбца> тип_данных> [ COLLATE <правило_сортировки> ] [ <ограничение_столбца> [ ... ] ] | <ограничение_таблицы> }
  [, ... ]
] )
[ INHERITS ( <таблица_родитель> [, ... ] ) ]
  SERVER name_сервера>
  [ OPTIONS ( опция 'значение' [, ... ] ) ]

CREATE FOREIGN TABLE [ IF NOT EXISTS ] <таблица_имен>
  PARTITION OF <главная_таблица> [ (
  { имя_столбца> [ WITH OPTIONS ] [ <ограничение_столбца> [ ... ] ]
    | <ограничение_таблицы> }
    [, ... ]
) ]
{ FOR VALUES <границы_секции> | DEFAULT }
  SERVER name_сервера>
  [ OPTIONS ( опция 'значение' [, ... ] ) ]
```

Параметры команды CREATE FOREIGN TABLE

IF NOT EXISTS	Если указан этот параметр, PostgreSQL не выдаст ошибку, если таблица с указанным именем существует. Не гарантируется, что существующий объект соответствует той таблице, которая могла бы быть создана
name_таблицы	Имя таблицы, может включать имя схемы (`имя_схемы`.`имя_таблицы`)
имя_столбца	Имя столбца, который должен быть создан в новой таблице
тип_данных	Тип данных столбца, также может включать указание массива с этим типом. За дополнительными сведениями о типах данных обратитесь к статье Data types
COLLATE <правило_сортировки>	Назначает правило сортировки для столбца (которой должно иметь тип, поддерживающий сортировку). Если правило не указано, используется правило сортировки по умолчанию, установленное для типа данных столбца
INHERITS (таблица_родитель> [, ...])	Указывает список таблиц, от которых новая таблица автоматически наследует все столбцы. Родительские таблицы могут быть обычными таблицами или внешними таблицами. За дополнительной информацией обратитесь к статье Наследование
PARTITION OF <главная_таблица>	Создаёт внешнюю таблицу как секцию указанной главной таблицы с указанными значениями границы секции. Обратитесь к статье Декларативное секционирование таблиц за дополнительной информацией. Обратите внимание, что нельзя создавать внешнюю таблицу как раздел главной таблицы, если в главной таблице есть внешний `UNIQUE`
<ограничение_столбца>	Ограничение столбца, которое описано в разделе Использование ограничений
<ограничение_таблицы>	Ограничение таблицы, которое описано в разделе Использование ограничений
name_сервера	Определяет имя существующего внешнего сервера, используемого для внешней таблицы
OPTIONS (опция 'значение' [, ...])	Указывает параметры, связанные с внешней таблицей или одним из её столбцов. Допустимые имена и значения параметров зависят от обёртки внешних данных

Кроме того, вы можете импортировать внешнюю схему целиком. Для этого выполните команду IMPORT FOREIGN SCHEMA. Она создаст внешние таблицы, структурированные подобно существующим внешним таблицам. PostgreSQL импортирует внешние таблицы с определенными столбцов и параметрами, которые существуют в таблицах со стороннего сервера. Пользователь, выполняющий команду `IMPORT FOREIGN SCHEMA`, становится владельцем внешних таблиц. Команда имеет следующий синтаксис:

```
IMPORT FOREIGN SCHEMA <внешняя_схема>
  [ { LIMIT TO | EXCEPT } ( <имя_таблицы> [, ...] ) ]
  FROM SERVER name_сервера
  INTO <локальная_схема>
  [ OPTIONS ( опция 'значение' [, ... ] ) ]
```

Параметры команды IMPORT FOREIGN SCHEMA

<внешняя_схема>	Указывает внешнюю схему для импорта. Что именно представляет собой внешняя схема, зависит от применяемой обёртки внешних данных
LIMIT TO (имя_таблицы> [, ...])	Импортирует только внешние таблицы с заданными именами. Другие таблицы существующие во внешней схеме, будут проигнорированы
EXCEPT (имя_таблицы> [, ...])	Исключает из импорта указанные внешние таблицы. Данная команда импортирует все таблицы, существующие во внешней схеме, за исключением перечисленных в этом выражении
<имя_сервера>	Определяет внешний сервер для импорта
<локальная_схема>	Указывает схему, в которой будут созданы импортированные таблицы
OPTIONS (опция 'значение' [, ...])	Указывает параметры, которые будут использоваться во всех импортах. Допустимые имена и значения параметров зависят от обёртки внешних данных

Список импортируемых таблиц может быть ограничен определенным подмножеством, либо отдельные таблицы могут быть исключены из импорта. Локальная схема для импорта существовать на момент выполнения команды IMPORT FOREIGN SCHEMA.

Чтобы выполнить `IMPORT FOREIGN SCHEMA`, пользователь должен иметь привилегию `USAGE` для внешнего сервера и привилегию `CREATE` для целевой локальной схемы.

Чтобы отобразить внешние таблицы, существующие в текущей базе данных, используйте следующий запрос:

```
SELECT * FROM information_schema.foreign_tables;
```

Примеры

Расширение contrib содержит следующие обёртки внешних данных:

- `postgres_fdw` – используется для доступа к данным, хранящимся на сторонних серверах PostgreSQL. Модуль описан в статье postgres_fdw.
- `file_fdw` – используется для доступа к файлам данных в файловой системе сервера или для выполнения программ на сервере и чтения их вывода. Модуль описан в статье file_fdw.

Примеры использования этих обёрток приведены ниже.

Доступ к данным, хранящимся на сторонних серверах PostgreSQL

Предположим, что внешний сервер PostgreSQL содержит таблицы `books` и `authors` в базе данных `books_store`.

Таблица `authors` создана следующим образом:

```
CREATE TABLE authors (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);
```

Чтобы получить доступ к данным с внешнего сервера, выполните следующие действия:

1. Используйте существующую роль на удалённом сервере или создайте нового пользователя и предоставьте ему соответствующие права. Например:

```
CREATE ROLE user1 WITH PASSWORD 'password' LOGIN;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO user1;

GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO user1;
```

2. Поскольку ADPG включает расширение contrib, нужно просто выполнить команду `CREATE EXTENSION` на локальном сервере, чтобы использовать обёртку `postgres_fdw`:

```
CREATE EXTENSION postgres_fdw;
```

3. Создайте внешний сервер и свой локальный base данных. Следующий код создает сервер, который подключается к base данных `books_store`, расположенной на хосте `10.92.4.229`:

```
CREATE SERVER server1 FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host '10.92.4.229', port '5432', dbname 'books_store');
```

Если удалённый сервер использует стандартный порт, `port` можно опустить.

4. Создайте объект user mapping. Передайте учетные данные пользователя, которого вы создали на удаленном сервере, в качестве параметров. Следующий код создает user mapping для пользователя `postgres` с локального сервера:

```
CREATE USER MAPPING FOR postgres SERVER server1 OPTIONS (user 'user1', password 'password');
```

5. Создайте внешнюю таблицу на локальном сервере:

```
CREATE FOREIGN TABLE authors (
  id int,
  name VARCHAR(100) NOT NULL
)
SERVER server1;
```

Проверьте результат:

```
SELECT * FROM authors;
```

Вывод команды:

```
 id |      name
----+-----------------
  1 | Virginia Woolf
  2 | Harper Lee
  3 | F. Scott Fitzgerald
  4 | J.R.R. Tolkien
  5 | George Orwell
```

Также можно импортировать схему целиком:

```
CREATE SCHEMA schema1;
IMPORT FOREIGN SCHEMA public FROM SERVER server1 into schema1;
```

Отобразить существующие внешние таблицы, чтобы проверить результат:

```
SELECT * FROM information_schema.foreign_tables;
```

Результат:

```
foreign_table_catalog | foreign_table_schema | foreign_table_name | foreign_server_catalog | foreign_server_name
----------------------+----------------------+--------------------+------------------------+--------------------
postgres              | public               | authors            | postgres               | server1
postgres              | schema1              | authors            | postgres               | server1
postgres              | schema1              | books              | postgres               | server1
```

Таблицы `books` и `authors` были импортированы в схему `schema1`.

Доступ к файлам в файловой системе сервера

Предположим, у нас есть следующий файл books.csv:

```
2,Mrs. Dalloway,1,1925,novel
3,To the Lighthouse,1,1927,novel
3,To Kill a Mockingbird,2,1960,novel
4,The Lord of the Rings,4,1954,fantasy
5,1984,5,1949,sci-fi
```

Нам нужно использовать SQL для доступа к данным из файла. Для решения этой задачи выполните следующие шаги:

1. Выполните команду `CREATE EXTENSION`, чтобы использовать обёртку внешних данных `file_fdw`:

```
CREATE EXTENSION file_fdw;
```

2. Создайте внешний сервер:

```
CREATE SERVER file_server FOREIGN DATA WRAPPER file_fdw;
```

3. Создайте внешнюю таблицу таким образом, чтобы ее структура совпадала со структурой файла. Передайте путь к файлу и формат в выражении `OPTIONS`:

```
CREATE FOREIGN TABLE foreign_books (
  id int,
  title VARCHAR(255),
  author_id INT,
  publish_year INT,
  genre VARCHAR(50)
) SERVER file_server
OPTIONS ( filename '/var/lib/pgsql/books.csv', format 'csv' );
```

Теперь вы можете использовать команды SQL для доступа к данным, например:

```
SELECT * FROM foreign_books;
```

Результат:

```
 id |      title      | author_id | publish_year |  genre
----+-----------------+-----------+--------------+---------
  1 | Mrs. Dalloway   |         1 |         1925 | novel
  2 | To the Lighthouse |       1 |         1927 | novel
  3 | To Kill a Mockingbird | 2 |         1960 | novel
  4 | The Lord of the Rings | 4 |         1955 | fantasy
  5 | 1984            |         5 |         1949 | sci-fi
```

TOAST

Евгения Кузина

Содержание

Назад к содержанию

Обзор

Отдельное размещение TOAST на диске

Отдельное размещение TOAST в памяти

Обзор

В PostgreSQL используется фиксированный размер страницы (значение по умолчанию — 8 КБ), также кортежи не могут занимать несколько страниц, поэтому невозможно напрямую хранить большие значения полей. PostgreSQL сжимает большие значения и/или разбивает их на несколько физических строк. Такие строки сохраняются в специальное хранилище. Этот метод известен как **TOAST** (The Oversized-Attribute Storage Technique). PostgreSQL также использует механизм TOAST для улучшения обработки данных большого размера в памяти.

Только определенные типы данных поддерживают TOAST (например, text, json, jsonb и bytea). Чтобы поддерживать TOAST, тип данных должен представлять значение переменной длины (varlena), в котором первое четырехбайтовое слово хранящегося значения содержит общую длину значения в байтах (включая само это слово). Содержание оставшейся части значения TOAST не ограничивает. Алгоритмы TOAST работают, интерпретируя первое слово. Таким образом, функции уровня C, работающие с типом данных, поддерживающим TOAST, должны аккуратно обращаться со входными значениями, которые могут быть в TOAST-формате. Входные данные могут не состоять из четырехбайтового слова длины и содержимого после него, пока не будут распакованы. Обычно в таких случаях нужно использовать макрос `PG_DETOAST_DATUM` прежде чем что-либо делать с входным значением, но в некоторых ситуациях возможны и более эффективные подходы. За подробностями обратитесь к статье TOAST Considerations.

TOAST использует два бита слова длины varlena для дополнительных настроек, поэтому длина TOAST-значения ограничена 1 ГБ. Когда оба бита равны нулю, значение является обычным значением не в TOAST-формате и оставшиеся биты слова длины задают общий размер данных в байтах. Когда установлен старший (или младший, в зависимости от архитектуры) бит, значение имеет однобайтовый заголовок вместо четырехбайтового, а оставшиеся биты этого байта задают общий размер данных в байтах. Этот подход поддерживает компактное хранение значений короче 127 байт, но при этом позволяет при необходимости расширить значение этого типа данных до 1 ГБ. В особом случае, если все оставшиеся биты однобайтового заголовка равны нулю, значение представляет собой TOAST-указатель на отдельно размещенные данные, с несколькими возможными вариантами, описанными ниже. Код, хранящийся во втором байте, определяет тип и размер TOAST-указателя. Если старший (или младший, в зависимости от архитектуры) бит очищен, а соседний бит установлен, содержимое данных хранится в упакованном виде и должно быть распаковано перед использованием. В этом случае оставшиеся биты четырехбайтового слова длины задают общий размер сжатых, а не исходных данных. Сжатие также возможно и для данных, хранимых отдельно, но заголовок varlena не говорит, имеет ли оно место — это определяется содержимым TOAST-указателя.

Вы можете выбрать метод сжатия для каждого столбца, который будет применяться при внутреннем и внешнем хранении. Для этого необходимо установить параметр `COMPRESSION` в команде `CREATE TABLE` или `ALTER TABLE`. Параметр default_toast_compression определяет значение по умолчанию, если `COMPRESSION` не установлен.

Существуют разные типы TOAST-указателей:

- Самый старый и наиболее распространенный тип — это указатель на **отделенные данные на диске**, хранящиеся в отдельной TOAST-таблице, связанной с таблицей, содержащей TOAST-указатель. Такой указатель создается кодом обработки TOAST, когда кортеж, сохраняемый на диск, оказывается слишком большим.
- TOAST-указатель может ссылаться на **отделенные данные в памяти**. Такие данные недолговечны и не могут быть сохранены на диск. Этот тип указателей очень полезен, чтобы избежать избыточного копирования и обработки данных большого размера.

Отдельное размещение TOAST на диске

Если какие-либо столбцы таблицы хранятся в TOAST-формате, с таблицей связана TOAST-таблица. `OID` TOAST-таблицы сохраняется в значении `pg_class.reltoastrelid` основной таблицы. Все TOAST-значения из главной таблицы помещаются в эту TOAST-таблицу.

PostgreSQL делит значения для внешнего хранения (после сжатия, если оно используется) на фрагменты размером не более `TOAST_MAX_CHUNK_SIZE` байт. Значение по умолчанию для этого параметра составляет около 2000 байт. Таким образом, четыре строки фрагментов помещаются на странице. Каждый фрагмент хранится в виде отдельной строки в TOAST-таблице. Каждая TOAST-таблица имеет следующие столбцы:

- `chunk_id` — OID, идентифицирующий конкретное TOAST-значение;
- `chunk_seq` — последовательный номер для фрагмента внутри значения;
- `chunk_data` — непосредственно данные фрагмента.

Уникальный индекс по полям `chunk_id` и `chunk_seq` обеспечивает быстрый поиск значений. TOAST-указатель содержит `OID` связанной TOAST-таблицы и `OID` конкретного значения (его `chunk_id`). Указатель также хранит логический размер данных (исходную длину несжатых данных), физический сохраненный размер (отличается, если применяется сжатие) и метод сжатия, если он указан. Общий размер TOAST-указателя составляет 18 байт независимо от фактического размера представляемого значения.

Код обработки TOAST срабатывает, только когда значение строки, которое должно храниться в таблице, по размеру больше, чем `TOAST_TUPLE_THRESHOLD` байт (обычно это 2 КБ). Код TOAST будет сжимать и/или выносить значения поля за пределы таблицы до тех пор, пока значение строки не станет меньше `TOAST_TUPLE_TARGET` байт (переменная величина, так же обычно 2 КБ) или уменьшить объем станет невозможно. Во время операций `UPDATE` значения неизменных полей обычно сохраняются как есть, поэтому модификация строки с отдельно хранимыми значениями не несет издержек, связанных с TOAST, если все TOAST-значения остаются без изменений.

Код обработки TOAST распознает четыре различные стратегии хранения столбцов, совместимых с TOAST, на диске:

- `PLAIN` — не допускает ни сжатия, ни отдельного хранения. Он также отключает использование однобайтовых заголовков для типов varlena. Это единственная возможная стратегия для столбцов типов данных, не поддерживающих TOAST.
- `EXTENDED` — допускает как сжатие, так и внешнее хранение. Это значение по умолчанию для большинства типов данных, поддерживаемых TOAST. PostgreSQL сначала использует сжатие, а затем отдельное хранение, если строка все еще слишком велика.
- `EXTERNAL` — допускает отдельное хранение, но не сжатие. Использование `EXTERNAL` ускоряет операции над частями строк в столбцах типа text и bytea (ценой увеличения объема памяти для хранения), так как эти операции оптимизированы для извлечения только требуемых частей отделенного значения, когда оно не сжато.
- `MAIN` — допускает сжатие, но не внешнее хранение. Для таких столбцов по-прежнему будет выполняться внешнее хранение, в том случае, если нет другого способа сделать строку достаточно маленькой, чтобы она поместилась на странице.

Каждый тип данных, совместимый с TOAST, определяет стандартную стратегию для столбцов этого типа данных, но стратегия для заданного столбца таблицы может быть изменена с помощью команды ALTER TABLE … SET STORAGE.

Вы также можете использовать `ALTER TABLE … SET (toast_tuple_target = N)`, чтобы изменить `TOAST_TUPLE_TARGET` для каждой таблицы.

Эта схема имеет ряд преимуществ по сравнению с более простым подходом, например, позволяющим значениям строк занимать несколько страниц. Так как запросы обычно выполняются путем сравнения относительно небольших значений ключа, большая часть работы будет выполняться с использованием строк основной таблицы. Большие TOAST-значения извлекаются только во время отправки результата клиенту. Основная таблица получается намного меньше, и в общий буферный кеш помещается больше её строк, чем их было бы без использования отдельного хранения. Наборы данных для сортировок также уменьшаются, а сортировки чаще будут выполняться исключительно в памяти.

Отдельное размещение TOAST в памяти

Указатели TOAST также могут указывать на данные, расположенные в памяти текущего серверного процесса. В настоящее время есть два типа указателей:

- **Косвенный TOAST-указатель** ссылается на значение varlena, хранящееся в памяти. В настоящее время применяется при логическом декодировании, чтобы не приходилось создавать физические кортежи больше одного 1 ГБ, что может потребоваться при консолидации всех отделенных значений полей в одном кортеже. Данный вариант имеет ограниченное применение, так как целевые данные будут существовать, только пока существует указатель, и никакой инфраструктуры для их сохранения нет.
- **Указатели на развернутые TOAST-данные** полезны для сложных типов, представление которых на диске плохо приспособлено для вычислительных целей. Например, стандартное представление в виде varlena массива PostgreSQL включает информацию о размерности, битовую карту элементов NULL (если они в нем содержатся), а затем значения всех элементов по порядку. Когда элемент сам по себе имеет переменную длину, единственный способ найти элемент — это просканировать все предыдущие элементы. Это представление компактно и поэтому подходит для хранения на диске, но для вычислительной обработки массива гораздо удобнее иметь "развернутое" или "деконструированное" представление, в котором можно определить начальные адреса всех элементов. Механизм указателей TOAST способствует решению этой задачи, допуская передачу по ссылке элемента datum как указателя на стандартное значение varlena (представление на диске) или TOAST-указателя на развернутое представление где-то в памяти. Детали развернутого представление определяются самим типом данных, хотя оно может иметь стандартный заголовок и удовлетворять другим требованиям API, описанным в Declarations for access to "expanded" value representations.

TOAST-указатели на развернутые значения далее подразделяются на указатели для чтения/записи и указатели только для чтения. Представление, на которое они указывают, одинаковое, но функции, получающей указатель для чтения/записи, разрешается модифицировать целевые данные. Тогда как функция, получающая указатель только для чтения, должна сначала сделать копию, чтобы получить измененную версию значения. Это отличие и связанные с ним соглашения позволяют избежать излишнего копирования развернутых значений при выполнении запросов.

Для всех типов указателей TOAST на данные в памяти, код обработки TOAST гарантирует, что такие данные не окажутся случайно сохранены на диске. Указатели TOAST в памяти автоматически сворачиваются в обычные значения varlena перед сохранением — а затем могут преобразоваться в указатели TOAST на диске, если без этого не смогут уместиться в содержащем их кортеже.

Табличные пространства

Евгения Кузина

Назад к содержанию

Табличные пространства определяют физическое размещение данных в файловой системе.

Табличные пространства можно создавать на разных разделах диска или на разных дисках. Например, часто используемый индекс можно разместить на быстром и надежном SSD-диске, а таблица с архивными данными, которые редко используются, может быть размещена на более медленном носителе. Также при нехватке места в разделе, на котором был инициализирован кластер, табличное пространство можно создать в другом разделе.

ВАЖНО

Несмотря на возможное внешнее размещение относительно основного каталога хранения данных, табличные пространства являются частью кластера и не могут рассматриваться как самостоятельная коллекция файлов данных. Они зависят от метаданных, расположенных в главном каталоге, и не могут быть подключены к другому кластеру или копироваться самостоятельно. Также в случае потери табличного пространства (например, при удалении файлов или сбое диска) кластер может оказаться недоступным или не сможет запуститься. При размещении табличного пространства во временной файловой системе (например, RAM-диске) возникает угроза надежности всего кластера.

Чтобы определить табличное пространство, используйте команду CREATE TABLESPACE:

```
CREATE TABLESPACE tablespace1 LOCATION '/home/user1/tablespace1_data';
```

Где `/home/user1/tablespace1_data` — каталог, в котором будут размещаться файлы объектов, принадлежащих этому табличному пространству. Он должен быть уже создан, принадлежать пользователю операционной системы, под которым запущен ADPG и быть пустым.

Также для конкретного табличного пространства можно переопределить предполагаемую стоимость чтения страниц, используемую планировщиком, и характеристики предварительной выборки во время выполнения запросов, заданные параметрами конфигурации. Это полезно, если табличное пространство расположено на диске, который быстрее или медленнее других. Доступные для переопределения параметры: seq_page_cost, random_page_cost, effective_io_concurrency и maintenance_io_concurrency. Используйте выражение `WITH`, чтобы установить эти параметры, например:

```
CREATE TABLESPACE tablespace2 LOCATION '/home/user1/tablespace2_data' WITH (seq_page_cost=0.8, random_page_cost=6.0);
```

ADPG работает с данными на дисках только через файловые системы, наиболее популярные EXT3, EXT4 и XFS. Raw devices не поддерживаются. Хорошей практикой является создание одного табличного пространства на одну логическую файловую систему, так как нет возможности контролировать расположение отдельных файлов в файловой системе.

Создание табличного пространства должно выполняться от имени суперпользователя базы данных. После этого можно разрешить его использование обычным пользователям, предоставив им привилегию `CREATE` командой GRANT.

Таблицы, индексы и базы данных могут быть размещены в определенном табличном пространстве. Для этого пользователь с привилегией `CREATE` для данного табличного пространства должен передать имя табличного пространства в качестве параметра соответствующей команде (`CREATE` или `ALTER`).

Например, следующий код создает таблицу `t1` в табличном пространстве `tablespace1`:

```
CREATE TABLE t1 (c1 integer) TABLESPACE tablespace1;
```

Команды, приведенные ниже, переносят таблицу `t1` и базу данных `database1` в табличные пространства `tablespace2` и `tablespace3` соответственно:

```
ALTER TABLE t1 SET TABLESPACE tablespace2;

ALTER DATABASE database1 SET TABLESPACE tablespace3;
```

Вы можете использовать параметр default_tablespace, чтобы задать табличное пространство, которое будет использоваться по умолчанию в командах `CREATE TABLE` и `CREATE INDEX`:

```
SET default_tablespace = tablespace2;
```

Вы также можете добавить этот параметр в разделе ***ADPG configurations*** на странице ***Clusters → ADPG cluster → Services → ADPG → Primary configuration*** в пользовательском интерфейсе ADCM. За дополнительной информацией обратитесь к статье Конфигурационные параметры.

Кроме этого, существует параметр temp_tablespaces, который указывает на пространства для размещения временных таблиц и индексов, а также файлов, создаваемых при операциях сортировки больших наборов данных. В качестве значения этого параметра предпочтительнее 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.

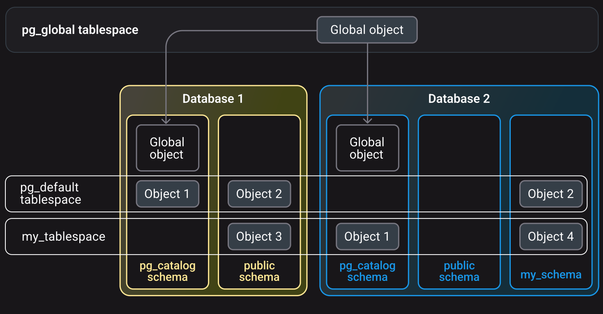
При установке кластера создаются два табличных пространства, они описаны в таблице ниже.

Табличные пространства по умолчанию

Название	Описание
pg_default	Используется по умолчанию для баз данных, а также шаблонов `template1` и `template0`, располагается в каталоге */pg_data1/adpg16/base*. Внутри этого каталога объекты размещаются в подкаталогах баз данных
pg_global	В этом табличном пространстве размещаются общие системные объекты всего кластера. Располагается в каталоге */pg_data1/adpg16/global*. На логическом уровне организации данных схема `pg_catalog` содержит объекты из пространства pg_global

Первая часть пути к каталогам (*/pg_data1*) определяется конфигурационным параметром `Data directory`.

Объекты одной базы данных могут быть размещены в нескольких табличных пространствах. В одном табличном пространстве могут быть размещены объекты нескольких баз данных.



Физический уровень организации данных в кластере ADPG

Чтобы получить список табличных пространств кластера, используйте метакоманды psql `\db` и `\db+`. Также можно сделать запрос к системному каталогу `pg_tablespace`:

```
SELECT * FROM pg_tablespace;
```

Поскольку табличное пространство можно использовать в рамках любой базы данных, его невозможно удалить до тех пор, пока не будут удалены все объекты, использующие это пространство. Для удаления пустого табличного пространства вызовите команду DROP TABLESPACE:

```
DROP TABLESPACE tablespace2;
```

PostgreSQL использует символические ссылки для упрощения реализации табличных пространств. Соответственно, табличные пространства могут использоваться только в системах, поддерживающих символические ссылки. Каталог */pg_data1/adpg16/pg_tblspc* содержит символические ссылки, которые указывают на внешние табличные пространства кластера.

Представление и материализованные представления

Открыть с оглавлением

Содержание

- Создание представления
- Удаление представления
- Материализованные представления
- Изменение данных представления
- Удаление материализованного представления
- Сравнение представлений и материализованных представлений
- Получение информации о представлениях и материализованных представлениях
- Системные представления

Представления

Представление - это именованный запрос, который при его вызове выполняется для создания набора строк, возвращаемых представлением. Представления логически основаны на результатах запросов SELECT. Представления могут участвовать в запросах SELECT, INSERT, UPDATE и DELETE, но ограничения на то, как определенное представление может быть обновлено, не допускают произвольных мутаций. Представления - это инструмент, который помогает вам структурировать ваш SQL-код, инкапсулировать повторяющийся код, изменять интерфейс данных и ограничивать доступ к базовым таблицам.

```
SELECT * FROM pg_views WHERE schemaname = 'public';
SELECT * FROM information_schema.views WHERE table_schema = 'public';
```

Создание представления

Представления создаются с помощью команды **CREATE VIEW**. В самом простом случае, она дает представлению имя и определяет запрос, который будет выполняться при обращении к представлению:

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ...] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

OR REPLACE - если представление с таким именем уже существует, оно будет заменено. Новый запрос должен генерировать те же столбцы, что и существующий запрос (то есть те же имена столбцов в том же порядке и с теми же типами данных), но он может добавлять дополнительные столбцы в конец списка. Вычисления, стоящие за представлением, могут быть изменены произвольным образом.

TEMP | TEMPORARY - если указано, представление создается как временное представление. Временные представления автоматически удаляются в конце текущего сеанса. Существующие постоянные отношения с тем же именем не видны текущему сеансу, пока временное представление существует, если только они не называются схема-квалифицированными. Любые представления, созданные другими представлениями или правилами, также должны быть временными.

RECURSIVE - создает рекурсивное представление. Синтаксис эквивалентен CREATE VIEW name AS WITH RECURSIVE name (column_name, ...) AS (query) SELECT column_name, ... FROM name;. Для рекурсивного представления должен быть указан список имен столбцов, и этот список должен быть написан ровно один раз.

name - имя создаваемого представления, может быть схема-квалифицированным.

column_name - необязательный список имен, используемых в качестве имен столбцов представления. Если не указано, имена столбцов выводятся из запроса.

WITH (view_option_name [= view_option_value] [, ...]) - этот необязательный предложение WITH устанавливает необязательные параметры для представления. В настоящее время поддерживаются следующие параметры:

- **check_option** (enum) - управляет поведением автоматически обновляемых представлений. Допустимые значения: local и cascaded.
- **security_barrier** (boolean) - должен использоваться, если представление предназначено для обеспечения безопасности на уровне строк.
- **security_invoker** (boolean) - если true, проверки прав доступа к базовым таблицам выполняются с правами пользователя, вызывающего представление, а не с правами владельца представления.

query - команда SELECT, которая будет выполнена при обращении к представлению.

WITH [CASCADED | LOCAL] CHECK OPTION - управляет поведением автоматически обновляемых представлений. Если указано, INSERT и UPDATE команды для представления будут проверять, что новые строки удовлетворяют условию, определенному представлением (то есть, что новые строки проходят через условие WHERE представления). Если CHECK OPTION не указано, INSERT и UPDATE команды будут иметь возможность создавать строки, которые не видны через представление. Если указано CASCADED, ограничение проверяется для этого представления и любых других представлений, основанных на этом. Если указано LOCAL, проверяется только это представление. Если ни CASCADED, ни LOCAL не указаны, то предполагается CASCADED.

```
CREATE VIEW vista AS SELECT 'Hello World';
```

Результат:

```
 ?column?
-------------
 Hello World
```

Более сложный пример: создание представления, состоящего из объединения двух таблиц:

```
CREATE VIEW comedies AS
    SELECT f.*,
           country_code_to_name(f.country_code) AS country,
           (SELECT avg(r.rating)
            FROM user_ratings r
            WHERE r.film_id = f.id) AS avg_rating
    FROM films f
    WHERE f.kind = 'Comedy';
```

Результат:

```
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           |          |
 kind   | text    |           |          |
 name   | text    |           |          |
 date   | date    |           |          |
 len    | interval|           |          |
```

Создание временного представления, которое будет автоматически удалено в конце текущего сеанса:

```
CREATE TEMP VIEW snacks AS
    SELECT s.*, sum(i.quantity) AS total_quantity
    FROM snacks s
    JOIN inventory i ON i.snack_id = s.id
    GROUP BY s.id, s.name, s.price;
```

Создание рекурсивного представления, которое возвращает числа от 1 до 100:

```
CREATE RECURSIVE VIEW nums_1_100 (n) AS
    VALUES (1)
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

Создание представления с WITH CHECK OPTION:

```
CREATE VIEW pg_customers AS
    SELECT id, name, email, country_code
    FROM customers
    WHERE country_code = 'PG'
    WITH LOCAL CHECK OPTION;
```

Создание представления с security_barrier:

```
CREATE VIEW secure_customers WITH (security_barrier = true) AS
    SELECT id, name, email, country_code
    FROM customers
    WHERE country_code = 'PG';
```

Создание представления с security_invoker:

```
CREATE VIEW secure_customers WITH (security_invoker = true) AS
    SELECT id, name, email, country_code
    FROM customers
    WHERE country_code = 'PG';
```

Удаление представления

Представления удаляются командой **DROP VIEW**. Например:

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

IF EXISTS - не выдавать ошибку, если представление не существует. В этом случае будет выдано уведомление.

name - имя представления, которое нужно удалить, может быть схема-квалифицированным.

CASCADE - автоматически удалять объекты, которые зависят от представления (такие как другие представления), и в свою очередь все объекты, которые зависят от этих объектов.

RESTRICT - не удалять представление, если от него зависят какие-либо объекты. Это поведение по умолчанию.

```
DROP VIEW kinds;
```

Материализованные представления

Материализованное представление - это объект базы данных, который содержит результаты запроса. Например, вы можете периодически получать агрегированную версию большой таблицы продаж для использования в панели управления. Материализованные представления похожи на кэши вызовов для запросов. Материализованное представление может быть проиндексировано для повышения производительности, и индексы будут храниться вместе с материализованным представлением.

Материализованное представление имеет много общего с таблицей. Оно также имеет столбцы и пространство данных, которое занимает на диске или в памяти. Однако материализованное представление не может быть изменено напрямую с помощью INSERT, UPDATE или DELETE. Единственный способ изменить данные в материализованном представлении - периодически обновлять его с помощью команды REFRESH MATERIALIZED VIEW.

Создание материализованного представления

Материализованные представления создаются с помощью команды **CREATE MATERIALIZED VIEW**. Например:

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] table_name
    [ (column_name [, ...] ) ]
    [ USING method ]
    [ WITH ( storage_parameter [= value] [, ...] ) ]
    [ TABLESPACE tablespace_name ]
    AS query
    [ WITH [ NO ] DATA ]
```

IF NOT EXISTS - не выдавать ошибку, если материализованное представление уже существует. В этом случае будет выдано уведомление.

table_name - имя материализованного представления, которое нужно создать, может быть схема-квалифицированным.

column_name - необязательный список имен, используемых в качестве имен столбцов материализованного представления. Если не указано, имена столбцов выводятся из запроса.

USING method - необязательное предложение USING указывает метод доступа к таблице, используемый для хранения содержимого материализованного представления. Значение по умолчанию - это метод доступа, определенный переменной default_table_access_method.

WITH (storage_parameter [= value] [, ...]) - это необязательное предложение WITH устанавливает необязательные параметры хранения для материализованного представления. Допустимые параметры хранения такие же, как для CREATE TABLE.

TABLESPACE tablespace_name - необязательное предложение TABLESPACE указывает табличное пространство, в котором будет создано материализованное представление. Если не указано, используется default_tablespace.

query - команда SELECT, которая будет выполнена для заполнения материализованного представления.

WITH [NO] DATA - это необязательное предложение WITH управляет тем, заполняется ли материализованное представление данными во время создания. Если указано WITH DATA, материализованное представление заполняется данными. Если указано WITH NO DATA, материализованное представление не заполняется данными, и оно помечается как неотсканируемое. Любая попытка сканировать материализованное представление приведет к ошибке.

```
CREATE MATERIALIZED VIEW kinds_mv AS SELECT * FROM kinds;
```

Результат:

```
SELECT 10
```

Создание материализованного представления, которое будет заполнено данными позже:

```
CREATE MATERIALIZED VIEW kinds_mv AS SELECT * FROM kinds WITH NO DATA;
```

Результат:

```
SELECT 0
```

Создание материализованного представления с индексом:

```
CREATE MATERIALIZED VIEW kinds_mv AS SELECT * FROM kinds;
CREATE INDEX ON kinds_mv (id);
```

Результат:

```
SELECT 10
CREATE INDEX
```

Удаление материализованного представления

Материализованные представления удаляются командой **DROP MATERIALIZED VIEW**. Например:

```
DROP MATERIALIZED VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

IF EXISTS - не выдавать ошибку, если материализованное представление не существует. В этом случае будет выдано уведомление.

name - имя материализованного представления, которое нужно удалить, может быть схема-квалифицированным.

CASCADE - автоматически удалять объекты, которые зависят от материализованного представления, и в свою очередь все объекты, которые зависят от этих объектов.

RESTRICT - не удалять материализованное представление, если от него зависят какие-либо объекты. Это поведение по умолчанию.

```
DROP MATERIALIZED VIEW kinds_mv;
```

Изменение данных представления

Представления могут быть обновляемыми или необновляемыми. Представление является обновляемым, если оно удовлетворяет следующим условиям:

- Оно содержит ровно одну запись в предложении FROM, которая должна быть таблицей или другим обновляемым представлением.
- Оно не содержит предложений DISTINCT, GROUP BY, HAVING, LIMIT или OFFSET на верхнем уровне.
- Оно не содержит операций UNION, INTERSECT или EXCEPT на верхнем уровне.
- Оно не содержит WITH на верхнем уровне.
- Оно не содержит агрегатных, оконных или функций, возвращающих множества, в своем списке выборки.

Обновляемое представление может содержать смесь обновляемых и необновляемых столбцов. Столбец является обновляемым, если он ссылается на обновляемый столбец базового отношения. Необновляемые столбцы допустимы, но они не могут быть изменены напрямую.

```
CREATE TABLE products (
    id serial PRIMARY KEY,
    name text NOT NULL,
    price numeric(10,2) NOT NULL,
    stock integer NOT NULL DEFAULT 0
);

INSERT INTO products (name, price, stock) VALUES
    ('Widget', 9.99, 100),
    ('Gadget', 19.99, 50),
    ('Doohickey', 4.99, 200);

CREATE VIEW expensive_products AS
    SELECT id, name, price, stock
    FROM products
    WHERE price > 10.00;

SELECT * FROM expensive_products;

UPDATE expensive_products SET price = 24.99 WHERE name = 'Gadget';

SELECT * FROM expensive_products;
```

Результат:

```
 id |  name  | price | stock
----+--------+-------+-------
  2 | Gadget | 19.99 |    50

UPDATE 1

 id |  name  | price | stock
----+--------+-------+-------
  2 | Gadget | 24.99 |    50
```

Создание обновляемого представления с WITH CHECK OPTION:

```
CREATE VIEW cheap_products AS
    SELECT id, name, price, stock
    FROM products
    WHERE price < 10.00
    WITH CHECK OPTION;

INSERT INTO cheap_products (name, price, stock) VALUES ('Thingamajig', 15.00, 10);
```

Результат:

```
ERROR:  new row violates check option for view "cheap_products"
DETAIL:  Failing row contains (4, Thingamajig, 15.00, 10).
```

Создание представления с security_barrier:

```
CREATE VIEW secure_products WITH (security_barrier = true) AS
    SELECT id, name, price, stock
    FROM products
    WHERE price > 10.00;
```

Создание представления с security_invoker:

```
CREATE VIEW secure_products WITH (security_invoker = true) AS
    SELECT id, name, price, stock
    FROM products
    WHERE price > 10.00;
```

Удаление материализованного представления

Материализованные представления удаляются командой **DROP MATERIALIZED VIEW**. Например:

```
DROP MATERIALIZED VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

IF EXISTS - не выдавать ошибку, если материализованное представление не существует. В этом случае будет выдано уведомление.

name - имя материализованного представления, которое нужно удалить, может быть схема-квалифицированным.

CASCADE - автоматически удалять объекты, которые зависят от материализованного представления, и в свою очередь все объекты, которые зависят от этих объектов.

RESTRICT - не удалять материализованное представление, если от него зависят какие-либо объекты. Это поведение по умолчанию.

```
DROP MATERIALIZED VIEW kinds_mv;
```

Сравнение представлений и материализованных представлений

Представления и материализованные представления имеют много общего, но также имеют и существенные различия.

Представления:

- Не занимают пространство данных, кроме как для хранения определения запроса.
- Всегда отражают текущее состояние данных, которые они запрашивают.
- Вычисляются каждый раз, когда к ним обращаются.
- Не могут быть проиндексированы.
- Могут быть обновляемыми или необновляемыми.

Материализованные представления:

- Занимают пространство данных для хранения результатов запроса.
- Могут быть устаревшими, если данные, которые они запрашивают, изменились.
- Вычисляются только при создании или обновлении.
- Могут быть проиндексированы.
- Всегда необновляемые.

Получение информации о представлениях и материализованных представлениях

Для получения информации о представлениях используется **\dv**, для материализованных представлений - **\dm**.

```
\dv
```

Результат:

```
                 List of relations
 Schema |       Name       | Type |  Owner
--------+------------------+------+----------
 public | cheap_products   | view | postgres
 public | expensive_products | view | postgres
 public | secure_products  | view | postgres
```

```
\dm
```

Результат:

```
                 List of relations
 Schema |       Name       | Type |  Owner
--------+------------------+------+----------
 public | kinds_mv         | materialized view | postgres
```

Для получения информации о столбцах представления используется **\d+ имя_представления**.

```
\d+ expensive_products
```

Результат:

```
                                  View "public.expensive_products"
 Column |  Type   | Collation | Nullable | Default | Storage | Description
--------+---------+-----------+----------+---------+---------+-------------
 id     | integer |           |          |         | plain   |
 name   | text    |           |          |         | extended|
 price  | numeric(10,2) |     |          |         | main    |
 stock  | integer |           |          |         | plain   |
View definition:
 SELECT id, name, price, stock
   FROM products
  WHERE price > 10.00;
```

Для получения информации о материализованном представлении:

```
\d+ kinds_mv
```

Результат:

```
                                Materialized view "public.kinds_mv"
 Column |  Type   | Collation | Nullable | Default | Storage | Description
--------+---------+-----------+----------+---------+---------+-------------
 id     | integer |           |          |         | plain   |
 kind   | text    |           |          |         | extended|
 name   | text    |           |          |         | extended|
 date   | date    |           |          |         | plain   |
 len    | interval|           |          |         | plain   |
View definition:
 SELECT * FROM kinds;
```

Системные представления

Системное представление	Описание
pg_available_extensions	Доступные расширения
pg_available_extension_versions	Версии доступных расширений
pg_backend_memory_contexts	Контексты памяти бэкенд-процесса
pg_config	Параметры, с которыми был собран PostgreSQL
pg_cursors	Доступные курсоры
pg_file_settings	Содержимое файлов конфигурации PostgreSQL
pg_group	Группы баз данных
pg_hba_file_rules	Правила файла pg_hba.conf
pg_ident_file_mappings	Сопоставления файла pg_ident.conf
pg_indexes	Индексы
pg_locks	Удерживаемые блокировки
pg_matviews	Материализованные представления
pg_policies	Политики безопасности на уровне строк
pg_prepared_statements	Подготовленные операторы
pg_prepared_xacts	Подготовленные транзакции
pg_publication	Публикации логической репликации
pg_publication_tables	Таблицы в публикациях логической репликации
pg_replication_origin	Источники репликации
pg_replication_origin_status	Статус источников репликации
pg_roles	Роли баз данных
pg_rules	Правила
pg_seclabels	Метки безопасности
pg_sequences	Последовательности
pg_settings	Параметры
pg_shadow	Пользователи баз данных
pg_shmem_allocations	Выделения разделяемой памяти
pg_stat_activity	Активность бэкенд-процессов
pg_stat_all_indexes	Статистика по всем индексам
pg_stat_all_tables	Статистика по всем таблицам
pg_stat_archiver	Статистика архивирования WAL
pg_stat_bgwriter	Статистика фонового писателя
pg_stat_database	Статистика баз данных
pg_stat_database_conflicts	Статистика конфликтов баз данных
pg_stat_gssapi	Статистика GSSAPI
pg_stat_progress_analyze	Прогресс команды ANALYZE
pg_stat_progress_basebackup	Прогресс команды pg_basebackup
pg_stat_progress_cluster	Прогресс команд CLUSTER и VACUUM FULL
pg_stat_progress_copy	Прогресс команды COPY
pg_stat_progress_create_index	Прогресс команд CREATE INDEX и REINDEX
pg_stat_progress_vacuum	Прогресс команды VACUUM
pg_stat_replication	Статистика репликации
pg_stat_replication_slots	Статистика слотов репликации
pg_stat_slru	Статистика SLRU
pg_stat_ssl	Статистика SSL
pg_stat_subscription	Статистика подписок
pg_stat_subscription_stats	Статистика подписок
pg_stat_sys_indexes	Статистика по системным индексам
pg_stat_sys_tables	Статистика по системным таблицам
pg_stat_user_functions	Статистика по пользовательским функциям
pg_stat_user_indexes	Статистика по пользовательским индексам
pg_stat_user_tables	Статистика по пользовательским таблицам
pg_stat_wal	Статистика WAL
pg_stat_wal_receiver	Статистика приемника WAL
pg_stat_xact_all_tables	Транзакционная статистика по всем таблицам
pg_stat_xact_sys_tables	Транзакционная статистика по системным таблицам
pg_stat_xact_user_functions	Транзакционная статистика по пользовательским функциям
pg_stat_xact_user_tables	Транзакционная статистика по пользовательским таблицам
pg_statio_all_indexes	Статистика ввода-вывода по всем индексам
pg_statio_all_sequences	Статистика ввода-вывода по всем последовательностям
pg_statio_all_tables	Статистика ввода-вывода по всем таблицам
pg_statio_sys_indexes	Статистика ввода-вывода по системным индексам
pg_statio_sys_sequences	Статистика ввода-вывода по системным последовательностям
pg_statio_sys_tables	Статистика ввода-вывода по системным таблицам
pg_statio_user_indexes	Статистика ввода-вывода по пользовательским индексам
pg_statio_user_sequences	Статистика ввода-вывода по пользовательским последовательностям
pg_statio_user_tables	Статистика ввода-вывода по пользовательским таблицам
pg_stats	Статистика планировщика
pg_stats_ext	Расширенная статистика планировщика
pg_stats_ext_exprs	Расширенная статистика планировщика по выражениям
pg_tables	Таблицы
pg_timezone_abbrevs	Сокращения часовых поясов
pg_timezone_names	Названия часовых поясов
pg_user	Пользователи баз данных
pg_user_mappings	Сопоставления пользователей
pg_views	Представления

Последовательности

Алексей Лесовский

Содержание

Создание последовательности
Изменение последовательности
Удаление последовательности
Работа с последовательностью
- nextval
- setval
- currval
- lastval

Получение информации о последовательности

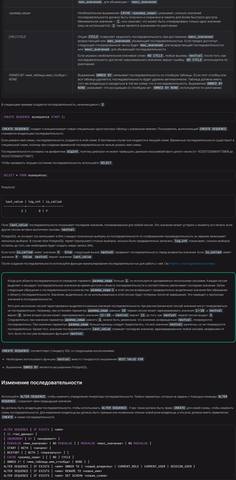
Последовательность — это объект в базе данных, который генерирует целые числа в порядке возрастания или убывания. Последовательности часто используются для создания уникальных числовых значений, таких как первичные ключи в таблицах. Они гарантируют, что каждое сгенерированное значение будет уникальным, даже при одновременном доступе нескольких пользователей.

PostgreSQL поддерживает работу с последовательностями.

Создание последовательности

Чтобы создать последовательность, выполните команду `CREATE SEQUENCE`, которая создаст новый генератор последовательности:

```
CREATE [ TEMPORARY | TEMP ] [ IF NOT EXISTS ] имя_последовательности
    [ AS тип_данных ]
    [ INCREMENT [ BY ] приращение ]
    [ MINVALUE минимальное_значение | NO MINVALUE ] [ MAXVALUE максимальное_значение | NO MAXVALUE ]
    [ START [ WITH ] начальное_значение ]
    [ CACHE кэш ] [ [ NO ] CYCLE ]
    [ OWNED BY { имя_таблицы.имя_столбца | NONE } ]
```



Параметры команды CREATE SEQUENCE

`TEMPORARY | TEMP`	Если указано, объект последовательности создается только для текущего сеанса. Временные последовательности удаляются в конце сеанса. Существующая постоянная последовательность с тем же именем не будет видна в текущем сеансе.
`IF NOT EXISTS`	Не выдавать ошибку, если последовательность с таким именем уже существует. В этом случае будет выдано лишь уведомление. Обратите внимание, что нет никакой гарантии, что существующая последовательность имеет какое-либо сходство с той, которую могла бы создать эта команда.
`имя`	Имя создаваемого объекта. Имя может включать имя схемы (`имя_схемы.имя_последовательности`).
`тип_данных`	Тип данных последовательности. Этот тип данных может быть любым целочисленным типом (`SMALLINT`, `INTEGER` или `BIGINT`). Тип данных определяет минимальное и максимальное значения последовательности по умолчанию. Тип данных не ограничивает значения, которые могут быть сохранены в столбцах, которые будут использовать последовательность. Например, последовательность может быть назначена столбцу `INTEGER`, но последовательность может генерировать числа, которые превышают диапазон `INTEGER` (больше 2147483647). По умолчанию используется `BIGINT`.
`приращение`	Значение, которое будет добавлено к текущему значению последовательности для создания нового значения. Положительное значение создаст возрастающую последовательность, отрицательное — убывающую. Значение по умолчанию равно 1.
`минимальное_значение`	Определяет минимальное значение, которое последовательность будет генерировать. Если указано `NO MINVALUE` (или не указано), будут использованы значения по умолчанию: 1 для возрастающей последовательности и -2^63-1 для убывающей.
`максимальное_значение`	Определяет максимальное значение последовательности. Если указано `NO MAXVALUE` (или не указано), будут использованы значения по умолчанию: 2^63-1 для возрастающей последовательности и -1 для убывающей.
`начальное_значение`	Необязательное предложение, которое позволяет начать последовательность с любого значения. По умолчанию `START WITH минимальное_значение` для возрастающей последовательности и `START WITH максимальное_значение` для убывающей.
`кэш`	Указывает, сколько значений последовательности будет предварительно выделено и сохранено в памяти для более быстрого доступа. Минимальное значение — 1 (только одно значение может быть сгенерировано за раз, т. е. `NO CACHE`), и это также значение по умолчанию.
`CYCLE` | `NO CYCLE`	Указывает, должна ли последовательность продолжать генерировать значения после достижения своего максимального или минимального значения. Если достигнут предел возрастающей или убывающей последовательности, будет сгенерировано следующее значение, соответственно, максимальное или минимальное значение. Если `NO CYCLE` (значение по умолчанию), любой вызов `nextval` после достижения предела будет возвращать ошибку.
`OWNED BY имя_таблицы.имя_столбца` | `OWNED BY NONE`	Связывает последовательность с определенным столбцом таблицы. Если используется `OWNED BY`, последовательность будет автоматически удалена при удалении столбца, на который она ссылается. Это обеспечивает правильную очистку последовательностей, когда они больше не нужны. По умолчанию `OWNED BY NONE`, что означает, что последовательность не связана с каким-либо столбцом.

В следующем примере создается последовательность `seq_example`:

```
CREATE SEQUENCE seq_example START 1;
```

Команда `CREATE SEQUENCE` создает новый генератор последовательности с именем `seq_example`. Параметр `START 1` указывает, что последовательность должна начинаться с 1.

После создания этого генератора последовательности вы можете использовать функции для работы с ним. Функция `nextval` используется для получения следующего значения последовательности. Например:

```
SELECT nextval('seq_example');
```

Результат:

```
 nextval
---------
       1
(1 row)
```

Поле `last_value` последовательности показывает последнее значение, сгенерированное для любой сессии. Это значение может устать изменить его вызов или откат транзакции. Например:

```
SELECT last_value FROM seq_example;
```

Результат:

```
 last_value
------------
          1
(1 row)
```

Если вы вызовете `nextval` для последовательности в новом сеансе, вы получите значение после того, которое было возвращено в предыдущем сеансе:

```
SELECT nextval('seq_example');
```

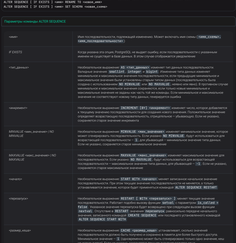
Результат:

```
 nextval
---------
       2
(1 row)
```

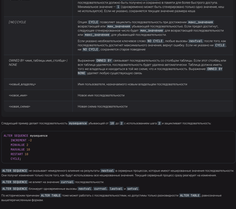
Если вы хотите, чтобы ваша последовательность была связана с таблицей, вы можете использовать опцию `OWNED BY`. Например:

```
CREATE SEQUENCE seq_example2 OWNED BY mytable.myid;
```

Эта команда создаст новую последовательность с именем `seq_example2`, которая будет связана со столбцом `myid` таблицы `mytable`. Это означает, что последовательность будет автоматически удалена, если столбец `myid` или вся таблица `mytable` будут удалены.



Когда для объекта последовательности определен параметр `OWNED BY`, он автоматически удаляется из своего текущего владельца, если только новый владелец не является таблицей, которая уже владеет последовательностью. Кроме того, он автоматически становится зависимым от схемы и владельца новой таблицы, если только `OWNED BY` не указывает на таблицу в той же схеме. Эти поведения позволяют избежать случайного удаления последовательности, которая все еще используется, и обеспечивают правильную миграцию последовательности при перемещении таблицы в другую схему. Однако они могут быть неожиданными, если вы ожидали, что `OWNED BY` будет работать иначе.



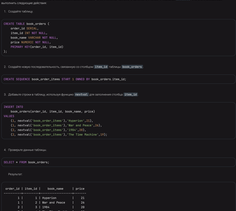
Когда для объекта последовательности определен параметр `OWNED BY`, он автоматически удаляется из своего текущего владельца, если только новый владелец не является таблицей, которая уже владеет последовательностью. Кроме того, он автоматически становится зависимым от схемы и владельца новой таблицы, если только `OWNED BY` не указывает на таблицу в той же схеме. Эти поведения позволяют избежать случайного удаления последовательности, которая все еще используется, и обеспечивают правильную миграцию последовательности при перемещении таблицы в другую схему. Однако они могут быть неожиданными, если вы ожидали, что `OWNED BY` будет работать иначе.

Вы можете узнать больше о последовательностях в официальной документации: https://postgrespro.ru/docs/.

Изменение последовательности

Используйте `ALTER SEQUENCE`, чтобы изменить параметры генератора последовательности. Любые параметры, которые не заданы с помощью команды `ALTER SEQUENCE`, сохраняют свои предыдущие настройки.

```
ALTER SEQUENCE [ IF EXISTS ] имя
    [ AS тип_данных ]
    [ INCREMENT [ BY ] приращение ]
    [ MINVALUE минимальное_значение | NO MINVALUE ] [ MAXVALUE максимальное_значение | NO MAXVALUE ]
    [ START [ WITH ] начальное_значение ]
    [ RESTART [ [ WITH ] перезапуск_значение ] ]
    [ CACHE кэш ] [ [ NO ] CYCLE ]
    [ OWNED BY { имя_таблицы.имя_столбца | NONE } ]
ALTER SEQUENCE [ IF EXISTS ] имя RENAME TO новое_имя
ALTER SEQUENCE [ IF EXISTS ] имя SET SCHEMA новая_схема
```



Параметры команды ALTER SEQUENCE

`имя`	Имя последовательности, подлежащей изменению. Может включать имя схемы (`имя_схемы.имя_последовательности`).
`IF EXISTS`	Не выдавать ошибку, если последовательность с таким именем не существует. В этом случае будет выдано лишь уведомление.
`тип_данных`	Тип данных должен быть целочисленным типом (`SMALLINT`, `INTEGER` или `BIGINT`). Изменение типа данных автоматически изменяет минимальное и максимальное значения последовательности (если предыдущие значения были значениями по умолчанию для старого типа данных), чтобы соответствовать значениям по умолчанию для нового типа данных.
`приращение`	Необязательное предложение `INCREMENT BY` изменяет значение, которое добавляется к текущему значению последовательности для создания нового значения. Положительное значение создаст возрастающую последовательность, отрицательное — убывающую. Если не указано, приращение остается прежним.
`минимальное_значение`	Необязательное предложение `MINVALUE` изменяет минимальное значение последовательности. Если указано `NO MINVALUE`, будут использованы значения по умолчанию: 1 для возрастающей последовательности и -2^63-1 для убывающей. Если не указано, минимальное значение остается прежним.
`максимальное_значение`	Необязательное предложение `MAXVALUE` изменяет максимальное значение последовательности. Если указано `NO MAXVALUE`, будут использованы значения по умолчанию: 2^63-1 для возрастающей последовательности и -1 для убывающей. Если не указано, максимальное значение остается прежним.
`начало`	Необязательное предложение `START WITH` изменяет начальное значение последовательности. Это не влияет на текущее значение последовательности; оно влияет только на значение, которое будет использовать последующая команда `ALTER SEQUENCE RESTART`.
`перезапуск`	Необязательное предложение `RESTART` изменяет значение последовательности. Значение может быть указано, или можно использовать значение по умолчанию, которое является начальным значением, которое было указано в предыдущей команде `ALTER SEQUENCE START` или начальным значением, которое было указано в последующей команде `CREATE SEQUENCE`. Команда `ALTER SEQUENCE RESTART` не может использовать значение по умолчанию.
`кэш`	Указывает, сколько значений последовательности будет предварительно выделено и сохранено в памяти для более быстрого доступа. Минимальное значение — 1 (только одно значение может быть сгенерировано за раз, т. е. `NO CACHE`).
`CYCLE` | `NO CYCLE`	Указывает, должна ли последовательность продолжать генерировать значения после достижения своего максимального или минимального значения. Если достигнут предел возрастающей или убывающей последовательности, будет сгенерировано следующее значение, соответственно, максимальное или минимальное значение. Если `NO CYCLE`, любой вызов `nextval` после достижения предела будет возвращать ошибку.
`OWNED BY имя_таблицы.имя_столбца` | `OWNED BY NONE`	Связывает последовательность с определенным столбцом таблицы. Если используется `OWNED BY`, последовательность будет автоматически удалена при удалении столбца, на который она ссылается. Это обеспечивает правильную очистку последовательностей, когда они больше не нужны. По умолчанию `OWNED BY NONE`, что означает, что последовательность не связана с каким-либо столбцом.
`новый_владелец`	Имя пользователя, становящегося новым владельцем последовательности.
`новое_имя`	Новое имя последовательности.
`новая_схема`	Новая схема последовательности.

Необязательное предложение `RESTART` может быть использовано для сброса последовательности, как и при создании последовательности.

```
ALTER SEQUENCE последовательность
    INCREMENT 2
    MINVALUE 0
    MAXVALUE 10
    START 0
    RESTART 0
    CACHE 1
    NO CYCLE;
```

`ALTER SEQUENCE` не оказывает немедленного влияния на последовательность. Изменения вступают в силу для каждого последующего вызова `nextval`. Текущее значение последовательности не изменяется.

`ALTER SEQUENCE` не оказывает немедленного влияния на последовательность. Изменения вступают в силу для каждого последующего вызова `nextval`. Текущее значение последовательности не изменяется.

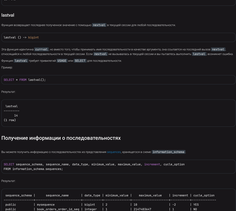
`ALTER SEQUENCE` не оказывает немедленного влияния на последовательность. Изменения вступают в силу для каждого последующего вызова `nextval`. Текущее значение последовательности не изменяется.

Полную информацию смотрите в официальной документации: https://postgrespro.ru/docs/postgrespro/current/sql-altersequence.

Удаление последовательности

Команда `DROP SEQUENCE` удаляет последовательность. Это операция может осуществлять только владелец последовательности.

```
DROP SEQUENCE [ IF EXISTS ] имя [, ...] [ CASCADE | RESTRICT ]
```



Параметры команды DROP SEQUENCE

`IF EXISTS`	Не выдавать ошибку, если последовательность не существует. В этом случае будет выдано лишь уведомление.
`имя`	Имя последовательности (возможно, дополненное именем схемы).
`CASCADE`	Удалить объекты, зависящие от данной последовательности, и все зависящие от этих объектов (см. раздел 5.14 документации).
`RESTRICT`	Не удалять последовательность, если от нее зависят какие-либо объекты. Это поведение по умолчанию.

Пример:

```
DROP SEQUENCE последовательность;
```

Работа с последовательностью

Функции, описанные ниже, предоставляют простые и безопасные способы работы с последовательностями:

nextval

Функция `nextval()` продвигает объект последовательности к следующему значению и возвращает это значение.

```
nextval ( 'генератор' ) → bigint
```

Если объект последовательности был создан с параметрами по умолчанию, `nextval` последовательно возвращает следующие значения, начиная с 1. Другие поведения можно получить, изменив параметры последовательности, как показано в команде `CREATE SEQUENCE`.

Функция `nextval()` продвигает объект последовательности к следующему значению и возвращает это значение. Это делается атомарно: даже если несколько сеансов выполняют `nextval()` одновременно, каждый из них безопасно получит отдельное значение последовательности.

Функция `nextval()` продвигает объект последовательности к следующему значению и возвращает это значение. Это делается атомарно: даже если несколько сеансов выполняют `nextval()` одновременно, каждый из них безопасно получит отдельное значение последовательности.

Функция `nextval()` продвигает объект последовательности к следующему значению и возвращает это значение. Это делается атомарно: даже если несколько сеансов выполняют `nextval()` одновременно, каждый из них безопасно получит отдельное значение последовательности.

Операции с последовательностями никогда не откатываются назад, чтобы обеспечить гарантии одновременности, описанные выше. Если транзакция, использующая некоторые значения последовательности, прерывается, эти значения потеряны для других транзакций. Аналогично, использованное значение последовательности никогда не может быть возвращено. Это может привести к «дырам» в последовательностях, поскольку генерируемые значения могут никогда не использоваться в таблице. Счетчики последовательности восстановления не откатываются назад, даже если изменения, которые они отслеживают, откатываются назад.

Операции с последовательностями никогда не откатываются назад, чтобы обеспечить гарантии одновременности, описанные выше. Если транзакция, использующая некоторые значения последовательности, прерывается, эти значения потеряны для других транзакций. Аналогично, использованное значение последовательности никогда не может быть возвращено. Это может привести к «дырам» в последовательностях, поскольку генерируемые значения могут никогда не использоваться в таблице. Счетчики последовательности восстановления не откатываются назад, даже если изменения, которые они отслеживают, откатываются назад.

Операции с последовательностями никогда не откатываются назад, чтобы обеспечить гарантии одновременности, описанные выше. Если транзакция, использующая некоторые значения последовательности, прерывается, эти значения потеряны для других транзакций. Аналогично, использованное значение последовательности никогда не может быть возвращено. Это может привести к «дырам» в последовательностях, поскольку генерируемые значения могут никогда не использоваться в таблице. Счетчики последовательности восстановления не откатываются назад, даже если изменения, которые они отслеживают, откатываются назад.

Создайте таблицу:

```
CREATE TABLE book_orders (
    order_id SERIAL,
    item_id TEXT NOT NULL,
    price NUMERIC(10,2) NOT NULL,
    PRIMARY KEY(order_id, item_id)
);
```

Создайте новую последовательность и свяжите ее с таблицей `book_orders`:

```
CREATE SEQUENCE book_order_item START 1 OWNED BY book_orders.item_id;
```

Добавьте строки в таблицу, используя функцию `nextval` для заполнения столбца `item_id`:

```
INSERT INTO
    book_orders(order_id, item_id, book_name, price)
VALUES
    (nextval('book_order_item'), 'hyperion',2),
    (nextval('book_order_item'), 'War and Peace',1),
    (nextval('book_order_item'), 'FOR',10),
    (nextval('book_order_item'), 'The Time Machine',25);
```

Проверьте данные таблицы:

```
SELECT * FROM book_orders;
```

Результат:

```
 order_id | item_id |    book_name    | price
----------+---------+-----------------+-------
        1 | hyperion                  |  2.00
        2 | War and Peace             |  1.00
        3 | FOR                       | 10.00
        4 | The Time Machine          | 25
(4 rows)
```

Если вы удалите таблицу `book_orders`, последовательность `book_order_item` также будет удалена.

setval

Функция `setval()` устанавливает текущее значение объекта последовательности и, если был `is_called`, `setval` также выставляет у него флаг `is_called`.

```
setval ( 'генератор', результат [, is_called ] ) → bigint
```

- `генератор` — имя генератора `text`;
- `результат` — значение `bigint`;
- `is_called` — значение `boolean`, по умолчанию `true`.

Если `is_called` имеет значение `true`, следующий вызов `nextval` будет возвращать значение, следующее за `setval`. В противном случае следующий вызов `nextval` возвращает значение `setval`. Например, следующая последовательность вызовов возвращает 32 и 33:

```
SELECT setval('foo', 32);
SELECT nextval('foo');
```

Результат:

```
 setval
--------
     32
(1 row)

 nextval
---------
      33
(1 row)
```

Последовательность вызовов возвращает 5 и 6:

```
SELECT setval('foo', 5, false);
SELECT nextval('foo');
```

Результат:

```
 setval
--------
      5
(1 row)

 nextval
---------
       6
(1 row)
```

Функция `setval` не оказывает немедленного влияния на последовательность. Изменения вступают в силу для каждого последующего вызова `nextval`. Текущее значение последовательности не изменяется.

Функция `setval` не оказывает немедленного влияния на последовательность. Изменения вступают в силу для каждого последующего вызова `nextval`. Текущее значение последовательности не изменяется.

Функция `setval` не оказывает немедленного влияния на последовательность. Изменения вступают в силу для каждого последующего вызова `nextval`. Текущее значение последовательности не изменяется.

currval

Функция `currval()` возвращает последнее значение, полученное с помощью `nextval` для указанной последовательности в текущем сеансе.

```
currval ( 'генератор' ) → bigint
```

Если `currval()` вызывается для последовательности в текущем сеансе, но `nextval` для данной последовательности в текущем сеансе не вызывался, `currval` выдает ошибку. Поскольку `currval` относится к значению, возвращенному последним вызовом `nextval` для данного сеанса, неважно, что делают другие сеансы между выполнением `nextval` и `currval` в вашем сеансе. Если другой сеанс вызывает `nextval` для той же последовательности, `currval` возвращает значение, возвращенное вашим собственным вызовом `nextval`, если таковой имеется.

Функция `currval` возвращает последнее значение, полученное с помощью `nextval` для указанной последовательности в текущем сеансе.

Функция `currval` возвращает последнее значение, полученное с помощью `nextval` для указанной последовательности в текущем сеансе.

Функция `currval` возвращает последнее значение, полученное с помощью `nextval` для указанной последовательности в текущем сеансе.

```
SELECT currval('seq_example');
```

Результат:

```
 currval
---------
       2
(1 row)
```

lastval

Функция возвращает последнее полученное значение с помощью `nextval` в текущем сеансе (это работает независимо от того, какая последовательность использовалась):

```
lastval ( ) → bigint
```

Это функция идентична `currval`, за исключением того, что вместо указания имени последовательности она возвращает последнее значение, полученное с помощью `nextval` для любой последовательности в текущем сеансе. Если `nextval` еще не вызывался в текущем сеансе, вызов `lastval` приведет к ошибке.

Это функция идентична `currval`, за исключением того, что вместо указания имени последовательности она возвращает последнее значение, полученное с помощью `nextval` для любой последовательности в текущем сеансе. Если `nextval` еще не вызывался в текущем сеансе, вызов `lastval` приведет к ошибке.

Это функция идентична `currval`, за исключением того, что вместо указания имени последовательности она возвращает последнее значение, полученное с помощью `nextval` для любой последовательности в текущем сеансе. Если `nextval` еще не вызывался в текущем сеансе, вызов `lastval` приведет к ошибке.

```
SELECT lastval();
```

Результат:

```
 lastval
---------
       2
(1 row)
```

Получение информации о последовательностях

Вы можете получить информацию о последовательностях из представления `sequences`, доступного в схеме `information_schema`:

```
SELECT sequence_schema, sequence_name, data_type, minimum_value, maximum_value, increment, cycle_option
FROM information_schema.sequences;
```

Результат:

```
 sequence_schema |  sequence_name  | data_type | minimum_value | maximum_value | increment | cycle_option
-----------------+-----------------+-----------+---------------+---------------+-----------+--------------
 public          | book_orders     | bigint    | 1             | 10            | 1         | NO
 public          | book_order_id_seq | bigint  | 1             | 2147483647    | 1         | NO
 public          | book_order_item | bigint    | 1             | 9223372036854775807 | 1       | NO
(3 rows)
```

Результат включает автоматически созданную последовательность `book_orders_order_id_seq`, для столбца `order_id` из `book_orders`.

Вы можете получить информацию о конкретной последовательности, указав ее имя в условии `WHERE`:

```
SELECT * FROM information_schema.sequences
WHERE sequence_name = 'имя_последовательности';
```

Пример:

```
SELECT * FROM information_schema.sequences
WHERE sequence_name = 'book_order_item';
```

Результат:

```
-[ RECORD 1 ]--------+-------------------
sequence_catalog     | postgres
sequence_schema      | public
sequence_name        | book_order_item
data_type            | bigint
numeric_precision    | 64
numeric_precision_radix | 2
numeric_scale        | 0
start_value          | 1
minimum_value        | 1
maximum_value        | 9223372036854775807
increment            | 1
cycle_option         | NO
```

Кроме того, вы также можете получить данные из системного каталога `pg_sequences` и `pg_class`, используя следующий запрос:

```
SELECT relname,relowner,relkind,relpersistence,seqstart,seqmin,seqmax,seqincrement,seqcycle,seqcache
FROM pg_class JOIN pg_sequence ON pg_class.oid = pg_sequence.seqrelid;
```

Результат:

```
-[ RECORD 1 ]+-------
relname      | book_order_item
relowner     | 10
relkind      | S
relpersistence | p
seqstart     | 1
seqmin       | 1
seqmax       | 9223372036854775807
seqincrement | 1
seqcycle     | f
seqcache     | 1
```

Или, вы можете использовать команду `psql` `\ds`:

```
\ds
```

Результат:

```
 Schema |      Name       |   Type   |  Owner
--------+-----------------+----------+----------
 public | book_order_item | sequence | postgres
 public | book_order_id_seq | sequence | postgres
 public | book_orders_order_id_seq | sequence | postgres
(3 rows)
```

Содержание

Назад к содержанию

- Обзор
- Типы индексов
  - B-дерево
    - Хеш
    - GiST
    - SP-GiST
    - GIN
    - BRIN
- Управление индексами
- Перестроение индексов
- Составные индексы
- Уникальные индексы
- Индексы по выражениям
- Частичные индексы
- Контроль использования индекса

Обзор

PostgreSQL поддерживает несколько типов индексов B-дерево, GiST, SP-GiST, GIN, BRIN и расширение Bloom. Каждый тип индексов использует свой алгоритм, который подходит для работы с определенными типами запросов. По умолчанию команда `CREATE INDEX` создает индекс типа B-дерево, которые эффективен в большинстве случаев.

B-дерево

Индексы B-дерево (balanced tree — сбалансированное дерево) по структуре напоминают сбалансированное дерево поиска, но, в отличие от него, могут иметь более двух потомков у узла. Такая структура сохраняет сбалансированность дерева при произвольном порядке добавления данных и позволяет эффективно искать, добавлять и удалять данные. Время выполнения этих операций растет логарифмически от общего количества элементов, поэтому даже при большом количестве элементов в индексе обращение к ним происходит быстро.

Индексы B-дерево могут использоваться в запросах, которые содержат условия с операторами сравнения `<`, `<=`, `=`, `>=`, `>`, а также `BETWEEN`, `IN`, `IS NULL` и `IS NOT NULL`. Кроме того, они могут использоваться для сортировки данных.

Индексы B-дерево могут быть составными, то есть включать несколько столбцов. Например, если таблица содержит столбцы `col1` и `col2`, то можно создать индекс, который будет использоваться для запросов, содержащих условия на оба столбца, только на `col1` или на `col1` и `col2`, но не только на `col2`.

```
CREATE INDEX ON book (col1, col2);
```

Хеш

Хеш-индекс хранит 32-битный хеш-код, вычисленный на основе значений столбца. Следовательно, такие индексы могут обрабатывать только простые сравнения с использованием операторов `=` и `IN`. Составные хеш-индексы не поддерживаются.

```
CREATE INDEX ON book USING HASH (id);
```

GiST

Индексы GiST (Generalized Search Tree) предназначены для работы со сложными типами данных, которые не поддерживают сравнение. Такого рода данные могут быть, например, многомерными (географические координаты, геометрические фигуры) или текстовыми (полнотекстовый поиск). Индексы GiST — это инфраструктура, которая позволяет реализовать произвольную стратегию индексирования для конкретного типа данных. Например, встроенное расширение PostGIS использует индексы GiST для работы с географическими данными.

```
CREATE INDEX ON point USING GIST (x, y);
```

SP-GiST

Индексы SP-GiST (Space-Partitioned GiST) поддерживают, как и GiST, произвольную инфраструктуру индексирования. SP-GiST позволяет реализовать различные структуры данных, такие как префиксные деревья, деревья квадрантов и k-мерные деревья. Эти структуры данных изначально предназначались для работы в оперативной памяти, и SP-GiST позволяет эффективно реализовать их в виде индексов на диске.

Встроенные операторы индекса SP-GiST для точек поддерживают операторы `<<`, `>>`, `~=`, `<@`, `<^` и `>^`.

```
CREATE INDEX ON point USING SPGIST (x, y);
```

GIN

Индексы GIN (Generalized Inverted Index — «инвертированные индексы») подходят для ситуаций, когда значения индекса являются составными, и запрос ищет элементы, которые могут находиться внутри этих значений. Такие индексы могут быть полезны, например, для массивов, документов в формате JSONB и полнотекстового поиска.

Индекс GIN может ускорить поиск по составным значениям, но замедлить операции изменения данных. Поэтому GIN-индексы лучше подходят для данных, которые редко изменяются.

Встроенные операторы GIN для массивов поддерживают операторы `&&`, `@>`, `<@` и `=`.

```
CREATE INDEX ON book USING GIN (tsvector);
```

BRIN

Индексы BRIN (Block Range Index) хранят сводную информацию о значениях, последовательно расположенных в больших таблицах. Они работают эффективно, если данные в таблице упорядочены определенным образом, например, по времени. Индекс BRIN может быть очень маленьким по размеру, но при этом ускорить выполнение запросов с использованием диапазонов. BRIN поддерживает индексирование встроенных типов данных, которые поддерживают операторы сравнения, а также типов данных, которые поддерживают операторы `<<`, `&<`, `&>` и `>>`.

Пример создания индекса BRIN для столбца `created_at` типа `timestamp`:

```
CREATE INDEX ON book USING BRIN (created_at);
```

Управление индексами

Во многих СУБД, например MySQL, индексы являются частью схемы данных и должны быть заданы при проектировании структуры таблиц. В PostgreSQL индексы можно создавать и удалять в любой момент, независимо от структуры таблицы. Для создания нового индекса используется команда `CREATE INDEX`, а для удаления — `DROP INDEX`.

```
CREATE INDEX book_idx ON books (title);

DROP INDEX book_idx;
```

Чтобы изменить тип индекса, удалите его и создайте заново. Изменить можно также имя индекса, для этого используется команда `ALTER INDEX`.

```
ALTER INDEX book_idx RENAME TO books_idx;
```

В приведенном коде команда `ALTER INDEX` переименовывает индекс `book_idx` в `books_idx`.

Команда `ALTER INDEX` может также использоваться для изменения параметров хранения индекса. Например, можно изменить коэффициент заполнения индекса, который определяет, насколько плотно будут заполнены страницы индекса при его создании. Подробнее об этом можно прочитать в документации.

Также команда `ALTER INDEX` позволяет изменить статистику, которая используется планировщиком запросов. Команда `ALTER INDEX` используется для изменения статистики конкретного индекса. Подробнее об этом можно прочитать в документации.

```
ALTER INDEX books_idx SET TABLESPACE ts_tablespace;
```

Чтобы удалить индекс, используйте `DROP INDEX`.

Следующая команда удаляет индекс `SupplierName_Idx`:

```
DROP INDEX suppliers_idx;
```

Перестроение индексов

В PostgreSQL есть команда `REINDEX`, которая позволяет перестроить индекс. `REINDEX` перестраивает индекс, используя данные, хранящиеся в таблице индекса, и заменяет старую копию индекса. `REINDEX` может быть полезен в нескольких ситуациях:

- Индекс повреждается, и вы не можете восстановить данные. Использование `REINDEX` может помочь восстановить индекс и избежать потери данных.
- Вы изменили параметр хранения, такой как `FILLFACTOR`, и хотите, чтобы изменения вступили в силу. `REINDEX` перестраивает индекс, используя текущие параметры.
- Индекс стал слишком большим и содержит много пустого пространства. `REINDEX` может помочь уменьшить размер индекса и повысить производительность.

Команда `REINDEX` может быть выполнена для индекса, таблицы или базы данных. Вы можете использовать различные параметры, чтобы указать, какие индексы следует перестроить.

```
REINDEX INDEX my_index;

REINDEX TABLE my_table;
```

Составные индексы

Индекс может быть создан сразу по нескольким столбцам таблицы. Например, имеется следующая таблица:

```
CREATE TABLE authors (
    id bigint PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL
);
```

В таблице есть вымышленные следующие записи:

```
SELECT * FROM authors WHERE first_name = 'Valiet' AND last_name = 'Valiev';
```

Вы можете определить составной индекс для столбцов `first_name` и `last_name`:

```
CREATE INDEX idx_authors_name ON authors (last_name, first_name);
```

Составной индекс будет использоваться для запросов, которые включают в себя столбцы, указанные в индексе. В приведенном выше примере индекс `idx_authors_name` будет использоваться для запросов, которые включают в себя столбцы `last_name` и `first_name`. Однако индекс не будет использоваться для запросов, которые включают в себя только столбец `first_name`.

Составные индексы могут быть полезны для запросов, которые включают в себя несколько столбцов. Однако составные индексы могут быть менее эффективными, чем индексы для отдельных столбцов, если запрос включает в себя только один столбец.

Составные индексы могут быть полезны для запросов, которые включают в себя несколько столбцов. Однако составные индексы могут быть менее эффективными, чем индексы для отдельных столбцов, если запрос включает в себя только один столбец.

Уникальные индексы

Индексы также могут гарантировать уникальность значений в столбце. Для этого используется ключевое слово `UNIQUE`. Чтобы создать уникальный индекс, используйте следующую команду:

```
CREATE UNIQUE INDEX name ON tbl(expr [,expr...]);
```

Например:

```
CREATE UNIQUE INDEX book_unique ON book (column1, column2);
```

В настоящее время уникальность могут иметь только индексы типа B-дерево.

Если индекс создается как уникальный, в таблицу невозможно добавить строки с одинаковыми значениями ключей. Значения `NULL` не считаются одинаковыми. Составной уникальный индекс отклоняет только случаи, когда все столбцы индекса имеют одинаковые значения в двух строках. Уникальность поддерживается с помощью B-дерева. Если таблица уже содержит данные, которые не удовлетворяют ограничению уникальности, создание индекса завершится с ошибкой. Для получения дополнительной информации см. Создание индекса в документации PostgreSQL.

Индексы по выражениям

Индекс может быть создан для функции более или менее сложных столбцов таблицы. Например, имеется следующая таблица:

```
SELECT * FROM table WHERE lower(column1) = 'value1';
```

Этот запрос может использовать индекс, определенный для результата `lower(column1)`:

```
CREATE INDEX idx_lower_column1 ON table (lower(column1));
```

Этот индекс создаст индекс на основе результата `lower(column1)`. Когда запрос использует `lower(column1)`, он будет использовать индекс, чтобы ускорить поиск. Индекс будет полезен только для запросов, которые используют `lower(column1)`. Если запрос не использует `lower(column1)`, индекс не будет использоваться.

```
SELECT * FROM authors WHERE (first_name || ' ' || last_name) = 'Ivan Bunin';
```

Для этого запроса можно создать индекс:

```
CREATE INDEX idx_name ON authors ((first_name || ' ' || last_name));
```

Синтаксис команды `CREATE INDEX` требует записывать выражение индекса особым образом: сначала идет ключевое слово `ON`, затем имя таблицы, затем выражение индекса в скобках.

Частичные индексы

Частичный индекс — это индекс, построенный по подмножеству таблицы. Определяется условием, указанным в команде. Индекс включает только строки, которые удовлетворяют этому условию. Частичные индексы могут быть полезны, если вы часто выполняете запросы, которые включают в себя только небольшое подмножество данных в таблице.

Одной из основных причин использования частичного индекса является исключение индексации распространенных значений. Если значение содержится в значительном числе строк, то индекс, включающий эти строки, не будет полезен для большинства запросов. Исключение этих строк из индекса может повысить производительность запросов, использующих индекс, и уменьшить размер индекса.

```
CREATE TABLE access_log (
    url varchar,
    client_ip inet,
);

CREATE INDEX ON access_log (client_ip)
    WHERE url = '/index.html';
```

В приведенном выше примере создается индекс для столбца `client_ip`, но только для строк, где столбец `url` имеет значение `'/index.html'`. Этот индекс может быть использован для запросов, которые ищут строки с определенным значением `client_ip` и `url`, равным `'/index.html'`.

Частичный индекс может использоваться для запросов, которые не включают в себя условие, указанное в команде. Однако индекс может быть менее эффективным, чем индекс, который включает в себя условие.

```
SELECT * FROM access_log
    WHERE url = '/index.html' AND client_ip = inet '192.168.100.23';
```

Частичный индекс может использовать этот запрос:

```
SELECT * FROM access_log
    WHERE url = '/index.html' AND client_ip = inet '212.78.10.32';
```

Частичный индекс может использоваться, если запрос включает в себя условие `url`, указанное в индексе:

```
SELECT * FROM access_log
    WHERE url = '/index.html' AND client_ip = inet '192.168.100.23';
```

Этот тип запросов в первую очередь использует столбец `client_ip`, поэтому индекс будет использовать лучше всего использовать для столбца `client_ip`, а не для столбца `url`. Частичный индекс может быть полезен, если вы часто выполняете запросы, которые включают в себя только небольшое подмножество данных в таблице.

```
CREATE INDEX orders_unpaid_ON orders (order_no)
    WHERE paid is not true;
```

Однако индекс также можно использовать для запросов, которые не включают в себя условие, указанное в индексе.

```
SELECT * FROM orders WHERE paid is not true AND order_no < 10000;
```

Здесь индекс может использоваться, потому что строки, которые удовлетворяют предикату, также являются подмножеством строк, которые удовлетворяют условию `order_no < 10000`.

```
SELECT * FROM orders WHERE paid is not true AND amount < 5000.00;
```

Этот запрос не может использовать индекс, потому что строки, которые удовлетворяют предикату, могут не быть подмножеством строк, которые удовлетворяют условию `amount < 5000.00`.

```
SELECT * FROM orders WHERE order_no = 4261;
```

Здесь `NULL` может быть использован, потому что строки, которые удовлетворяют предикату, являются подмножеством строк, которые удовлетворяют условию `order_no = 4261`.

В этом примере показано, что индексы не используются, если они не нужны. Обращение к таблице может быть быстрее, чем использование индекса, если запрос возвращает большую часть строк в таблице. Частичные индексы могут быть полезны, если вы часто выполняете запросы, которые включают в себя только небольшое подмножество данных в таблице. Однако индекс также можно использовать для запросов, которые не включают в себя условие, указанное в индексе. Частичные индексы могут быть менее эффективными, чем индексы для отдельных столбцов, если запрос включает в себя только один столбец.

```
CREATE TABLE tests (
    subject text,
    target text,
    success boolean,
);

CREATE UNIQUE INDEX tests_success_constraint ON tests (subject, target)
    WHERE success;
```

Контроль использования индекса

Если индекс в PostgreSQL не используется, обслуживание его настройка, анализ скорость, может индекс для быстрых использования. Можно использовать команду `EXPLAIN`, убедиться в использовании индекса, но для этого нужно запустить запрос. Если в таблице много записей, это занимает время. Кроме того, запрос может изменить данные. Чтобы этого не происходило, можно использовать транзакцию с откатом изменений.

Есть несколько способов статистику использования индекса. Самый простой — воспользоваться представлением `pg_stat_user_indexes`. Например, такой запрос `idx_scan` покажет, сколько раз использовался каждый индекс, имя индекса и имя таблицы, для которой он создан:

```
SELECT
    relname,
    indexrelname,
    idx_scan
FROM pg_stat_user_indexes
ORDER BY idx_scan DESC;
```

Вот несколько дополнительных советов по использованию индексов:

- Всегда сначала выполняйте `ANALYZE`. Эта команда собирает статистику о распределении данных в таблице. Планировщик запросов использует эту статистику, чтобы принимать решения о том, какие индексы использовать. Если статистика устарела, планировщик может принять неоптимальное решение.
- Используйте `EXPLAIN ANALYZE`. Эта команда выполняет запрос и показывает план выполнения, который использовался планировщиком. Она также показывает фактическое время выполнения каждого шага. Это может помочь вам понять, почему индекс не используется.
- Не создавайте слишком много индексов. Каждый индекс занимает место на диске и замедляет операции вставки, обновления и удаления. Создавайте только те индексы, которые действительно необходимы.
- Рассмотрите возможность использования частичных индексов. Частичный индекс — это индекс, который включает только часть строк в таблице. Это может быть полезно, если вы часто выполняете запросы, которые включают только небольшое подмножество данных.
- Рассмотрите возможность использования составных индексов. Составной индекс — это индекс, который включает несколько столбцов. Это может быть полезно, если вы часто выполняете запросы, которые включают несколько столбцов.
- Если индекс не нужен, удалите его. Неиспользуемые индексы замедляют операции вставки, обновления и удаления. Вы можете использовать представление `pg_stat_user_indexes`, чтобы найти индексы, которые не использовались в течение длительного времени.
- Используйте `pg_stat_statements`, чтобы найти медленные запросы. Это расширение собирает статистику о выполнении всех запросов. Вы можете использовать эту статистику, чтобы найти запросы, которые выполняются медленно, и затем попытаться оптимизировать их.
- Используйте `auto_explain`, чтобы автоматически записывать планы выполнения медленных запросов. Это может помочь вам найти запросы, которые не используют индексы, даже если вы не знаете, что они медленные.
- Используйте `pg_stat_activity`, чтобы отслеживать текущие запросы. Это может помочь вам найти запросы, которые выполняются в данный момент.
- Используйте `pg_stat_bgwriter`, чтобы отслеживать активность фонового процесса записи. Это может помочь вам понять, как часто данные записываются на диск.
- Используйте `pg_stat_database`, чтобы отслеживать статистику базы данных. Это может помочь вам понять, как часто выполняются запросы, как часто данные читаются с диска и как часто данные записываются на диск.
- Используйте `pg_stat_user_tables`, чтобы отслеживать статистику пользовательских таблиц. Это может помочь вам понять, как часто выполняются запросы к таблицам, как часто данные читаются с диска и как часто данные записываются на диск.
- Используйте `pg_stat_user_indexes`, чтобы отслеживать статистику пользовательских индексов. Это может помочь вам понять, как часто используются индексы, как часто данные читаются с диска и как часто данные записываются на диск.
- Используйте `pg_stat_user_functions`, чтобы отслеживать статистику пользовательских функций. Это может помочь вам понять, как часто выполняются функции

Триггеры

Артем Ануфриев

Содержание

Назад к содержанию

- Обзор
- Типы триггеров
  - Триггеры уровня строки и уровня оператора
  - Триггеры BEFORE, AFTER и INSTEAD OF
- Операции с триггерами
  - Создание триггера
  - Изменение триггера
  - Отключение триггера
  - Удаление триггера
  - Отображение списка триггеров
- Особенности поведения триггеров
  - Триггер AFTER
  - Секционированные и родительские таблицы
  - Возвращаемые значения триггерных функций
  - Генерируемые столбцы
  - Порядок срабатывания триггеров
  - Условие триггера WHEN
  - Каскадные триггеры
  - Аргументы триггеров
- Видимость изменений данных

Обзор

Триггер - это указание, которое хотелось бы PostgreSQL должен автоматически вызывать определенную функцию при каждом выполнении операции указанного типа. Триггеры можно прикрепить к таблицам, представлениям и внешним таблицам.

Для любого из внешних таблицы можно определить триггеры, которые вызываются при выполнении операций INSERT, UPDATE или DELETE, либо один раз для каждой модифицированной строки, либо один раз для каждого оператора SQL. Триггеры UPDATE могут дополнительно также вызываться, чтобы определенные столбцы были указаны в списке UPDATE. Триггеры TRUNCATE срабатывают только при выполнении TRUNCATE.

Функция, вызываемая триггером для определения триггера. Она должна быть объявлена как функция без аргументов и возвращать тип TRIGGER. Сама по ч ее функции можно получить доступ к данным, которые описывают вызывающее событие.

Типы триггеров

Триггеры уровня строки и уровня оператора

PostgreSQL позволяет создавать триггеры для каждой строки или для каждого оператора. При использовании триггеров для каждой строки PostgreSQL вызывает триггерную функцию для каждой строки, которая обновляется, вставляется или удаляется оператором.

PostgreSQL вызывает триггеры для каждого оператора только один раз, независимо от того, сколько строк обновляется, вставляется или удаляется. Если оператор изменяет ноль строк, ни таких выполнение триггеров оператора (если они определены) все равно происходит.

Триггеры BEFORE, AFTER и INSTEAD OF

Триггеры также классифицируются в зависимости от того, срабатывают ли они до, после или вместо операции. Триггеры BEFORE, которые определяются для операторов, вызываются до начала выполнения каждого оператора. Триггеры AFTER вызываются после завершения каждого оператора. Эти типы триггеров могут быть определены для таблиц, представлений или внешних таблиц.

Триггеры BEFORE на уровне строки срабатывают перед обработкой конкретной строки. Они часто используются триггеры AFTER на уровне строки срабатывают в конце оператора, но до любых триггеров AFTER на уровне оператора. Эти типы триггеров не используются с таблицами или представлениями.

Триггеры INSTEAD OF можно определять только для представлений и только на уровне строки. Они срабатывают, чтобы ответить на каждую строку представления, которая будет обновлена оператором, вызывающим триггер представления.

Триггеры, которые BEFORE, AFTER или INSTEAD OF, могут также вызываться для операторов TRUNCATE, но только при выполнении TRUNCATE на уровне оператора, а не на уровне строки.

Операции с триггерами

Создание триггера

Для создания нового триггера используется следующий синтаксис:

1. Создать триггерную функцию.
2. Использовать оператор CREATE TRIGGER, чтобы привязать триггерную функцию к таблице.

Рассмотрим пример с триггерами. Для начала создадим таблицу books и триггерную функцию с отдельной таблицей. В триггере BEFORE UPDATE.

Существует таблица books, в которой хранится информация о книгах:

```
CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    book_name VARCHAR NOT NULL,
    price NUMERIC NOT NULL
);

INSERT INTO
    books( book_name, price)
VALUES
    ('Hyperion',25),
    ('War and Peace',15),
    ('1984',20),
    ('The Time Machine',10);
```

Создайте новую таблицу для сохранения изменений:

```
CREATE TABLE books_log (
    id SERIAL PRIMARY KEY,
    book_id INT,
    book_name VARCHAR NOT NULL,
    price NUMERIC NOT NULL,
    change_date TIMESTAMP NOT NULL
);
```

Создайте триггерную функцию, которая для этого специальная язык. Код имеет конструкт синтаксис. Триггерной функции на PL/pgSQL. Создайте переменную NEW, в которой хранится информация о новой строке:

```
CREATE OR REPLACE FUNCTION save_changes()
    RETURNS TRIGGER
    LANGUAGE PLPGSQL
AS
$$
BEGIN
    IF (NEW.book_name <> OLD.book_name) OR (NEW.price <> OLD.price) THEN
        INSERT INTO books_log(book_id,book_name,price,change_date)
        VALUES(OLD.id,OLD.book_name,OLD.price, now());
    END IF;

    RETURN NEW;
END;
$$
```

Когда функция PL/pgSQL вызывается триггером для изменения данных, она помещает специальную переменную с несколько специальными переменными. Они перечислены в таблице ниже. В приведенных выше определениях переменных NEW и OLD для доступа к новой и старой строке таблицы соответственно.

Переменные триггерной функции изменения данных PL/pgSQL

Имя	Тип данных	Описание
NEW	record	Новая строка таблицы для операций INSERT/UPDATE в триггере уровня строки. Эта переменная имеет значение NULL в триггере уровня оператора и для операций DELETE.
OLD	record	Старая строка таблицы для операций UPDATE/DELETE в триггере уровня строки. Эта переменная имеет значение NULL в триггере уровня оператора и для операций INSERT.
TG_NAME	name	Имя сработавшего триггера
TG_WHEN	text	BEFORE, AFTER или INSTEAD OF в зависимости от определения триггера
TG_LEVEL	text	ROW или STATEMENT в зависимости от определения триггера
TG_OP	text	Операция, для которой сработал триггер: INSERT, UPDATE, DELETE или TRUNCATE
TG_RELID	oid (reference pg_class.oid)	Идентификатор объекта таблицы, вызвавшей срабатывание триггера
TG_TABLE_NAME	name	Имя таблицы, вызвавшей срабатывание триггера
TG_TABLE_SCHEMA	name	Имя схемы, содержащей таблицу, вызвавшую срабатывание триггера
TG_NARGS	integer	Количество аргументов, передаваемых триггерной функции в операторе CREATE TRIGGER
TG_ARGV[]	Data type array of text	Аргументы оператора CREATE TRIGGER. Индекс считается от 0. Недопустимые индексы (меньшие 0 или меньшие или равные tg_nargs) возвращают NULL.

Для дополнительной информации обратитесь к статье Trigger functions.

Используйте оператор CREATE TRIGGER, чтобы связать триггер. Чтобы создать или заменить триггер повторно, выполнить должен иметь привилегию TRIGGER для таблицы, а также привилегию EXECUTE для функции триггера.

```
CREATE TRIGGER last_changes
    BEFORE UPDATE
    ON books
    FOR EACH ROW
    EXECUTE PROCEDURE save_changes();
```

Триггер last_changes создается как триггер BEFORE UPDATE и имеет строки таблицы books. Он вызывает триггерную функцию save_changes. За дополнительной информацией о синтаксисе CREATE TRIGGER обратитесь к статье CREATE TRIGGER.

Обновите таблицу books, чтобы активировать триггер:

```
UPDATE books SET price = 32 WHERE ID = 1;
```

Проверьте содержимое таблицы books_log:

```
SELECT * FROM books_log;
```

Результат:

```
 id | book_id | book_name | price |     change_date
----+---------+-----------+-------+---------------------
  1 |       1 | 1984      |    20 | 2024-04-04 14:38:07.851583
```

Изменение триггера

Используйте команду ALTER TRIGGER, чтобы изменить триггер.

В следующем примере триггер переименовывается:

```
ALTER TRIGGER last_changes ON books RENAME TO last_changes_new_name;
```

Отключение триггера

Чтобы временно отключить или повторно включить триггер, используйте команду ALTER TABLE с как показано ниже.

Отключение триггера:

```
ALTER TABLE books DISABLE TRIGGER last_changes_new_name;
```

Подключение триггера:

```
ALTER TABLE books ENABLE TRIGGER last_changes_new_name;
```

Удаление триггера

Чтобы удалить триггер, используйте команду DROP TRIGGER:

```
DROP TRIGGER last_changes_new_name ON books;
```

Чтобы выполнить эту команду, текущий пользователь должен владеть таблицы, для которой определен триггер.

Отображение списка триггеров

Для вывода списка всех триггеров в текущей базе данных, используйте системное представление information_schema.triggers:

```
SELECT event_object_table, trigger_name FROM information_schema.triggers;
```

Результат:

```
 event_object_table |  trigger_name
--------------------+-----------------
 books              | last_changes
```

Также можно использовать системный каталог pg_trigger для вывода списка триггеров:

```
SELECT oid, tgrelid, tgparentid, tgname FROM pg_trigger;
```

Результат:

```
  oid  | tgrelid | tgparentid |     tgname
-------+---------+------------+-----------------
 151888 |  150751 |          0 | last_changes
```

Особенности поведения триггеров

В этом разделе описываются некоторые особенности триггеров, которые могут оказаться важными.

Триггер AFTER

Выполнение триггера AFTER может быть отложено до конца транзакции, и не до конца оператора, это триггер AFTER по умолчанию CONSTRAINT. Все если случаях триггер выполняется не часть той же транзакции, что и выполнение, поэтому если оператор или триггер вызывает ошибку, результат этого триггера не может быть.

Секционированные и родительские таблицы

Оператор, указанный на родительской таблице в иерархии наследования или секционирования, не приводит к срабатыванию триггеров уровня оператора для затронутых дочерних таблиц; срабатывают только триггеры уровня строки. Например, INSERT с аргументом RETURNING в родительской таблице возвращает только строки, которые были фактически вставлены в родительскую таблицу. Однако триггеры уровня строки дочерних таблиц могут изменить данные, прежде чем они будут вставлены.

Триггеры уровня строки, определенные для родительских таблиц, также срабатывают для строк, которые изменяются в дочерних таблицах. Например, триггер BEFORE UPDATE, определенный в родительской таблице, срабатывает при обновлении строки в дочерней таблице, если только дочерняя таблица не имеет собственный триггер BEFORE UPDATE. Аналогично триггеры AFTER DELETE уровня строки срабатывают, если удаление затрагивает дочерние таблицы. Это поведение согласуется с ограничениями внешнего ключа, которые также распространяются на дочерние таблицы.

Триггеры уровня оператора определяются только для таблиц, на которые ссылается оператор в своем списке целей. Триггеры уровня оператора не срабатывают автоматически для таблиц, которые являются потомками таблиц, указанных в списке целей. Например, если DELETE применяется к таблице с потомками, он срабатывает триггеры DELETE уровня оператора, определенные для таблицы, указанной непосредственно в команде DELETE, но не для ее потомков. Напротив, триггеры уровня строки срабатывают для потомков, только если они существуют в целевой таблице.

Возвращаемые значения триггерных функций

Триггерная функция, написанная на любом языке, должна возвращать NULL или структурную функцию, возвращающую строку, таблицу значения типа RECORD. Триггер BEFORE на уровне строки может возвращать следующий результат:

- **NULL** - строка вставлена, обновлена или удалена. Для PostgreSQL это является отсутствием. Триггеры INSERT, UPDATE или DELETE для конкретной строки таблицы, а также последующие триггеры.
- **Строка таблицы** - только для триггеров INSERT и UPDATE, возвращаемую строку будет вставлена или изменен обновлена строку. Это позволяет триггерной функции изменять строку, которая будет сохранена.

Если не хотите вставлять или изменять строку, вызывающую триггер BEFORE уровня строки, вы должны вернуть NULL. Для этого можно использовать ту же строку, которая была передана - новую строку для триггеров INSERT и UPDATE или старую строку для DELETE. Неучтение возвращаемое значение приводит к тому, что строка не будет обработана.

Триггер INSTEAD OF на уровне строки должен возвращать либо NULL, чтобы указать, что он не изменяет никаких данных из базовых таблиц, либо новую строку (для INSERT и UPDATE) или старую строку (для DELETE), чтобы указать, что функция должна выполнить свою работу. Это указывает, что должно быть количество строк, затронутых INSERT или DELETE. Триггер INSTEAD OF на уровне строки может возвращать ненулевое значение, чтобы сигнализировать, что он выполнил требуемую работу.

Значение возвращаемое для триггеров уровня строки, игнорируется для триггеров уровня оператора, которые всегда возвращают NULL.

Генерируемые столбцы

Генерируемые столбцы вычисляют особого внимания. Генерируемый столбец вычисляется после триггера BEFORE и до триггера AFTER. Следовательно, изменение столбца, от которого зависит генерируемый столбец, в триггере BEFORE приведет к обновлению генерируемого столбца, если триггер AFTER в триггере BEFORE существует OLD. Значение столбца, сгенерированное значение, из NEW. Если генерируемый столбец в триггере BEFORE, то значение, которое будет записано в столбец, будет возвращено, независимо от того, что NEW возвращается в будет переписано.

Порядок срабатывания триггеров

Если для одного и того же события на одной и той же таблице определено несколько триггеров, они срабатывают в алфавитном порядке по имени триггера. В случае триггеров BEFORE и INSTEAD OF порча, вызывающая изменение триггером, становится входными данными для следующего триггера. Если какой-либо триггер BEFORE или INSTEAD OF возвращает NULL, операция для этой строки прекращается и последующие триггеры для этой строки не срабатывают.

Условие триггера WHEN

Определение триггера также может содержать логическое условие WHEN, которое проверяет, должен ли срабатывать триггер. В триггере уровня строки условие WHEN может проверять старые и новые значения столбцов строки. Триггеры уровня оператора также могут использовать условие WHEN, хотя для них это функция не так полезна. Условие WHEN триггера BEFORE может проверять только новые значения, поскольку старые значения еще не доступны. Условие WHEN триггера AFTER может проверять старые и новые значения.

Триггеры, для которых определено условие WHEN, могут не срабатывать, либо они могут работать в режиме «отложенного». Это требование предотвращает ситуацию, когда работа строки, для которой условие WHEN возвращает false, может быть нарушена последующим изменением строки триггером BEFORE, но не триггером AFTER, и возвращает true, чем необходимо ставить, поэтому в показано, что триггер AFTER должен быть запущен. Это может значительно ускорить выполнение операторов, изменяющих большое количество строк, в показано, что триггер, который должен срабатывать только в некоторых случаях. Триггеры INSTEAD OF не поддерживают условие WHEN.

Каскадные триггеры

Если триггерная функция выполняет команды SQL, он команды могут сами вызвать триггеры, это называется каскадными триггерами. Это совершенно нормально, и триггерные функции написаны с учетом этого возможного. Например, триггер, который вызывается, INSERT или UPDATE таблицы, может выполнять команды, которые вызывают собственные триггеры. Эти каскадные триггеры могут быть собственной логикой, поэтому вы должны быть осторожны при проектировании такого набора триггеров, чтобы избежать бесконечных рекурсий. Ограничение глубины вложенности не предусмотрено, реализующая большая часть, в показано, что триггеры, которые должны срабатывать только в некоторых случаях. Триггеры INSTEAD OF не поддерживают условие WHEN.

Аргументы триггеров

При определении триггера для него можно указать аргументы. Это обычная практика, позволяющая писать предоставляемая возможность различными способами. Аргументы триггера помещаются в TG_ARGV в функции, которая вызывается триггером. Индекс начинается с 0. Если указано недопустимое число (меньше 0 или больше или равно tg_nargs), возвращается NULL. Аргументы триггера не имеют никакого другого значения. Они передаются функции триггера, которая может использовать их по своему усмотрению, и не влияют на сам триггер. Если вы планируете использовать аргументы, они должны быть согласованы между собой, и функция триггера должна быть согласована с определением триггера - другими словами, что ожидает функция триггера, должно соответствовать тому, что указано в CREATE TRIGGER. Для триггеров уровня строки старые и новые значения столбцов строки доступны через NEW и OLD, для доступа к новой и старой строке таблицы соответственно.

Видимость изменений данных

Если триггерные функции, вызываемые триггерами, или функции, вызываемые ими, должны в соответствии со следующими правилами видимости данных:

- Что одно из изменений, внесенных оператором, видимо триггерам BEFORE, которые не все изменения внешних триггеров уровня оператора BEFORE.
- Что любые изменения, внесенные оператором, видимы для триггеров AFTER, вызванных оператором, либо вызванных триггером BEFORE на уровне строки. Поскольку они еще не произошли.

К команде SQL, выполняемым триггером BEFORE, он может видеть последствия изменений данных, сделанных предыдущими действиями, вызванными тем же оператором. В этому следует подходить с осторожностью, поскольку порядок SQL, обрабатывающих несколько строк, может обращаться к строкам в любом порядке.

Триггер INSTEAD OF на уровне строки может видеть последствия изменений данных, сделанных предыдущими действиями того же триггера INSTEAD OF и той же команде.

К тогда срабатывают триггеры AFTER на уровне строки могут видеть последствия изменений данных, сделанных предыдущими командой для операторов и запроса в выражения SELECT.

Если триггерная функция написана на любом из стандартных процедурных языков, выполняемые внутри различий версии, только если функция объявлена как VOLATILE. Функции, объявленные как STABLE или IMMUTABLE, не имеют доступа к изменениям, внесенным вызывающей командой.

Пользовательские функции и процедуры

Антон Бобров

Содержание

Назад к содержанию

Обзор
- Пользовательские функции
- Пользовательские процедуры

Функции на языке запросов SQL
- SQL-функции с базовыми типами
- SQL-функции с составными типами
- SQL-функции с выходными параметрами
- SQL-процедуры с выходными параметрами
- SQL-функции с переменными числом аргументов
- SQL-функции, возвращающие множества
- SQL-функции, возвращающие таблицы

Функции на процедурных языках

Обзор

Пользовательские функции

В PostgreSQL существуют следующие виды пользовательских функций:

- [Функции на языке запросов SQL](#) — Подробно тут: sql-functions.html
- [Функции на процедурных языках](#) — функции, написанные на языках, похожих на C, и "более полноценных языках (Postgres Pro Enterprise поддерживает PL/Perl, PL/Python, PL/Tcl, PL/Java, PL/Lua, PL/pgSQL, PL/Python, PL/Tcl, PL/Perl, PL/Java, PL/Lua, PL/pgSQL)
- Внутренние функции — функции, написанные на языке программирования Си, которые являются частью ядра PostgreSQL. Встроенные функции обеспечивают доступ к внутренним системным частям ядра. Пользователи могут использовать [CREATE FUNCTION](#) для создания дополнительных внутренних функций. Обычно внутренние функции помечаются как [internal language](#).
- Функции языка Си — функции, написанные на Си. [CREATE FUNCTION](#) для динамической загрузки объекта (также называемого разделяемым библиотекой) с функцией, написанной на Си. Динамически загружаемый объект должен быть скомпилирован для [PG_MODULE_MAGIC](#). Для получения дополнительной информации см. [Compiling Extensions](#).

Каждый тип функции может принимать [параметры](#) и возвращать [результат](#). Подробнее тут: sql-syntax-calling-funcs.html. Функции также могут возвращать большие [наборы](#) результатов, которые называются [наборами результатов](#).

Функции могут быть [перегружены](#), что означает, что одно и то же имя может использоваться для [PG_FUNCTION_INFO_V1](#) различных функций, при условии, что они принимают [CREATE FUNCTION](#) различные типы входных параметров. Набор типов входных параметров [CREATE FUNCTION](#) не обязательно должен быть полным.

Пользовательские процедуры

Процедуры — это объект базы данных, аналогичный функции и имеющий следующие отличия:

- Процедура создается с помощью команды [CREATE PROCEDURE](#).
- Процедуры не возвращают значение, поэтому [RETURNS](#) не содержит выражения [RETURNS](#). Однако процедура могут изменять данные, которые получает.
- Функция может быть вызвана в операторе [SELECT](#), процедура — нет. Процедура вызывается отдельно с помощью [CALL](#).
- Процедура может выполнять ряд операций [TRANSACTION](#) внутри себя. Например, [START TRANSACTION](#), [COMMIT](#) и [ROLLBACK](#), которые недоступны в функциях.
- Некоторые атрибуты функции (например, [STRICT](#)) относятся к процедурам. Это атрибуты относятся и к вызову процедуры.

Процедуры также отличаются от функций тем, что [CREATE PROCEDURE](#) не [RETURNS](#) [CREATE FUNCTION](#) [RETURNS](#).

Функции на языке запросов SQL

SQL-функции выполняют произвольный список операторов SQL, возвращая результат последнего запроса в списке. Используйте команду [CREATE FUNCTION](#) в PostgreSQL, чтобы определить новую функцию.

В простом случае перечисляются параметры функции, тип результата, который она возвращает, и [тело](#) функции, которое должно быть [RETURNS](#) [SELECT](#) запросом. Альтернативный способ описания функции на языке SQL — указать имена и типы ее параметров, а также [RETURNS](#) [SELECT](#) запрос, в котором параметры функции могут быть [RETURNS](#) по их именам. Например, [RETURNS](#) [SELECT](#) [CREATE FUNCTION](#) [RETURNS](#) [SELECT](#) [RETURNS](#) [SELECT](#) [RETURNS](#) [SELECT

Уровни изоляции транзакций

Д. Евгения Кузина

Содержание

Назад к содержанию

Уровень изоляции Read committed

Уровень изоляции Repeatable read

Уровень изоляции Serializable

Стандарт SQL определяет четыре уровня изоляции транзакций. Самый строгий — Serializable. Согласно стандарту SQL, любое одновременное выполнение набора сериализуемых транзакций должно давать тот же эффект, что и их поочередное выполнение в некотором порядке. Остальные три уровня определяются через описание допустимых аномалий, которые являются результатом взаимодействия между параллельными транзакциями. Определение аномалии не должно возникать на определенном уровне изоляции. Ни одна из этих аномалий несовместима на уровне Serializable.

Могут возникнуть следующие аномалии:

- **Грязное чтение (dirty read)** — транзакция считывает данные, записанные параллельной незафиксированной транзакцией.
- **Неповторяемое чтение (non-repeatable read)** — транзакция перечитывает ранее прочитанные данные и обнаруживает, что данные были изменены другой транзакцией (которая была зафиксирована после момента первоначального чтения).
- **Фантомное чтение (phantom read)** — транзакция повторно выполняет запрос, который возвращает набор строк для некоторого условия, и обнаруживает, что набор строк, удовлетворяющих условию, изменился из-за другой недавно зафиксированной транзакции.
- **Аномалия сериализации (serialization anomaly)** — результат успешной фиксации группы транзакций несовместим со всеми возможными порядками выполнения этих транзакций по одной.

Уровни изоляции транзакций описаны в таблице ниже. Значение `Не разрешено в ADPG/PostgreSQL` используется для аномалий, которые разрешены стандартом SQL, но запрещены в ADPG/PostgreSQL.

Уровень изоляции	Грязное чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
Read uncommitted	Не разрешено в ADPG/PostgreSQL	Допускается	Допускается	Допускается
Read committed	Не допускается	Допускается	Допускается	Допускается
Repeatable read	Не допускается	Не допускается	Не разрешено в ADPG/PostgreSQL	Допускается
Serializable	Не допускается	Не допускается	Не допускается	Не допускается

В ADPG/PostgreSQL вы можете запросить любой из четырех стандартных уровней изоляции транзакций, но внутри реализованы только три различные уровня изоляции. Режим PostgreSQL Read uncommitted ведет себя как Read committed. Такое решение позволяет сопоставить стандартные уровни изоляции с архитектурой управления многоверсионным параллелизмом PostgreSQL.

В таблице также показано, что реализация PostgreSQL Repeatable read не допускает фантомного чтения. Это приемлемо в соответствии со стандартом SQL, поскольку стандарт определяет, какие аномалии не должны возникать на определенных уровнях изоляции. Более строгие ограничения допустимы.

> **ВАЖНО**
>
> Некоторые типы данных и функции PostgreSQL имеют специальные правила относительно поведения транзакций. В частности, изменения, внесенные в последовательность, не следовательно, счетчик столбца, объявленный как `serial`), немедленно видны всем другим транзакциям и не откатываются, если транзакция, внесшая изменения, прерывается.

Уровень изоляции по умолчанию в ADPG/PostgreSQL — Read committed. Чтобы убедиться в этом, выполните следующую команду `psql`:

```
SHOW TRANSACTION ISOLATION LEVEL;
```

Результат:

```
 transaction_isolation
-----------------------
 read committed
(1 row)
```

Чтобы выбрать желаемый уровень изоляции транзакции, используйте команду SET TRANSACTION.

```
BEGIN;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
...
COMMIT;
```

Также возможно установить уровень изоляции транзакций в командах BEGIN и START TRANSACTION.

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
...
COMMIT;

START TRANSACTION ISOLATION LEVEL SERIALIZABLE;
...
COMMIT;
```

Уровень изоляции Read committed

Как упоминалось выше, Read committed — это уровень изоляции по умолчанию в ADPG/PostgreSQL. Когда транзакция использует этот уровень изоляции, запрос `SELECT` (без выражения `FOR UPDATE/SHARE`) видит только данные, зафиксированные до начала запроса. Он никогда не обращается ни к незафиксированным данным, ни к изменениям, зафиксированным в ходе выполнения запроса параллельными транзакциями. То есть запрос `SELECT` видит снимок базы данных в момент начала выполнения запроса. Однако `SELECT` видит результаты предыдущих обновлений, выполненных в рамках его собственной транзакции, даже если они еще не были зафиксированы. Также обратите внимание, что два последовательных оператора `SELECT` могут получать разные данные, даже если они выполняются в рамках одной транзакции, если изменения, внесенные другой транзакцией после запуска первого `SELECT`, до запуска второго `SELECT`.

Команды `UPDATE`, `DELETE`, `SELECT FOR UPDATE` и `SELECT FOR SHARE` ведут себя так же, как `SELECT`, с точки зрения поиска целевых строк: они находят только целевые строки, которые были зафиксированы в момент запуска команды. Однако строка, которая может быть уже обновлена, удалена или заблокирована другой параллельной транзакцией к моменту обнаружения. В этом случае запланированное изменение будет отложено до фиксации первой транзакции или ее отката, если она все еще выполняется. Если первая транзакция откатывается, вторая может продолжить обновление первоначально найденной строки. Если первая транзакция фиксируется, вторая попытается применить свои операции к обновленной версии строки. Если применяется строгий уровень изоляции, если первая транзакция удалила ее. Условие поиска (выражение `WHERE`) применяется после того, как строка была заблокирована, чтобы проверить, соответствует ли обновленная строка этому условию. Если это так, вторая транзакция продолжается с измененной версией строки. В случае `SELECT FOR UPDATE` и `SELECT FOR SHARE` это означает, что обновленная версия строки блокируется и возвращается клиенту.

`INSERT` с выражением `ON CONFLICT DO UPDATE` ведет себя аналогично. В режиме Read committed каждая строка, предложенная для вставки, будет либо вставлена, либо обновлена. Если конфликты вызваны другой транзакцией, результат которой еще не виден для `INSERT`, выражение `UPDATE` подействует на эту строку, даже несмотря на то, что эта команда может не видеть никакую версию этой строки.

В режиме Read committed, если вы выполните команду `INSERT` с выражением `ON CONFLICT DO NOTHING`, строка может не быть добавлена в результате другой транзакции, эффект которой не видно за снепшот команды `INSERT`.

Из-за вышеуказанных правил команда обновления может увидеть несогласованное состояние: она может видеть результаты параллельных команд, изменяющих те же строки, что пытается изменить она, но при этом не видит результаты этих команд в других строках таблиц. Такое поведение делает режим Read committed непригодным для команд, логически сложных с несколькими командами. Однако во многих ситуациях это и более простые случаи. Например, рассмотрим обновление банковских балансов — перевод денег с одного счета на другой с помощью следующих команд:

```
BEGIN;
UPDATE accounts SET balance = balance - 1000.00 WHERE acc_number = 5432;
UPDATE accounts SET balance = balance + 1000.00 WHERE acc_number = 1234;
COMMIT;

BEGIN;
UPDATE accounts SET balance = balance - 1000.00 WHERE acc_number = 4543;
UPDATE accounts SET balance = balance + 1000.00 WHERE acc_number = 1234;
COMMIT;
```

Если эти две транзакции одновременно попытаются изменить баланс счета `1234`, вторая транзакция должна работать с измененной версией строки.

```
BEGIN;
UPDATE accounts SET balance = balance - 1000.00 WHERE acc_number = 5432;

/* Выполнение из другой сессии:
BEGIN;
UPDATE accounts SET balance = balance - 1000.00 WHERE acc_number = 4543;
UPDATE accounts SET balance = balance + 1000.00 WHERE acc_number = 1234;
COMMIT;
*/

UPDATE accounts SET balance = balance + 1000.00 WHERE acc_number = 1234;
COMMIT;
```

Поскольку каждая команда влияет только на определенную строку, если команда увидит измененную версию строки, это не приведет к проблеме несогласованности.

Более сложные сценарии могут привести к нежелательным результатам в режиме Read committed. Например, рассмотрим команду `DELETE`, работающую с данными, которые одновременно изменяются. Предположим, что мы удаляем строку из таблицы `website` — это таблица из двух строк, где `website.hits` равны `9` и `10`. Выполним следующие команды:

```
BEGIN;
UPDATE website SET hits = hits + 1;
/* Выполнено из другой сессии: DELETE FROM website WHERE hits = 10;
COMMIT;
```

Команда `DELETE` не будет иметь никакого эффекта, даже если до и после `UPDATE` есть строка с полем `hits`, равным `10`. Это происходит потому, что значение `9` перед обновлением пропускается, а когда `UPDATE` завершается и `DELETE` получает блокировку, новое значение строки больше не `10`, а `11`, что не соответствует условию.

Частичная изоляция транзакций, обеспечиваемая режимом Read committed, подходит для многих приложений, и этот режим достаточно быстр и прост в использовании. Однако этого недостаточно для всех случаев. Приложениям, выполняющим сложные запросы и обновления, может потребоваться более строго согласованное представление базы данных, чем обеспечивает режим Read committed.

Уровень изоляции Repeatable read

В режиме Repeatable read видны только те данные, которые были зафиксированы до начала транзакции, и незафиксированные данные и изменения, внесенные другими транзакциями во время выполнения транзакции, не видны. Однако запрос видит результаты предыдущих обновлений, выполненных в рамках его транзакции, даже если они еще не зафиксированы. Это предотвращает все явления, описанные в таблице аномалий, за исключением аномалий сериализации.

Этот уровень отличается от Read committed тем, что запрос в транзакции Repeatable read видит снепшот данных в начале первого оператора транзакции (команды управления транзакцией не учитываются). Последующие команды `SELECT` в одной и той же транзакции будут идти и те же данные — они не видят изменений, внесенных и зафиксированных другими транзакциями после начала текущей транзакции.

Приложения, использующие этот уровень, должны быть готовы к повторным попыткам транзакций из-за сбоев сериализации.

Команды `UPDATE`, `DELETE`, `SELECT FOR UPDATE` и `SELECT FOR SHARE` ведут себя как `SELECT`: при поиске целевых строк. Они находят только те целевые строки, которые были зафиксированы на момент начала текущей транзакции. Однако к моменту обнаружения эти целевые строки могут уже быть изменены, удалены или заблокированы другой параллельной транзакцией. В этом случае транзакция в режиме Repeatable read ожидает завершения первой транзакции, изменяющей данные (если она все еще выполняется). Если первая транзакция откатывается, вторая транзакция может продолжить обновление первоначально найденной строки. Если первая транзакция фиксируется и обновляет или удаляет строку, а не просто блокирует ее, транзакция Repeatable read будет отменена с сообщением `ERROR: could not serialize access due to concurrent update`. Транзакция Repeatable read не может изменять или блокировать строки, изменяемые после ее начала. Когда произошли попытки подобного обновления, будет выдана ошибка:

Содержание

- Блокировки на уровне таблицы
- Блокировки на уровне строки
- Команда LOCK
- Просмотр блокировок
- Взаимоблокировки
- Рекомендательные блокировки

В зависимости от типа объекта блокировки бывают разные.

- Блокировки на уровне таблицы – блокируют таблицу целиком.
- Блокировки на уровне строки – блокируют определенные строки.

Также блокировки могут быть:

- **исключительными (exclusive)** – предназначены могут удерживаться только одной транзакцией.
- **разделяемыми (shared)** – могут удерживаться несколькими транзакциями.

Блокировки на уровне таблицы

В таблице можно установить различные режимы блокировки в строке, в которых не используется ACID(PostgreSQL). Вы также можете получить набор из этих блокировок вам нужно использовать команду LOCK. Все режимы блокировок таблиц будут конфликтовать с режимом блокировки доступа, связанным с этим. Тип: Accessexclusivelock.

Блокировки на уровне таблицы

Название	Использование блокировки
ACCESS SHARE (AccessShareLock)	Команда `SELECT` получает эту блокировку для работы с таблицами, в целом, любой запрос, которые только читает таблицу и не изменяет ее, получает этот режим блокировки
ROW SHARE (RowShareLock)	Команда `SELECT FOR UPDATE` и `SELECT FOR SHARE` получает эту блокировку для работы с таблицами, а также любой запрос, который получает блокировку `ACCESS SHARE`
ROW EXCLUSIVE (RowExclusiveLock)	Команда `UPDATE`, `DELETE` и `INSERT` получают этот режим блокировки для обрабатываемой таблицы. Обычно этот режим блокировки получают запросы, которые изменяют данные в таблице
SHARE UPDATE EXCLUSIVE (ShareUpdateExclusiveLock)	Этот режим получают `VACUUM` (без `FULL`), `ANALYZE`, `CREATE INDEX CONCURRENTLY`, `CREATE STATISTICS`, `COMMENT ON` и некоторые формы `ALTER INDEX` и `ALTER TABLE` (более информацию смотрите в документации этих команд)
SHARE (ShareLock)	Эту блокировку получает `CREATE INDEX` (без `CONCURRENTLY`).
SHARE ROW EXCLUSIVE (ShareRowExclusiveLock)	Эту блокировку получает `CREATE TRIGGER` и некоторые формы `ALTER TABLE`.
EXCLUSIVE (ExclusiveLock)	Допускает только одновременные блокировки `ACCESS SHARE` – параллельно с транзакцией, которая получила эту блокировку. Получает эту блокировку `REFRESH MATERIALIZED VIEW CONCURRENTLY`.
ACCESS EXCLUSIVE (AccessExclusiveLock)	Гарантирует, что только транзакция может использовать эту таблицу. Эту блокировку получают `ALTER TABLE`, `DROP TABLE`, `TRUNCATE`, `REINDEX`, `CLUSTER`, `VACUUM FULL` и `REFRESH MATERIALIZED VIEW` (без `CONCURRENTLY`). Также эту блокировку получают некоторые формы `ALTER INDEX` и `ALTER TABLE`. Получает эту блокировку `LOCK TABLE`. Также `ACCESS EXCLUSIVE` получают команды `SELECT` без `ACCESS EXCLUSIVE` использует `LOCK TABLE`. Также `ACCESS EXCLUSIVE` получают команды `SELECT` без `FOR UPDATE/SHARE`.

Основная различия между одним режимом блокировки и другим – это набор режимов, с которыми он конфликтует. Две блокировки конфликтуют, если они не могут удерживаться одновременно разными сеансами (транзакциями). Для примера и его продолжения смотрите ниже. Хотя все одинаковые таблицы. При этом разные блокировки могут конфликтовать друг с другом, даже если они принадлежат разным строкам (поскольку область действия блокировки таблицы – вся таблица). Блокировки, которые были получены в ходе транзакции, всегда освобождаются во время ее завершения. Они неявно снимаются в конце транзакции, независимо от того, была ли она зафиксирована или отменена. Кроме того, некоторые команды управления данными используют блокировки таблиц, поскольку они работают вне контекста транзакций. Они включают `ACCESS SHARE` блокировки, которые защищают таблицы, на которые ссылаются во время выполнения `pg_dump`.

Конфликтующие режимы блокировки уровня таблицы

Запрашиваемый режим блокировки	Существующий режим блокировки							
	ACCESS SHARE	ROW SHARE	ROW EXCLUSIVE	SHARE UPDATE EXCLUSIVE	SHARE	SHARE ROW EXCLUSIVE	EXCLUSIVE	ACCESS EXCLUSIVE
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCLUSIVE				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

После получения блокировки обычно удерживается до конца транзакции. Но если блокировка была получена после точки сохранения, она освобождается сразу при откате сохранения.

Блокировки на уровне строки

Помимо блокировок на уровне таблицы, ACID(PostgreSQL) использует блокировки на уровне строки. Тогда блокировки, в которых не используется ACID(PostgreSQL), называются блокировками на уровне строки.

Блокировки на уровне строки

Название	Использование блокировки
FOR UPDATE	`FOR UPDATE` устанавливает блокировку, которая препятствует `SELECT`, или для изменения. Это блокировка, которая получается `UPDATE`, `DELETE` или `SELECT FOR UPDATE` для строк, затронутых этими командами. `SELECT FOR UPDATE` также получает `SELECT FOR KEY SHARE`, `SELECT FOR SHARE` и `SELECT FOR NO KEY UPDATE`. `SELECT FOR SHARE` — `SELECT FOR KEY SHARE`, `SELECT FOR NO KEY UPDATE` — `SELECT FOR UPDATE`.
FOR NO KEY UPDATE	Работает аналогично `FOR UPDATE`, но блокировка менее строгая. Она получается командой `UPDATE`, которая не использует `FOR UPDATE`. Эта блокировка блокирует другие попытки `DELETE` или `SELECT FOR UPDATE`, которые затронут эти строки, но не препятствует другим командам `UPDATE`, которые не получают эту блокировку, или `SELECT FOR KEY SHARE`, которые не получают блокировку.
FOR SHARE	Работает аналогично `FOR NO KEY UPDATE`, но получает более слабую блокировку. Она получается командой `SELECT` с опцией `FOR SHARE`. Эта блокировка блокирует `UPDATE`, `DELETE`, `SELECT FOR UPDATE` или `SELECT FOR NO KEY UPDATE`, но не препятствует другим командам `SELECT FOR SHARE` или `SELECT FOR KEY SHARE`.
FOR KEY SHARE	Работает аналогично `FOR SHARE`, но блокировка слабее. Блокировка получается `SELECT FOR KEY SHARE`. Эта блокировка блокирует только `SELECT FOR UPDATE` и `UPDATE`, которые изменяют ключевые значения строки. `DELETE` или `UPDATE`, которые не изменяют ключевые значения, не блокируются. `SELECT FOR KEY SHARE` блокирует `SELECT FOR SHARE`.

Обратите внимание, что `FOR UPDATE` и `FOR NO KEY UPDATE` конфликтуют с блокировкой, которую `FOR KEY SHARE` получает, но не с `FOR SHARE`. Это потому, что `FOR SHARE` не берет эксклюзивную блокировку. Две транзакции могут одновременно получать блокировки `FOR KEY SHARE` и `FOR SHARE` для одной и той же строки. Блокировки на уровне строки не влияют на выборку данных. Они предназначают только запись и удаление данных. Блокировки на уровне строки, используемые `FOR UPDATE`, `FOR SHARE` и т. д., сохраняются до конца транзакции. Но если точки сохранения создаются в ходе транзакции, блокировки, полученные после точки сохранения, освобождаются при откате точки сохранения.

Конфликтующие режимы блокировки уровня строки

Запрашиваемый режим блокировки	Существующий режим блокировки			
	FOR KEY SHARE	FOR SHARE	FOR NO KEY UPDATE	FOR UPDATE
FOR KEY SHARE				X
FOR SHARE			X	X
FOR NO KEY UPDATE		X	X	X
FOR UPDATE	X	X	X	X

PostgreSQL не ограничивает количество строк в памяти, поэтому нет ограничений на количество блокировок строк, которые можно заблокировать. Однако блокировка строки может привести к дисковой записи, например, если `SELECT FOR UPDATE` изменяет строки, помещенные на страницу диска, которая еще не была записана. Команда `LOCK` также влияет на данные.

Команда LOCK

Команда `LOCK` устанавливает блокировку таблицы уровня транзакции, ожидая любых конфликтующих блокировок. Команда `LOCK` необходима для транзакций, которым нужно гарантировать, что нужные таблицы не будут изменены другими транзакциями в течение всей транзакции. Команда `LOCK` без опции `NOWAIT` ставит запрос в очередь, если конфликтующие блокировки существуют, и ждет, пока они не будут удалены. Если `NOWAIT` указан, `LOCK` не ожидает, чтобы получить запрошенную блокировку, если конфликтующие блокировки уже существуют. Команда `LOCK` ожидает любых конфликтующих блокировок, а затем получает запрошенную блокировку. Если `NOWAIT` указан, `LOCK` не ожидает, чтобы получить блокировку, но вместо этого сообщает об ошибке, если блокировка не может быть получена немедленно. Поведение `LOCK` с `NOWAIT` зависит от того, какие блокировки должны быть получены. Одна таблица может иметь несколько блокировок, и общее поведение команды является мягким. Наиболее ограничительный режим блокировки через конфликт с другими, менее ограничительными режимами, в зависимости от того, какие режимы блокировки каждый сеанс запрашивает и удерживает. Чтобы запросить блокировку определенного режима, укажите требуемый режим блокировки. Чтобы запросить эксклюзивную блокировку доступа, используйте `ACCESS EXCLUSIVE` – наиболее ограничительный режим.

```
BEGIN;
LOCK TABLE author IN SHARE MODE;
SELECT id FROM author WHERE author_name='Alexey';
id
---
(0 rows)
INSERT INTO book (id, title, author_id)
VALUES (1, 'Learn Rust', 1),
       (2, 'Learn Python', 1),
       (3, 'The Tao Testing', 1);
INSERT 0 3
COMMIT;
```

```
LOCK [ TABLE ] [ ONLY ] name [ * ] [, ...] [ IN lockmode MODE ] [ NOWAIT ]
```

Где:

- `name` – имя существующей таблицы, при этом `ONLY` ограничивает обработку только указанной таблицей. Если `ONLY` не указано, таблица и все ее производные таблицы (если они существуют) будут заблокированы. Необязательно `*` может быть указано для явного включения производных таблиц.
- `lockmode` – режим блокировки, которая может называться `ROW EXCLUSIVE`, `SHARE` или `ACCESS EXCLUSIVE`. Если режим блокировки не указан, используется самый ограничительный режим, `ACCESS EXCLUSIVE`.
- `NOWAIT` – указывает, что `LOCK` не должен ожидать освобождения любых конфликтующих блокировок: если указанная блокировка не может быть получена немедленно без ожидания, транзакция отменяется. Команда `LOCK` в `NOWAIT` режиме будет по-прежнему блокировать, если конфликтующие блокировки были получены, но не освобождены. Если `NOWAIT` указан, `LOCK` не ожидает, чтобы получить блокировку, но вместо этого сообщает об ошибке, если блокировка не может быть получена немедленно.

`LOCK` не необходимость вводит ограничения для конкретных строк. Команда `LOCK` не блокирует всю строку. Вместо этого `LOCK` блокирует строки, которые `UPDATE` или `SELECT FOR UPDATE` будут блокировать. Однако, как только транзакция получает блокировку строки, она будет удерживать эту блокировку до конца транзакции. Таким образом, `LOCK` может предотвратить конфликты блокировки строк. Например, `LOCK` может использоваться для избежания взаимоблокировок. Например, если транзакция собирается обновить несколько строк, с помощью `SELECT FOR UPDATE`, первым получает блокировку строки, которую `UPDATE` изменяет. Если `UPDATE` выполняется после того, как другие транзакции уже имеют блокировки для одной или нескольких строк, которые он будет изменять, он будет ждать, пока транзакции, которые содержат конфликтующие блокировки, не завершатся, а затем продолжит работу. Если транзакция, которая будет выполнять обновление, сначала получает блокировку строки, которую она будет изменять, то блокировка будет удерживаться в течение периода, когда она ожидает, пока любые конфликтующие блокировки будут освобождены. Это приводит к тому, что эта транзакция ведет себя последовательно с любыми другими транзакциями, которые получают блокировки на одной строке. Любые отложенные изменения будут выполнены, а последующие изменения будут продолжены нормально. Если `UPDATE` или `DELETE` изменяет строку, которая уже была заблокирована, она будет ждать, пока транзакция, которая содержит существующую блокировку, не завершится, а затем продолжит работу. Возможна ситуация, когда одна транзакция ожидает завершения другой транзакции, которая уже содержит блокировку строки, которую первая транзакция должна изменить. В этом случае ожидающая транзакция

Пример работы с таблицами через psql

Содержание

Обзор

Шаг 1. Создание базы данных

Шаг 2. Создание таблицы

Шаг 3. Получение информации о таблице

Шаг 4. Добавление новых данных в таблицу

Шаг 5. Чтение данных из таблицы

Использование WHERE

Сортировка результатов

Группировка результатов

Соединение таблиц

Комбинация SELECT + CREATE + INSERT

Шаг 6. Обновление данных в таблице

Шаг 7. Удаление данных из таблицы

Шаг 8. Изменение структуры таблицы

Переименование таблицы

Переименование столбца

Удаление индекса

Шаг 9. Удаление таблицы

Назад к практикам

Обзор

Работа без super пользователя с PostgreSQL. Используем [psql](#) — CLI-клиент для PostgreSQL. Для этого создаем пользователя и базу данных, которыми будем управлять. Пользователь будет иметь ограниченные права, только для работы с этой базой данных.

Сначала нужно создать [роль](#) пользователя, которая будет владеть базой данных. Затем создаем [базу данных](#), которая будет принадлежать этой роли. Подключаемся к базе данных [psql](#) и работаем с ней.

```
$ sudo -u postgres

# psql

psql (14.5)
Type "help" for help.

postgres=# \q
```

Теперь нужно [создать](#) пользователя и базу данных. Для этого подключаемся к PostgreSQL от имени пользователя postgres.

```
$ sudo -u postgres psql
```

Теперь нужно создать роль и базу данных. Для этого используем [SQL](#) команды.

```
CREATE ROLE test_role;

CREATE DATABASE test_db OWNER test_role;
```

Теперь нужно подключиться к базе данных test_db от имени test_role.

```
\c test_db test_role

You are now connected to database "test_db" as user "test_role".
```

Шаг 1. Создание базы данных

Создаем базу данных [SQL](#) командой [CREATE DATABASE](#). Имя базы данных должно начинаться с буквы и может содержать [буквы](#), цифры и символ подчеркивания.

```
CREATE DATABASE имя_базы_данных;
```

Создадим базу данных с именем test_db.

```
CREATE DATABASE test_db;
```

Теперь нужно подключиться к базе данных. Для этого используем [SQL](#) команду [\c](#).

```
\c имя_базы_данных
```

Подключимся к базе данных test_db.

```
\c test_db
```

Теперь мы можем работать с базой данных test_db.

Шаг 2. Создание таблицы

Создаем таблицу с помощью [SQL](#) команды [CREATE TABLE](#). Имя таблицы должно начинаться с буквы и может содержать [буквы](#), цифры и символ подчеркивания. Для каждого столбца нужно указать [тип данных](#).

Создадим таблицу с именем table_name. Для столбца id используем [тип данных](#) [SERIAL](#), который [автоматически](#) создает [последовательность](#) и [INTEGER](#) столбец. Для столбца name используем тип данных [VARCHAR](#) с максимальной длиной 255 символов. Для столбца age используем тип данных [INTEGER](#).

```
CREATE TABLE table_name (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    age INTEGER
);
```


Автоматический

Если нужно, чтобы столбец id автоматически [увеличивался](#) при [вставке](#) новых [строк](#), можно использовать [SERIAL](#) тип данных. Он автоматически создает [последовательность](#) и [INTEGER](#) столбец.


```
CREATE TABLE table_name (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    age INTEGER
);
```

Можно также создать [последовательность](#) и [INTEGER](#) столбец вручную.

```
CREATE SEQUENCE table_name_id_seq;

CREATE TABLE table_name (
    id INTEGER PRIMARY KEY DEFAULT nextval('table_name_id_seq'),
    name VARCHAR(255),
    age INTEGER
);

ALTER SEQUENCE table_name_id_seq OWNED BY table_name.id;
```

Для [автоматического](#) увеличения [последовательности](#) при [вставке](#) новых [строк](#) можно использовать [функцию](#) [nextval](#).

```
INSERT INTO table_name (name, age) VALUES ('John', 30);

INSERT INTO table_name (id, name, age) VALUES (nextval('table_name_id_seq'), 'Jane', 25);
```

Можно также создать таблицу с [ограничениями](#) [NOT NULL](#) и [UNIQUE](#).

```
CREATE TABLE table_name (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE,
    age INTEGER
);
```

Можно также создать таблицу с [внешним ключом](#) [FOREIGN KEY](#).

```
CREATE TABLE table_name (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255),
    email VARCHAR(255),
    age INTEGER,
    other_table_id INTEGER,
    FOREIGN KEY (other_table_id) REFERENCES other_table(id)
);
```

Шаг 3. Получение информации о таблице

Чтобы проверить созданную таблицу и получить информацию о ней, можно использовать следующие команды.

[SQL](#) команда [\dt](#) показывает список таблиц в текущей базе данных. Команда [\dt+](#) показывает то же самое, но с [дополнительной](#) информацией.

```
\dt
```

Пример вывода

```
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | table_name | table | test_role
(1 row)
```

[SQL](#) команда [\d](#) показывает [структуру](#) таблицы. Команда [\d+](#) показывает то же самое, но с [дополнительной](#) информацией.

```
\d table_name
```

Пример вывода

```
                                  Table "public.table_name"
 Column |       Type        | Collation | Nullable |                Default
--------+-------------------+-----------+----------+---------------------------------------
 id     | integer           |           | not null | nextval('table_name_id_seq'::regclass)
 name   | character varying |           |          |
 age    | integer           |           |          |
Indexes:
    "table_name_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "table_name_other_table_id_fkey" FOREIGN KEY (other_table_id) REFERENCES other_table(id)
```

Для получения только списка столбцов можно использовать [SQL](#) команду [\d](#) с [дополнительным](#) параметром.

```
\d table_name id,name
Did not find any relation named "id,name".
```

Для получения [списка](#) [индексов](#) можно использовать [SQL](#) команду [\di](#).

```
\di
```

Пример вывода

```
                    List of relations
 Schema |       Name       | Type  |  Owner   |    Table
--------+------------------+-------+----------+-------------
 public | table_name_pkey  | index | test_role | table_name
(1 row)
```

[SQL](#) команда [\di+](#) показывает то же самое, но с [дополнительной](#) информацией.

```
\di+
```

Пример вывода

```
                                    List of relations
 Schema |       Name       | Type  |  Owner   |    Table    | Persistence | Access method |  Size   | Description
--------+------------------+-------+----------+-------------+-------------+---------------+---------+-------------
 public | table_name_pkey  | index | test_role | table_name | permanent   | btree         | 8192 kB |
(1 row)
```

Шаг 4. Добавление новых данных в таблицу

Добавим данные в [таблицу](#) с помощью [SQL](#) команды [INSERT](#).

```
INSERT INTO table_name (name, age) VALUES ('Alice', 30);
INSERT INTO table_name (name, age) VALUES ('Bob', 25);
INSERT INTO table_name (name, age) VALUES ('Charlie', 35);
INSERT INTO table_name (name, age) VALUES ('Dave', 40);
INSERT INTO table_name (name, age) VALUES ('Eve', 45);
```

Для добавления большого количества данных можно использовать [SQL](#) команду [COPY](#).

```
COPY table_name (name, age) FROM '/path/to/file.csv' DELIMITER ',' CSV;
```

Для добавления данных из [файла](#) можно использовать [SQL](#) команду [\copy](#).

```
\copy table_name (name, age) FROM '/path/to/file.csv' DELIMITER ',' CSV;
```

Для добавления данных из [результата](#) [запроса](#) можно использовать [SQL](#) команду [INSERT INTO SELECT](#).

```
INSERT INTO table_name (name, age)
SELECT name, age FROM other_table WHERE age > 30;
```

Шаг 5. Чтение данных из таблицы

Для чтения данных из [таблицы](#) можно использовать [SQL](#) команду [SELECT](#). Для выбора всех столбцов можно использовать [*](#).

```
SELECT * FROM table_name;
```

Можно указать конкретные [столбцы](#), которые нужно выбрать.

```
SELECT name, age FROM table_name;
```

Можно использовать [WHERE](#) для [фильтрации](#) [строк](#).

```
SELECT * FROM table_name WHERE age > 30;
```

Использование WHERE

Можно использовать [операторы](#) сравнения, такие как [=](#), [<](#), [>](#), [<=](#), [>=](#), [<>](#).

```
SELECT * FROM table_name WHERE age = 30;
SELECT * FROM table_name WHERE age < 30;
SELECT * FROM table_name WHERE age > 30;
SELECT * FROM table_name WHERE age <= 30;
SELECT * FROM table_name WHERE age >= 30;
SELECT * FROM table_name WHERE age <> 30;
```

Можно использовать [AND](#), [OR](#) и [NOT](#) для [комбинирования](#) [условий](#).

```
SELECT * FROM table_name WHERE age > 30 AND name = 'John';
SELECT * FROM table_name WHERE age > 30 OR name = 'John';
SELECT * FROM table_name WHERE NOT age > 30;
```

Можно использовать [IN](#) для [проверки](#) [значения](#) в [списке](#) [значений](#).

```
SELECT * FROM table_name WHERE name IN ('John', 'Jane');
```

Можно использовать [BETWEEN](#) для [проверки](#) [значения](#) в [диапазоне](#) [значений](#).

```
SELECT * FROM table_name WHERE age BETWEEN 30 AND 40;
```

Можно использовать [LIKE](#) для [поиска](#) по [шаблону](#).

```
SELECT * FROM table_name WHERE name LIKE 'J%';
```

Сортировка результатов

Можно использовать [ORDER BY](#) для [сортировки](#) [результатов](#) [запроса](#).

```
SELECT * FROM table_name ORDER BY age;
```

Можно использовать [ASC](#) для [сортировки](#) по [возрастанию](#) (по [умолчанию](#)).

```
SELECT * FROM table_name ORDER BY age ASC;
```

Можно использовать [DESC](#) для [сортировки](#) по [убыванию](#).

```
SELECT * FROM table_name ORDER BY age DESC;
```

Можно использовать [несколько](#) [столбцов](#) для [сортировки](#).

```
SELECT * FROM table_name ORDER BY age, name;
```

Группировка результатов

Можно использовать [GROUP BY](#) для [группировки](#) [результатов](#) [запроса](#).

```
SELECT age, COUNT(*) FROM table_name GROUP BY age;
```

Можно использовать [несколько](#) [столбцов](#) для [группировки](#).

```
SELECT age, name, COUNT(*) FROM table_name GROUP BY age, name;
```

Можно использовать [HAVING](#) для [фильтрации](#) [результатов](#) [группировки](#).

```
SELECT age, COUNT(*) FROM table_name GROUP BY age HAVING COUNT(*) > 1;
```

Соединение таблиц

Создадим вторую [таблицу](#) с именем other_table. Для [столбца](#) id используем [тип данных](#) [SERIAL](#). Для [столбца](#) name используем тип данных [VARCHAR](#) с максимальной длиной 255 символов.

```
CREATE TABLE other_table (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255)
);
```

Заполним [таблицу](#) данными.

```
INSERT INTO other_table (name) VALUES ('John');
INSERT INTO other_table (name) VALUES ('Jane');
INSERT INTO other_table (name) VALUES ('Bob');
```

Можно использовать [INNER JOIN](#) для [соединения](#) [таблиц](#).

```
SELECT table_name.name, other_table.name
FROM table_name
INNER JOIN other_table ON table_name.id = other_table.id;
```

Пример вывода

```
 name | name
------+------
 John | John
 Jane | Jane
 Bob  | Bob
(3 rows)
```

Можно использовать [LEFT JOIN](#) для [соединения](#) [таблиц](#).

```
SELECT table_name.name, other_table.name
FROM table_name
LEFT JOIN other_table ON table_name.id = other_table.id;
```

Пример вывода

```
 name | name
------+------
 John | John
 Jane | Jane
 Bob  | Bob
(3 rows)
```

Можно использовать [RIGHT JOIN](#) для [соединения](#) [таблиц](#).

```
SELECT table_name.name, other_table.name
FROM table_name
RIGHT JOIN other_table ON table_name.id = other_table.id;
```

Пример вывода

```
 name | name
------+------
 John | John
 Jane | Jane
 Bob  | Bob
(3 rows)
```

Можно использовать [FULL JOIN](#) для [соединения](#) [таблиц](#).

```
SELECT table_name.name, other_table.name
FROM table_name
FULL JOIN other_table ON table_name.id = other_table.id;
```

Пример вывода

```
 name | name
------+------
 John | John
 Jane | Jane
 Bob  | Bob
(3 rows)
```

Комбинация SELECT + CREATE + INSERT

Можно [SELECT](#) [результаты](#) [запроса](#) и [создать](#) из [результата](#) [новую](#) [таблицу](#). Для этого используем [CREATE](#) + [INSERT](#).

```
CREATE TABLE new_table AS SELECT * FROM table_name WHERE age > 30;
```


Альтернатива

Для [создания](#) [новой](#) [таблицы](#) на [основе](#) [существующей](#) [таблицы](#) можно использовать [SELECT INTO](#).

```
SELECT * INTO new_table FROM table_name WHERE age > 30;
```


Пример вывода

```
SELECT 3
```

Создадим [новую](#) [таблицу](#) с [данными](#) [столбца](#) name [таблицы](#) table_name.

```
SELECT name INTO new_table FROM table_name;
```

Пример вывода

```
SELECT 5
```

Чтобы проверить созданную [новую](#) [таблицу](#), можно использовать [SQL](#) команду [\dt](#).

```
\dt
```

Пример вывода

```
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | new_table  | table | test_role
 public | table_name | table | test_role
(2 rows)
```

Можно [SELECT](#) [данные](#) из [новой](#) [таблицы](#).

```
SELECT * FROM new_table;
```

Пример вывода

```
 name
------
 John
 Jane
 Bob
 Alice
 Charlie
(5 rows)
```

Шаг 6. Обновление данных в таблице

Обновим данные в [таблице](#) с помощью [SQL](#) команды [UPDATE](#). Чтобы указать [условие](#), используем [WHERE](#).

```
UPDATE table_name SET age = 31 WHERE name = 'John';
```

Пример вывода

```
UPDATE 1
```

Можно [обновить](#) [несколько](#) [столбцов](#) [одновременно](#).

```
UPDATE table_name SET age = 31, name = 'Johnny' WHERE name = 'John';
```

Пример вывода

```
UPDATE 1
```

Можно [обновить](#) [данные](#) [столбца](#) [таблицы](#) [значениями](#) [данных](#) [столбца](#) [другой](#) [таблицы](#). Для этого используем [UPDATE](#) [FROM](#).

```
UPDATE table_name SET name = other_table.name
FROM other_table
WHERE table_name.id = other_table.id;
```

Пример вывода

```
UPDATE 3
```

Можно [обновить](#) [данные](#) [столбца](#) [таблицы](#) [значениями](#) [данных](#) [столбца](#) [той](#) [же](#) [таблицы](#). Для этого используем [UPDATE](#) [FROM](#).

```
UPDATE table_name SET name = other_table.name
FROM table_name AS other_table
WHERE table_name.id = other_table.id;
```

Пример вывода

```
UPDATE 3
```


Альтернатива

Для [обновления](#) [данных](#) [столбца](#) [таблицы](#) [значениями](#) [данных](#) [столбца](#) [другой](#) [таблицы](#) можно использовать [подзапрос](#).

```
UPDATE table_name SET name = (SELECT name FROM other_table WHERE table_name.id = other_table.id);
```


Пример вывода

```
UPDATE 3
```

Шаг 7. Удаление данных из таблицы

Удалим данные из [таблицы](#) с помощью [SQL](#) команды [DELETE](#). Чтобы указать [условие](#), используем [WHERE](#).

```
DELETE FROM table_name WHERE name = 'John';
```

Пример вывода

```
DELETE 1
```

Можно [удалить](#) [данные](#) [столбца](#) [таблицы](#) [значениями](#) [данных](#) [столбца](#) [другой](#) [таблицы](#). Для этого используем [DELETE](#) [USING](#).

```
DELETE FROM table_name
USING other_table
WHERE table_name.id = other_table.id;
```

Пример вывода

```
DELETE 3
```

Можно [удалить](#) [данные](#) [столбца](#) [таблицы](#) [значениями](#) [данных](#) [столбца](#) [той](#) [же](#) [таблицы](#). Для этого используем [DELETE](#) [USING](#).

```
DELETE FROM table_name AS t1
USING table_name AS t2
WHERE t1.id = t2.id;
```

Пример вывода

```
DELETE 3
```


Альтернатива

Для [удаления](#) [данных](#) [столбца](#) [таблицы](#) [значениями](#) [данных](#) [столбца](#) [другой](#) [таблицы](#) можно использовать [подзапрос](#).

```
DELETE FROM table_name WHERE id IN (SELECT id FROM other_table);
```


Пример вывода

```
DELETE 3
```

Шаг 8. Изменение структуры таблицы

Изменим [структуру](#) [таблицы](#) с помощью [SQL](#) команды [ALTER TABLE](#). Чтобы указать [изменение](#), используем [ADD COLUMN](#), [DROP COLUMN](#), [RENAME COLUMN](#) и [ALTER COLUMN](#).

Переименование таблицы

Чтобы [переименовать](#) [таблицу](#), можно использовать [SQL](#) команду [RENAME TO](#).

```
ALTER TABLE table_name RENAME TO new_table;
```

Пример вывода

```
ALTER TABLE
```

Чтобы [переименовать](#) [таблицу](#) [обратно](#), можно использовать [SQL](#) команду [RENAME TO](#).

```
ALTER TABLE new_table RENAME TO table_name;
```

Пример вывода

```
ALTER TABLE
```

Переименование столбца

Чтобы [переименовать](#) [столбец](#), можно использовать [SQL](#) команду [RENAME COLUMN](#).

```
ALTER TABLE table_name RENAME COLUMN name TO new_name;
```

Пример вывода

```
ALTER TABLE
```

Чтобы [переименовать](#) [столбец](#) [обратно](#), можно использовать [SQL](#) команду [RENAME COLUMN](#).

```
ALTER TABLE table_name RENAME COLUMN new_name TO name;
```

Пример вывода

```
ALTER TABLE
```

Удаление индекса

Чтобы [удалить](#) [индекс](#), можно использовать [SQL](#) команду [DROP INDEX](#). Чтобы [проверить](#) [наличие](#) [индекса](#), можно использовать [SQL](#) команду [\di](#).

```
\di
```

Пример вывода

```
                    List of relations
 Schema |       Name       | Type  |  Owner   |    Table
--------+------------------+-------+----------+-------------
 public | table_name_pkey  | index | test_role | table_name
(1 row)
```

Чтобы [удалить](#) [индекс](#), можно использовать [SQL](#) команду [DROP INDEX](#).

```
DROP INDEX table_name_pkey;
```

Пример вывода

```
DROP INDEX
```

Чтобы [проверить](#) [удаление](#) [индекса](#), можно использовать [SQL](#) команду [\di](#).

```
\di
```

Пример вывода

```
Did not find any relations.
```

Шаг 9. Удаление таблицы

Удалим [таблицу](#) с помощью [SQL](#) команды [DROP TABLE](#). Для [удаления](#) [нескольких](#) [таблиц](#) можно [перечислить](#) их через [запятую](#).

```
DROP TABLE table_name;
```

Пример вывода

```
DROP TABLE
```

Чтобы [проверить](#) [удаление](#) [таблицы](#), можно использовать [SQL](#) команду [\dt](#). Чтобы [вывести](#) [данные](#).

```
\dt
```

Пример вывода

```
Did not find any relations.
```

Использование псевдонимов таблиц и столбцов в запросах

Евгения Кулева

Содержание

Назад к содержанию

Псевдонимы таблиц

Псевдонимы столбцов

Вы можете указать временное имя для таблицы или столбца. Это имя называется **псевдонимом (alias)**. PostgreSQL использует псевдонимы во время выполнения запроса и не сохраняет их в базе данных или на диске.

Создадим таблицу для примеров:

```
CREATE TABLE employees (
employee_id serial PRIMARY KEY,
full_name VARCHAR NOT NULL,
hours INT,
rate INT,
manager_id INT
);
```

Добавим данные в таблицу:

```
INSERT INTO employees (employee_id, full_name, hours, rate, manager_id)
    VALUES
    (1, 'James Wilson', 8, 500, NULL),
    (2, 'Mary Burton', 6, 450, 1),
    (3, 'Patricia Robinson', 7, 430, 1),
    (4, 'Robert Gray', 8, 480, 1),
    (5, 'Elizabeth Tucker', 7, 470, 2),
    (6, 'Joseph Lewis', 7, 470, 2),
    (7, 'William Ferguson', 7, 470, 2),
    (8, 'Linda Black', 6, 320, 3),
    (9, 'David Green', 6, 310, 3),
    (10, 'Daniel Gray', 5, 390, 5),
    (11, 'Mark Armstrong', 6, 320, 4),
    (12, 'Donald Carter', 9, 290, 7),
    (13, 'Elizabeth Collins', 8, 300, 7),
    (14, 'Paul Brown', 8, 280, 8),
    (15, 'Andrew Clarke', 8, 280, 8);
```

Результат:

```
SELECT * FROM employees;

 employee_id |     full_name     | hours | rate | manager_id
-------------+-------------------+-------+------+------------
           1 | James Wilson      |     8 |  500 |
           2 | Mary Burton       |     6 |  450 |          1
           3 | Patricia Robinson |     7 |  430 |          1
           4 | Robert Gray       |     8 |  480 |          1
           5 | Elizabeth Tucker  |     7 |  470 |          2
           6 | Joseph Lewis      |     7 |  470 |          2
           7 | William Ferguson  |     7 |  470 |          2
           8 | Linda Black       |     6 |  320 |          3
           9 | David Green       |     6 |  310 |          3
          10 | Daniel Gray       |     5 |  390 |          5
          11 | Mark Armstrong    |     6 |  320 |          4
          12 | Donald Carter     |     9 |  290 |          7
          13 | Elizabeth Collins |     8 |  300 |          7
          14 | Paul Brown        |     8 |  280 |          8
          15 | Andrew Clarke     |     8 |  280 |          8
```

Псевдонимы таблиц

Вы можете назначить псевдоним таблице или ссылке на сложную таблицу и использовать этот псевдоним в текущем запросе. Ссылка на таблицу может быть производной таблицей, например подзапросом, выражением JOIN или сложной комбинацией этих конструкций.

Используйте следующий синтаксис для назначения псевдонима:

```
FROM <ссылка_на_таблицу> AS <псевдоним>
```

или

```
FROM <ссылка_на_таблицу> <псевдоним>
```

Где:

- `<ссылка_на_таблицу>` — имя таблицы или сложная ссылка на таблицу;
- `<псевдоним>` — уникальный идентификатор в рамках запроса.

Ключевое слово `AS` не является обязательным.

Псевдонимы часто используются для присвоения коротких идентификаторов длинным именам таблиц, чтобы улучшить читаемость запросов. Например:

```
SELECT * FROM very_long_table_name1 a JOIN very_long_table_name2 b ON a.id = b.num_id;
```

Псевдоним становится новым именем таблицы в текущем запросе. Если назначен псевдоним, исходное имя таблицы нельзя использовать в запросе. Например, следующий запрос выдаст ошибку:

```
-- некорректный запрос

SELECT * FROM table1 AS t WHERE table1.id > 276;
```

Хотя псевдонимы в основном используются для удобства, они необходимы, когда таблица соединяется сама с собой:

```
SELECT manager.employee_id, manager.full_name,
    subordinate.employee_id, subordinate.full_name, subordinate.manager_id
FROM employees AS manager
JOIN employees AS subordinate
ON manager.employee_id = subordinate.manager_id;
```

Результат:

```
employee_id |     full_name     | employee_id |     full_name     | manager_id
------------+-------------------+-------------+-------------------+------------
          1 | James Wilson      |           2 | Mary Burton       |          1
          1 | James Wilson      |           3 | Patricia Robinson |          1
          1 | James Wilson      |           4 | Robert Gray       |          1
          2 | Mary Burton       |           5 | Elizabeth Tucker  |          2
          2 | Mary Burton       |           6 | Joseph Lewis      |          2
          2 | Mary Burton       |           7 | William Ferguson  |          2
          3 | Patricia Robinson |           8 | Linda Black       |          3
          3 | Patricia Robinson |           9 | David Green       |          3
          5 | Elizabeth Tucker  |          10 | Daniel Gray       |          5
          4 | Robert Gray       |          11 | Mark Armstrong    |          4
          7 | William Ferguson  |          12 | Donald Carter     |          7
          7 | William Ferguson  |          13 | Elizabeth Collins |          7
          8 | Linda Black       |          14 | Paul Brown        |          8
          8 | Linda Black       |          15 | Andrew Clarke     |          8
```

Псевдоним также требуется, если ссылка на таблицу является подзапросом. За дополнительными сведениями обратитесь к разделу Подзапросы в выражениях FROM.

В случае неоднозначности определения псевдонимов можно использовать скобки. В следующем примере первое выражение назначает псевдоним `staff` второму экземпляру `employees`, а второе выражение назначает псевдоним результату `CROSS JOIN`:

```
SELECT * FROM employees AS managers CROSS JOIN employees AS staff ...
SELECT * FROM (employees AS managers CROSS JOIN employees) AS staff ...
```

Псевдонимы недопустимы вне круглых скобок. Например, следующий запрос корректен:

```
SELECT a.* FROM table1 AS a JOIN table2 AS b ON a.id = b.num_id;
```

Однако псевдоним `a` не виден за скобками в приведенном ниже запросе:

```
-- некорректный запрос

SELECT a.* FROM (table1 AS a JOIN table2 AS b ON a.id = b.num_id) AS c;
```

Псевдонимы столбцов

Псевдоним столбца позволяет присвоить временное имя столбцу или выражению в списке значений `SELECT`. Псевдоним столбца существует только во время выполнения запроса.

Псевдонимы обычно состоят из строчных букв. Если вам нужно использовать заглавные буквы, специальные символы, пробелы или ключевые слова, заключите псевдоним в двойные кавычки ("").

Используйте следующий синтаксис для назначения псевдонима:

```
SELECT <имя_столбца> AS <псевдоним> FROM <таблица>;
```

или

```
SELECT <имя_столбца> <псевдоним> FROM <таблица>;
```

Где:

- `<имя_столбца>` — имя столбца;
- `<псевдоним>` — уникальный идентификатор столбца;
- `<таблица>` — имя таблицы.

Ключевое слово `AS` не является обязательным.

Пример:

```
SELECT employee_id AS id, full_name AS "full name" FROM employees LIMIT 3;
```

Результат:

```
 id |     full name
----+-------------------
  1 | James Wilson
  2 | Mary Burton
  3 | Patricia Robinson
```

Обратите внимание, что псевдоним выводится в заголовке столбца.

Вы можете указать псевдоним для выражения:

```
SELECT full_name "full name", hours*rate sum FROM employees LIMIT 3;
```

Результат:

```
     full name      | sum
--------------------+------
 James Wilson       | 4000
 Mary Burton        | 2700
 Patricia Robinson | 3010
```

Вы также можете использовать альтернативный синтаксис для назначения псевдонимов столбцам и таблице одновременно:

```
FROM <ссылка_на_таблицу> [AS] <псевдоним> ( <псевдоним_столбца1> [, <псевдоним_столбца2> [, ...]] )
```

Где:

- `<ссылка_на_таблицу>` — имя таблицы или сложная ссылка на таблицу;
- `<псевдоним>` — псевдоним таблицы;
- `<псевдоним_столбца1..n>` — псевдонимы столбцов.

Если псевдонимов столбцов оказывается меньше, чем столбцов в таблице, остальные столбцы сохраняют свои исходные имена.

Пример:

```
SELECT * FROM employees AS managers ( id, "full name") LIMIT 3;
```

Результат:

```
 id |     full name     | hours | rate | manager_id
----+-------------------+-------+------+------------
  1 | James Wilson      |     8 |  500 |
  2 | Mary Burton       |     6 |  450 |          1
  3 | Patricia Robinson |     7 |  430 |          1
```

Оператор JOIN

© Антон Перкин

FAD.PG

Содержание

Назад к содержанию

- CROSS JOIN
- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- Условия JOIN
  - ON
  - USING
  - NATURAL
- Комбинирование операторов JOIN

`JOIN` - оператор данных из двух таблиц в соответствии с правилами задаваемого типа `JOIN`. В общем виде синтаксис `JOIN` можно записать следующим образом:

```
<таблица1> <тип_JOIN> <таблица2> [ <условие_join> ]
```

Оператор `JOIN` - обрабатывающая совпадения строк, имеет следующий синтаксис:

```
<таблица1> { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN <таблица2> ON <выражение_boolean>
<таблица1> { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN <таблица2> USING ( <список_столбцов_join> )
<таблица1> NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN <таблица2>
```

`INNER` - `OUTER` - ключевые слова необязательные слушатели во всех версиях. Значение по умолчанию - `INNER`. Однако, если определяется `LEFT` `RIGHT` или `FULL` - оно по `INNER` - не `OUTER` - это выполняет запрос как `OUTER JOIN`.

Условие `JOIN` определяет, какие строки двух исходных таблиц соответствуют друг другу. Выражение `ON` и `USING` задают условие `JOIN`. Кроме того, ключевое слово `NATURAL` можно установить для неявного `JOIN`.

Чтобы проиллюстрить, как работают типы `JOIN` добавляет с данными создадим две таблицы:

```
CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR (50),
    cover_id INT
);

INSERT INTO books (title, cover_id) VALUES
('Hyperion', 1),
('1984', 2),
('War and Peace', 3),
('The Time Machine', 3),
('Mrs. Dalloway', 1);

CREATE TABLE covers (
    id SERIAL PRIMARY KEY,
    cover CHAR (50)
);

INSERT INTO covers (cover) VALUES
('Paperback'),
('Hardcover'),
('Hardcover with dust jacket'),
('Leatherette cover');
```

Таблицы `books` содержит поле `cover_id`. Это поле является внешним ключом, который ссылается на первичный ключ в таблице `covers` и служит для определения связей между таблицами.

CROSS JOIN

Результирующая таблица `CROSS JOIN` содержит все возможные комбинации строк из двух таблиц (декартово произведение). Набор столбцов объединенной таблицы включает все столбцы из первой таблицы, за которыми следуют все столбцы второй таблицы. Если в таблицах есть `N` и `M` строк соответственно, то в объединенной таблице будет `N*M` строк. `CROSS JOIN` эквивалентен `FROM table1 INNER JOIN table2 ON TRUE`.

Пример:

```
SELECT * FROM books CROSS JOIN covers;
```

Результат:

```
 id |      title      | cover_id | id |            cover
----+-----------------+----------+----+------------------------------
  1 | Hyperion        |        1 |  1 | Paperback
  2 | 1984            |        2 |  1 | Paperback
  3 | War and Peace   |        3 |  1 | Paperback
  4 | The Time Machine|        3 |  1 | Paperback
  5 | Mrs. Dalloway   |        1 |  1 | Paperback
  1 | Hyperion        |        1 |  2 | Hardcover
  2 | 1984            |        2 |  2 | Hardcover
  3 | War and Peace   |        3 |  2 | Hardcover
  4 | The Time Machine|        3 |  2 | Hardcover
  5 | Mrs. Dalloway   |        1 |  2 | Hardcover
  1 | Hyperion        |        1 |  3 | Hardcover with dust jacket
  2 | 1984            |        2 |  3 | Hardcover with dust jacket
  3 | War and Peace   |        3 |  3 | Hardcover with dust jacket
  4 | The Time Machine|        3 |  3 | Hardcover with dust jacket
  5 | Mrs. Dalloway   |        1 |  3 | Hardcover with dust jacket
  1 | Hyperion        |        1 |  4 | Leatherette cover
  2 | 1984            |        2 |  4 | Leatherette cover
  3 | War and Peace   |        3 |  4 | Leatherette cover
  4 | The Time Machine|        3 |  4 | Leatherette cover
  5 | Mrs. Dalloway   |        1 |  4 | Leatherette cover
(20 rows)
```

INNER JOIN

`INNER JOIN` для сопоставляет каждую строку в первой таблице и проверяет, содержит ли вторая таблица строку, которая удовлетворяет условию `JOIN`. Если соответствующая строка найдена, `INNER JOIN` комбинирует столбцы из обеих таблиц, а добавляет эти в набор результатов.

Пример:

```
SELECT books.id, books.title, covers.cover FROM books
    INNER JOIN covers ON books.cover_id = covers.id;
```

Результат содержит поля `id` `title` - `books` для каждой книги. Кроме `Mrs. Dalloway` не включена в результат. так как для её значения `cover_id` - `5` - не найдено соответствующая строка в `covers`.

```
 id |      title      |            cover
----+-----------------+------------------------------
  1 | Hyperion        | Paperback
  2 | 1984            | Hardcover
  3 | War and Peace   | Hardcover with dust jacket
  4 | The Time Machine| Hardcover with dust jacket
  5 | Mrs. Dalloway   | Leatherette cover
```

LEFT OUTER JOIN

`LEFT OUTER JOIN` сначала выполняет операцию `INNER JOIN`. После этого каждая строка первой таблицы, которая исключена из результата `INNER JOIN`, добавляется в результат. `LEFT OUTER JOIN` - если значения в столбцах второй таблицы. В объединенной таблице всегда есть те строки которые есть для каждой строки первой таблицы.

Пример:

```
SELECT books.id, books.title, covers.cover FROM books
    LEFT OUTER JOIN covers ON books.cover_id = covers.id;
```

Результат содержит строку с книгой `Mrs. Dalloway`, в которой столбец `covers.cover` с плейсхолдером `NULL` имеет null значение:

```
 id |      title      |            cover
----+-----------------+------------------------------
  1 | Hyperion        | Paperback
  2 | 1984            | Hardcover
  3 | War and Peace   | Hardcover with dust jacket
  4 | The Time Machine| Hardcover with dust jacket
  5 | Mrs. Dalloway   |
```

RIGHT OUTER JOIN

`RIGHT OUTER JOIN` сначала выполняет операцию `INNER JOIN`. После этого каждая строка второй таблицы, которая исключена из результата `INNER JOIN`, добавляется в результат. `RIGHT OUTER JOIN` - если значения в столбцах первой таблицы. В объединенной таблице всегда есть одна строка для каждой строки второй таблицы.

Пример:

```
SELECT books.id, books.title, covers.cover FROM books
    RIGHT OUTER JOIN covers ON books.cover_id = covers.id;
```

Результат строки со значением `Leatherette cover` в которой столбцы `books.id` и `books.title` имеют null значения.

```
 id |      title      |            cover
----+-----------------+------------------------------
  1 | Hyperion        | Paperback
  2 | 1984            | Hardcover
  3 | War and Peace   | Hardcover with dust jacket
  4 | The Time Machine| Hardcover with dust jacket
    |                 | Leatherette cover
```

FULL OUTER JOIN

`FULL OUTER JOIN` сначала выполняет операцию `INNER JOIN`. После этого каждая строка первой таблицы, исключенная из результата `INNER JOIN` добавляется в результат. `FULL OUTER JOIN` - если значения в столбцах второй таблицы. Затем каждая строка второй таблицы, исключенная из результата `INNER JOIN`, добавляется в результат. `FULL OUTER JOIN` - если значения в столбцах первой таблицы.

Пример:

```
SELECT books.id, books.title, covers.cover FROM books
    FULL OUTER JOIN covers ON books.cover_id = covers.id;
```

```
 id |      title      |            cover
----+-----------------+------------------------------
  1 | Hyperion        | Paperback
  2 | 1984            | Hardcover
  3 | War and Peace   | Hardcover with dust jacket
  4 | The Time Machine| Hardcover with dust jacket
  5 | Mrs. Dalloway   |
    |                 | Leatherette cover
```

Условия JOIN

Условие `JOIN` определяет, какие строки из двух исходных таблиц совпадают. Условие может быть задано с помощью выражений `ON` `USING` или `NATURAL`.

ON

`ON` определяет значение boolean. Строка из первой таблицы соответствует строке из второй таблицы, если выражение `ON` возвращает `true`.

Создадим две таблицы для тестовых примеров:

```
CREATE TABLE table1(
    num INT PRIMARY KEY,
    name VARCHAR (20)
);

INSERT INTO table1 (num, name) VALUES
(1, 'a'),
(2, 'b'),
(3, 'c');

CREATE TABLE table2(
    num INT PRIMARY KEY,
    value VARCHAR (20)
);

INSERT INTO table2 (num, value) VALUES
(1, 'value1'),
(2, 'value2'),
(4, 'value3');
```

`table1`:

```
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

`table2`:

```
 num |  value
-----+---------
   1 | value1
   2 | value2
   4 | value3
```

Выполним следующий запрос с условием `ON`:

```
SELECT * FROM table1 INNER JOIN table2 ON table1.num = table2.num;
```

Результат:

```
 num | name | num |  value
-----+------+-----+---------
   1 | a    |   1 | value1
   2 | b    |   2 | value2
```

Условие `JOIN` также может содержать дополнительные выражения, которые не относятся непосредственно к `JOIN`.

Пример:

```
SELECT * FROM table1 INNER JOIN table2 ON table1.num = table2.num AND table2.value = 'value1';
```

Результат:

```
 num | name | num |  value
-----+------+-----+---------
   1 | a    |   1 | value1
```

USING

Вы можете использовать `USING` когда столбцы в обеих таблицах имеют одинаковое имена. `USING` принимает в качестве параметра список имен столбцов, разделенных запятыми, которые должны присутствовать в обеих входных таблицах. Например, `table1 JOIN table2 USING (column1, column2)`. Аналогично с помощью `ON`: `table1.column1 = table2.column1 AND table1.column2 = table2.column2`.

Выполним запрос с условием `USING` содержащий один столбец для каждой из перечисленных пар столбцов в указанном порядке, из которых следует есть совпадения в столбцах из первой и второй таблиц.

Пример:

```
SELECT * FROM table1 INNER JOIN table2 USING (num);
```

Результат:

```
 num | name | value
-----+------+--------
   1 | a    | value1
   2 | b    | value2
```

NATURAL

Выражение `NATURAL` задает оператор `USING`. Он включает все совпадающие имена столбцов, содержащихся в обеих входных таблицах. Как и в случае с `USING` результат выражения `NATURAL` содержит столбцы, перечисленные `NATURAL` содержит один столбец для каждой из перечисленных пар столбцов в указанном порядке, из которых следует все остающиеся столбцы из `table1` и `table2`.

Пример:

```
SELECT * FROM table1 NATURAL INNER JOIN table2;
```

Результат:

```
 num | name | value
-----+------+--------
   1 | a    | value1
   2 | b    | value2
```

Если в таблицах нет столбцов с совпадающими именами. `NATURAL JOIN` выдаст тот же результат, что и `CROSS JOIN`.

Комбинирование операторов JOIN

Операторы `JOIN` можно легко можно объединить их комбинировать один в другой. По умолчанию операторы `JOIN` выполняются слева направо. Можно заключить `JOIN` в круглые скобки, чтобы изменить порядок выполнения.

Создадим `table3` для тестовых примеров в дополнение к `table1` - `table2`:

```
CREATE TABLE table3(
    num INT PRIMARY KEY,
    name VARCHAR (20)
);

INSERT INTO table3 (id, value) VALUES
    (1, 'value1'),
    (2, 'value3');
```

`table1`:

```
 num | name
-----+------
   1 | a
   2 | b
   3 | c
```

`table2`:

```
 num |  value
-----+---------
   1 | value1
   2 | value2
   4 | value3
```

`table3`:

```
 id | value
----+--------
  1 | value1
  2 | value3
```

Выполним запрос `JOIN` с круглыми скобками и без них и сравним результаты.

Выполним запрос без круглых скобок:

```
SELECT * FROM table1 CROSS JOIN table2 INNER JOIN table3 USING(value);
```

Сначала PostgreSQL выполнит выражение `SELECT * FROM table1 CROSS JOIN table2`. Это результат:

```
 num | name | num |  value
-----+------+-----+---------
   1 | a    |   1 | value1
   1 | a    |   2 | value2
   1 | a    |   4 | value3
   2 | b    |   1 | value1
   2 | b    |   2 | value2
   2 | b    |   4 | value3
   3 | c    |   1 | value1
   3 | c    |   2 | value2
   3 | c    |   4 | value3
```

После этого PostgreSQL выполнит `INNER JOIN` для результирующей таблицы `CROSS JOIN` + `table3`. Поскольку `USING` содержит поле `value`, это поле будет первым столбцом результирующей таблицы.

Результат:

```
 value | num | name | num | id
-------+-----+------+-----+----
 value1|   1 | a    |   1 |  1
 value1|   2 | b    |   1 |  1
 value1|   3 | c    |   1 |  1
```

Добавим скобки в запрос:

```
SELECT * FROM table1 CROSS JOIN (table2 INNER JOIN table3 USING(value));
```

Сначала PostgreSQL выполнит выражение `SELECT * FROM table2 INNER JOIN table3 USING(value)`. Это результат:

```
 value | num | id
-------+-----+----
 value1|   1 |  1
```

После этого PostgreSQL выполнит `CROSS JOIN` для `table1` и результирующей таблицы `INNER JOIN`

Результат:

```
 num | name | value | num | id
-----+------+-------+-----+----
   1 | a    | value1|   1 |  1
   2 | b    | value1|   1 |  1
   3 | c    | value1|   1 |  1
```

Подзапросы

Содержание

Обзор
Скалярный подзапрос
Подзапросы с операторами сравнения
Подзапросы с EXISTS
Подзапросы с IN
Подзапросы с NOT IN
Подзапросы с ANY/SOME
Подзапросы с ALL
Сравнение одиночных строк
Подзапросы в выражениях FROM

Обзор

Подзапрос (subquery) — это SQL-запрос, вложенный в другой запрос. Подзапрос может быть в секции `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET` или в другом подзапросе. Подзапрос обычно добавляется в дополнительных скобках. Подзапрос может возвращать скалярное значение или `NULL`, может возвращать одну строку или несколько строк. Подзапрос может возвращать значения для `IN` и `NOT IN`, может использоваться с операторами сравнения `ANY` и `ALL`, может быть коррелированным или некоррелированным.

Например, таблица `books`:

```
id | title                | author_id | publication_year | genre     | evaluation | price | total_sales
---+----------------------+-----------+------------------+-----------+------------+-------+------------
 1 | Mrs. Bellamy         |         1 |             1997 | novel     |        4.2 |  9.99 |    1523000
 2 | To the Lighthouse    |         2 |             1927 | novel     |        4.1 |  8.99 |     987000
 3 | To Kill A Mockingbird|         3 |             1960 | novel     |        4.8 |  7.99 |    2145000
 4 | The Great Gatsby     |         4 |             1925 | novel     |        4.0 |  6.99 |    1345000
 5 | The Lord of the Rings|         5 |             1954 | fantasy   |        4.9 | 12.99 |    3210000
 6 | The Hobbit, or There and Back Again | 5 | 1937 | fantasy   |        4.7 | 10.99 |    2567000
 7 | Harry Potter         |         6 |             1997 | fantasy   |        4.8 | 14.99 |    4123000
 8 | Hyperion             |         7 |             1989 | sci-fi    |        4.6 | 11.99 |     876000
 9 | The Time Machine     |         8 |             1895 | sci-fi    |        4.3 |  7.99 |     654000
```

```
SELECT id, title, genre
FROM books
WHERE genre = (SELECT genre FROM books WHERE id = 1);
```

Результат:

```
id | title                | genre
---+----------------------+-------
 1 | Mrs. Bellamy         | novel
 2 | To the Lighthouse    | novel
 3 | To Kill A Mockingbird| novel
 4 | The Great Gatsby     | novel
```

Подзапрос выполняется перед внешним запросом, и результат подзапроса передается родительскому запросу. В этом примере подзапрос возвращает `'novel'`, и внешний запрос использует это значение для фильтрации. Подзапрос не должен быть коррелированным.

Скалярный подзапрос

Скалярный подзапрос — это подзапрос, который возвращает только одно значение: одну строку и один столбец. Такое значение называется скалярным значением.

Скалярный подзапрос можно использовать в выражении: `SELECT MAX(total_sales) FROM books WHERE customer_id = 1?` максимальное значение поля `total_sales` для `customer_id = 1`.

```
INSERT INTO order_statistic (customer_id, max_sum) VALUES (
  1,
  (SELECT MAX(total) FROM orders WHERE customer_id = 1)
)
RETURNING *;
```

Результат:

```
customer_id | max_sum
------------+---------
          1 | 1000.00
```

Подзапросы с операторами сравнения

Можно использовать подзапросы с операторами сравнения. Этот тип подзапроса должен возвращать не более одного значения. Значение может быть константой или агрегатной функцией. Подзапрос должен возвращать ноль или одно значение. Если возвращается несколько значений, то другой тип оператора сравнения.

Оператор	Операция
`=`	Равно
`<`	Больше
`>`	Меньше
`<=`	Больше или равно
`>=`	Меньше или равно
`<>`	Не равно
`!=`	Не равно

Нулевое значение (`NULL`) не сравнивается ни с каким значением. Этот тип подзапроса должен возвращать не более одного значения. Значение может быть константой или агрегатной функцией. Подзапрос должен возвращать ноль или одно значение. `NULL` — это специальное значение, которое не сравнивается ни с каким значением, даже с `NULL`, поэтому для этого используется оператор `IS NULL`.

Например, операторы могут сравнивать числа с целым или дробным числом.

```
SELECT id, title, price
FROM books
WHERE price >
  (SELECT price FROM books);
```

Результат:

```
id | title                | price
---+----------------------+-------
 1 | The Great Gatsby     |   900
 2 | The Lord of the Rings|  1200
 3 | The Hobbit, or There and Back Again | 1100
 4 | War and Peace        |  1500
```

Подзапросы с EXISTS

```
EXISTS (<подзапрос>)
```

Аргумент `EXISTS` может быть подзапросом или литералом. `EXISTS` возвращает `TRUE`, если подзапрос возвращает хотя бы одну строку. Если же подзапрос не возвращает ни одной строки, `EXISTS` возвращает `FALSE`. Если подзапрос не возвращает строки, результат `EXISTS` будет `FALSE`.

Подзапрос может состоять из скалярных или агрегатных функций.

Например, таблица `books` содержит информацию о книгах и авторах книг. Получим данные авторов, которые есть в нашей небольшой базе книг. Получим информацию об авторах из таблицы `authors`, если в таблице `books` есть книги этих авторов (`EXISTS` и `WHERE`).

```
SELECT id, title, author_id
FROM books
WHERE EXISTS (SELECT 1 FROM authors WHERE id = a.author_id);
```

Результат:

```
id | title                | author_id
---+----------------------+----------
 1 | The Great Gatsby     |         1
 2 | 1Q84                 |         2
 3 | The Hobbit, or There and Back Again |  3
 4 | The Lord of the Rings|         4
 5 | War and Peace        |         5
 6 | To Kill A Mockingbird|         6
 7 | Hyperion             |         7
 8 | The Time Machine     |         8
```

Подзапросы с IN

Синтаксис выражения `IN`:

```
<где_выражение> IN (<где_подзапрос>)
```

`<где_выражение>` должно иметь тот же тип данных, что `<где_подзапрос>`. `<где_подзапрос>` выполняется и сравнивается с каждой строкой результата подзапроса. Результат `IN` может быть `TRUE`, `FALSE` или `NULL`. Если `<где_выражение>` равно `NULL`, или если ни одна строка равной строки из `<где_подзапроса>` не возвращает ни одна строка, то результат запроса отобразит всю внешнюю запрос.

```
SELECT id, title FROM books
WHERE author_id IN
  (SELECT id FROM authors WHERE country='USA');
```

Результат:

```
 id |        title
----+---------------------
  3 | To Kill A Mockingbird
  4 | The Great Gatsby
  7 | Hyperion
```

Обратите внимание, что если `<где_выражение>` возвращает значение `NULL`, или же возвращены значений подзапроса и ни одной равно одна строка подзапроса `NULL`, результат `IN` может быть `NULL`, а не `FALSE`.

Также часть выражения `IN` может быть конструктором списка.

```
<конструктор_строки> IN (<где_подзапрос>)
```

Пример:

```
SELECT DISTINCT books.id, books.title FROM books
WHERE author_id IN (
  SELECT id FROM authors WHERE country='USA');
```

Результат:

```
 id |        title
----+---------------------
  3 | To Kill A Mockingbird
  4 | The Great Gatsby
  7 | Hyperion
```

`IN` эквивалентно `= ANY`.

Подзапросы с NOT IN

Синтаксис выражения `NOT IN`:

```
<not_in_выражение> NOT IN (<not_in_подзапрос>)
```

`<not_in_выражение>` должно иметь тот же тип данных, что `<not_in_подзапрос>`. `<not_in_подзапрос>` выполняется и сравнивается с каждой строкой результата подзапроса. Результат `NOT IN` может быть `TRUE`, `FALSE` или `NULL`. Если `<not_in_выражение>` равно `NULL`, или если ни одна строка равной строки из `<not_in_подзапроса>` не возвращает ни одна строка, результат `NULL`. Если `<not_in_выражение>` не возвращает строк, результат `TRUE`. Если найдена только одна строка подзапроса, результат `NULL`, а не `FALSE`.

Также часть выражения `NOT IN` может быть конструктором списка.

```
<конструктор_строки> NOT IN (<not_in_подзапрос>)
```

Пример:

```
SELECT id, title FROM books
WHERE author_id NOT IN
  (SELECT id FROM authors WHERE country='USA');
```

Результат:

```
 id |        title
----+---------------------
  1 | Mrs. Bellamy
  2 | To the Lighthouse
  5 | The Lord of the Rings
  6 | The Hobbit, or There and Back Again
  8 | War and Peace
  9 | The Time Machine
```

`NOT IN` эквивалентно `!= ALL`.

Если часть выражения `NOT IN` может быть конструктором строки:

```
<конструктор_строки> NOT IN (<not_in_подзапрос>)
```

`<not_in_подзапрос>` должен возвращать такое же количество столбцов, которое содержится в выражении `<конструктор_строки>`.

Пример:

```
SELECT DISTINCT books.id, books.title FROM books
WHERE author_id NOT IN
  (SELECT id FROM authors WHERE country='USA');
```

Результат:

```
 id |        title
----+---------------------
  1 | Mrs. Bellamy
  2 | To the Lighthouse
  5 | The Lord of the Rings
  6 | The Hobbit, or There and Back Again
  8 | War and Peace
  9 | The Time Machine
```

Подзапросы с ANY/SOME

`SOME` — это синоним `ANY`. Синтаксис выражения `ANY/SOME`:

```
<выражение> <оператор> ANY (<подзапрос>)
<выражение> <оператор> SOME (<подзапрос>)
```

`<выражение>` должно иметь тот же тип данных, что `<подзапрос>`. `<выражение>` должно соответствовать каждой строкой результата подзапроса с использованием `<оператора>`. Результат `ANY` может быть `TRUE`, `FALSE` или `NULL`. Если `<выражение>` равно `NULL`, или если ни одна строка равной строки из `<подзапроса>` не возвращает ни одна строка, результат `NULL`. Если `<выражение>` не возвращает строк, результат `FALSE`. Если найдена только одна строка подзапроса, результат `NULL`, а не `FALSE`.

`<оператор>` может быть любым из операторов сравнения.

Часть выражения `ANY/SOME` может быть конструктором строки:

```
<конструктор_строки> <оператор> ANY (<подзапрос>)
<конструктор_строки> <оператор> SOME (<подзапрос>)
```

`<подзапрос>` должен возвращать такое же количество столбцов, которое содержится в выражении `<конструктор_строки>`.

Пример:

```
SELECT DISTINCT books.id, books.title FROM books
WHERE author_id = ANY
  (SELECT id FROM authors WHERE country='USA');
```

Результат (одно):

```
 id |        title
----+---------------------
  3 | To Kill A Mockingbird
  4 | The Great Gatsby
  7 | Hyperion
```

Часть выражения `ANY/SOME` может быть конструктором строки:

```
<конструктор_строки> <оператор> ANY (<подзапрос>)
<конструктор_строки> <оператор> SOME (<подзапрос>)
```

`<подзапрос>` должен возвращать такое же количество столбцов, которое содержится в выражении `<конструктор_строки>`.

Пример:

```
SELECT DISTINCT books.id, books.title FROM books
WHERE author_id = ANY
  (SELECT id FROM authors WHERE country='USA');
```

Результат:

```
 id |        title
----+---------------------
  3 | To Kill A Mockingbird
  4 | The Great Gatsby
  7 | Hyperion
```

Подзапросы с ALL

Синтаксис выражения `ALL`:

```
<all_выражение> <оператор> ALL (<all_подзапрос>)
```

`<all_выражение>` должно иметь тот же тип данных, что `<all_подзапрос>`. `<all_выражение>` должно соответствовать каждой строкой результата подзапроса с использованием `<оператора>`. Результат `ALL` может быть `TRUE`, `FALSE` или `NULL`. Если `<all_выражение>` равно `NULL`, или если ни одна строка равной строки из `<all_подзапроса>` не возвращает ни одна строка, результат `NULL`. Если `<all_выражение>` не возвращает строк, результат `TRUE`. Если найдена только одна строка подзапроса, результат `NULL`, а не `TRUE`. Если же подзапрос возвращает строки, результат `FALSE`, и есть еще строки подзапроса `NULL`.

`<оператор>` может быть любым из операторов сравнения.

Часть выражения `ALL` может быть конструктором строки:

```
<конструктор_строки> <оператор> ALL (<all_подзапрос>)
```

`<all_подзапрос>` должен возвращать такое же количество столбцов, которое содержится в выражении `<конструктор_строки>`.

Пример:

```
SELECT DISTINCT books.id, books.title FROM books
WHERE author_id != ALL
  (SELECT id FROM authors WHERE country='USA');
```

Результат:

```
 id |        title
----+---------------------
  1 | Mrs. Bellamy
  2 | To the Lighthouse
  5 | The Lord of the Rings
  6 | The Hobbit, or There and Back Again
  8 | War and Peace
  9 | The Time Machine
```

Сравнение одиночных строк

```
<конструктор_строки> <оператор> (<подзапрос>)
```

`<конструктор_строки>` должен возвращать такое же количество столбцов, которое содержится в выражении `<конструктор_строки>`. `<подзапрос>` должен возвращать ноль или одну строку. Если он не возвращает строк, результат `NULL`. Можно использовать следующие операторы сравнения: `=`, `<`, `<=`, `>`, `>=`, или `<>`. `<конструктор_строки>` должен включать в сравнение все строки. Следующий пример получает все книги, написанные теми авторами, которые родились в Англии.

```
SELECT book_id, book_title, percentage
FROM
  (SELECT id AS book_id, title AS book_title, public_user AS user, ROUND(total_sales/(SELECT SUM(total_sales) FROM books) * 100, 2) AS percentage FROM books) AS
t
WHERE user = 1000;
```

Результат:

```
book_id |        book_title        | percentage
--------+--------------------------+------------
      1 | Mrs. Bellamy             |       5.35
      2 | To the Lighthouse        |       3.47
      3 | To Kill A Mockingbird    |       9.56
      4 | The Great Gatsby         |       6.00
      5 | The Lord of the Rings    |      14.31
      6 | The Hobbit, or There and Back Again |  11.45
      7 | 1Q84                     |       7.31
      8 | The Hobbit, or There and Back Again |  11.45
      9 | Hyperion                 |       5.23
```

Подзапросы в выражениях FROM

SQL выражение с подзапросом в `FROM` называется скалярным.

```
SELECT <выражение> FROM (<подзапрос>) AS <псевдоним>
```

Каждый скалярный подзапрос `FROM` должен иметь псевдоним. Если столбцы в списке `SELECT` подзапроса должны иметь уникальные имена.

Следующий пример получает книги издательства после 1980 года или продаж от общего объема продаж:

```
SELECT book_id, book_title, percentage
FROM
  (SELECT id AS book_id, title AS book_title, public_user AS user, ROUND(total_sales/(SELECT SUM(total_sales) FROM books) * 100, 2) AS percentage FROM books) AS
t
WHERE user = 1000;
```

Результат:

```
book_id | book_title | percentage
--------+------------+------------
```

Комбинирование запросов (UNION, INTERSECT, EXCEPT)

Д. Евгения Кузина

AD.PG

Содержание

Назад к содержанию

Пример UNION

Пример INTERSECT

Пример EXCEPT

Сочетание операторов множеств

Вы можете использовать операторы множеств `UNION`, `INTERSECT` и `EXCEPT` для объединения результатов двух запросов. Эти операторы имеют следующий синтаксис:

```
<запрос1> UNION [ALL] <запрос2>
<запрос1> INTERSECT [ALL] <запрос2>
<запрос1> EXCEPT [ALL] <запрос2>
```

Где:

- `<запрос1>` и `<запрос2>` – запросы для объединения. Запросы должны быть совместимыми для объединения, т.е. возвращать одинаковое количество столбцов. Соответствующие столбцы должны иметь совместимые типы данных, как описано в следующей статье: UNION.
- `UNION` добавляет результат второго запроса к результату первого запроса. Порядок, в котором `UNION` возвращает строки, не определен.
- `INTERSECT` возвращает все строки, которые есть как в результате первого запроса, так и в результате второго запроса.
- `EXCEPT` возвращает все строки, которые включены в результат первого запроса, но не входят в результат второго запроса.
- `ALL` указывает, что в результат должны быть включены повторяющиеся строки. Если запрос не содержит модификатора `ALL`, операторы `UNION`, `INTERSECT` и `EXCEPT` исключают повторяющиеся строки из результата.

Чтобы посмотреть, как работают операторы множеств, создадим три таблицы:

```
CREATE TABLE book_store1(
    id INT PRIMARY KEY,
    title VARCHAR (100)
);

CREATE TABLE book_store2(
    id INT PRIMARY KEY,
    title VARCHAR (100)
);

CREATE TABLE book_store3(
    id INT PRIMARY KEY,
    title VARCHAR (100)
);

INSERT INTO book_store1 (id, title) VALUES
(1, 'Hyperion'),
(2, '1984'),
(3, 'War and Peace'),
(4, 'The Time Machine');

INSERT INTO book_store2 (id, title) VALUES
(3, 'War and Peace'),
(4, 'The Time Machine'),
(5, 'Mrs. Dalloway' );

INSERT INTO book_store3 (id, title) VALUES
(1, 'Hyperion'),
(2, '1984'),
(5, 'Mrs. Dalloway' );
```

Пример UNION

Следующий запрос использует `UNION` для объединения таблиц `book_store1` и `book_store2`:

```
SELECT * from book_store1 UNION SELECT * from book_store2;
```

Результат:

```
id |       title
---+-------------------
5 | Mrs. Dalloway
4 | The Time Machine
2 | 1984
3 | War and Peace
1 | Hyperion
```

Добавим модификатор `ALL` и сравним результаты:

```
SELECT * from book_store1 UNION ALL SELECT * from book_store2;
```

Результат содержит повторяющиеся строки:

```
 id |       title
----+-------------------
  1 | Hyperion
  2 | 1984
  3 | War and Peace
  4 | The Time Machine
  3 | War and Peace
  4 | The Time Machine
  5 | Mrs. Dalloway
```

Пример INTERSECT

Следующий запрос использует `INTERSECT` для возврата всех строк, которые есть в обеих таблицах: и в `book_store1`, и в `book_store2`:

```
SELECT * from book_store1 INTERSECT SELECT * from book_store2;
```

Результат:

```
 id |       title
----+-------------------
  4 | The Time Machine
  3 | War and Peace
```

Пример EXCEPT

Запрос ниже использует `EXCEPT`, чтобы вернуть все строки, которые есть в таблице `book_store1`, но отсутствуют в `book_store2`:

```
SELECT * from book_store1 EXCEPT SELECT * from book_store2;
```

Результат:

```
 id |  title
----+----------
  2 | 1984
  1 | Hyperion
```

Сочетание операторов множеств

Вы можете комбинировать операторы множеств в одном запросе.

Например, следующий запрос сначала объединяет таблицы `book_store1` и `book_store2`, а затем исключает из результата строки таблицы `book_store3`:

```
SELECT * from book_store1 UNION SELECT * from book_store2
    EXCEPT SELECT * from book_store3;
```

Результат:

```
 id |       title
----+-------------------
  4 | The Time Machine
  3 | War and Peace
```

Круглые скобки определяют порядок применения операторов. Без круглых скобок `UNION` и `EXCEPT` выполняются слева направо.

Следующие выражения эквивалентны:

```
<запрос1> UNION <запрос2> EXCEPT <запрос3>

(<запрос1> UNION <запрос2>) EXCEPT <запрос3>
```

`INTERSECT` имеет более высокий приоритет, чем `UNION` и `EXCEPT`. Следующие выражения эквивалентны:

```
<запрос1> UNION <запрос2> INTERSECT <запрос3>

<запрос1> UNION (<запрос2> INTERSECT <запрос3>)
```

Выполним следующий запрос с `INTERSECT`:

```
SELECT * from book_store1 UNION SELECT * from book_store2
    INTERSECT SELECT * from book_store3;
```

PostgreSQL сначала выполнит `INTERSECT`, его результат:

```
 id |    title
----+-------------
  5 | Mrs. Dalloway
```

После этого выполняется `UNION`, который объединяет результат, приведенный выше, со строками `book_store1`.

Результат:

```
 id |       title
----+-------------------
  5 | Mrs. Dalloway
  4 | The Time Machine
  2 | 1984
  3 | War and Peace
  1 | Hyperion
```

Добавим скобки в запрос:

```
(SELECT * from book_store1 UNION SELECT * from book_store2)
    INTERSECT SELECT * from book_store3;
```

PostgreSQL сначала выполнит `UNION`, его результат:

```
 id |       title
----+-------------------
  5 | Mrs. Dalloway
  4 | The Time Machine
  2 | 1984
  3 | War and Peace
  1 | Hyperion
```

После этого `INTERSECT` возвращает все строки, которые есть как в результате выполнения `UNION`, так и в таблице `book_store3`.

Результат:

```
 id |    title
----+-------------
  2 | 1984
  1 | Hyperion
  5 | Mrs. Dalloway
```

Чтобы использовать дополнительные выражения во входных запросах, например `LIMIT`, заключите запросы в круглые скобки. Если добавить дополнительное выражение в запрос без круглых скобок, может возникнуть синтаксическая ошибка или выражение применится к выходным данным оператора множеств, а не к отдельному запросу. Например, следующий запрос выполнится без ошибок, но PostgreSQL применит `LIMIT 10` к результату `UNION`:

```
SELECT a FROM b UNION SELECT x FROM y LIMIT 10;
```

Этот запрос эквивалентен следующему выражению:

```
(SELECT a FROM b UNION SELECT x FROM y) LIMIT 10;
```

Чтобы применить `LIMIT` ко второму запросу, заключите этот запрос в круглые скобки:

```
SELECT a FROM b UNION (SELECT x FROM y LIMIT 10);
```

Общие табличные выражения (CTE)

© Артем Ермаков

Содержание

Назад к содержанию

Оператор SELECT в CTE
Рекурсивные запросы
- Порядок поиска
- Выявление бесконечных циклов

Материализация CTE
Изменение данных

Общие табличные выражения (CTE) – это временный результат выполнения SQL-выражения, который можно использовать в других SQL-выражениях. CTE позволяют упрощать сложные SQL-запросы, разбивая их на составные части.

CTE имеют следующий синтаксис:

```
WITH <cte_name> [<column_list>] AS (
  <cte_query_definition>
)
<statement>;
```

В приведенном выше выражении `WITH` создает CTE, используя следующие параметры:

- `<CTE_name>` - идентификатор CTE;
- `<column_list>` - необязательный параметр, содержащий список столбцов CTE;
- `<CTE_query_definition>` - SQL-выражение, создающее CTE;
- `<statement>` - SQL-выражение, использующее CTE.

Вы также можете создать несколько CTE с помощью одного `WITH`.

Оператор SELECT в CTE

Приведенный ниже пример показывает общее количество продаж по книге в магазине с самыми высокими продажами. Оператор `WITH` определяет два вспомогательных подзапроса с именами `store_sales` и `top_stores`. Выходные данные `store_sales` используются в `top_stores`, а выходные данные `top_stores` – в главном запросе `SELECT`.

```
WITH store_sales AS (
  SELECT store, SUM(amount) AS total_sales
  FROM orders
  GROUP BY store
), top_stores AS (
  SELECT store
  FROM store_sales
  WHERE total_sales > (SELECT SUM(total_sales)/10 FROM store_sales)
)
SELECT store,
       book,
       SUM(quantity) AS book_quantity,
       SUM(amount) AS book_sales
FROM orders
WHERE store IN (SELECT store FROM top_stores)
GROUP BY store, book;
```

Если перелистать этот запрос без `WITH`, получится сложный запрос с двумя уровнями вложенных подзапросов `SELECT`.

Рекурсивные запросы

Рекурсивные запросы используются для обработки иерархических данных. Чтобы создать рекурсивный запрос, добавьте модификатор `RECURSIVE` в выражение `WITH`. Это позволит ссылаться на собственный вывод запроса.

Рекурсивное CTE имеет следующий синтаксис:

```
WITH RECURSIVE <CTE_name> AS(
    <CTE_query_definition>  -- инициализирующая часть
    UNION [ALL]
    <CTE_query_definition>  -- рекурсивная часть
)
```

Рекурсивные запросы содержат нерекурсивную часть, которая `UNION` или `UNION ALL` с рекурсивную часть. Только рекурсивная часть может содержать ссылку на собственный вывод запроса.

PostgreSQL выполняет рекурсивное CTE в следующей последовательности:

1. Выполняет нерекурсивную часть, чтобы создать первый набор результатов (R0).
2. Выполняет рекурсивную часть с R0 в качестве входных данных, чтобы вернуть результирующий набор R1 в качестве входных данных.
3. Повторяет шаг 2, пока не будет возвращен пустой результат.
4. Возвращает окончательный результат, который является итогом выполнения `UNION` или `UNION ALL` над наборами результатов R0, R1...Rn.

Создадим новую таблицу, чтобы продемонстрировать рекурсивный запрос PostgreSQL:

```
CREATE TABLE employees (
employee_id serial PRIMARY KEY,
full_name VARCHAR NOT NULL,
manager_id INT
);
```

Добавим данные в таблицу:

```
INSERT INTO employees (employee_id, full_name, manager_id)
    VALUES
    (1, 'James Wilson', NULL),
    (2, 'Mary Burton', 1),
    (3, 'Patricia Robinson', 1),
    (4, 'Robert Gray', 1),
    (5, 'Elizabeth Tucker', 2),
    (6, 'Joseph Lewis', 2),
    (7, 'William Ferguson', 2),
    (8, 'Linda Black', 3),
    (9, 'David Green', 3),
    (10, 'Daniel Gray', 3),
    (11, 'Mark Armstrong', 4),
    (12, 'Donald Carter', 4),
    (13, 'Elizabeth Collins', 7),
    (14, 'Paul Brown', 9),
    (15, 'Andrew Clarke', 9);
```

Следующий рекурсивный запрос возвращает сотрудника со значением `employee_id` равным 2 и всех его подчиненных.

```
WITH RECURSIVE subordinates (employee_id, full_name, manager_id) AS (
   SELECT employee_id, manager_id, full_name
   FROM employees WHERE employee_id = 2
   UNION
      SELECT e.employee_id, e.manager_id, e.full_name
      FROM employees e
   INNER JOIN subordinates s ON s.employee_id = e.manager_id
)
SELECT * FROM subordinates;
```

Нерекурсивная часть запроса возвращает базовый набор результатов (R0):

```
 employee_id | manager_id | full_name
-------------+------------+-------------
           2 |          1 | Mary Burton
```

Рекурсивная часть возвращает подчиненных сотрудника из первого результата. Это итог выполнения `JOIN` для таблицы `employees` и CTE `subordinates`. Первая итерация рекурсивного подзапроса возвращает следующие данные:

```
 employee_id | manager_id |     full_name
-------------+------------+--------------------
           5 |          2 | Elizabeth Tucker
           6 |          2 | Joseph Lewis
           7 |          2 | William Ferguson
```

Вторая итерация использует результаты первой итерации в качестве входных значений и вернет следующий результат:

```
 employee_id | manager_id |     full_name
-------------+------------+--------------------
          12 |          4 | Donald Carter
          13 |          7 | Elizabeth Collins
          10 |          3 | Daniel Gray
```

Третья итерация вернет пустой результат, потому что у сотрудников с `employee_id` 10, 12 и 13 нет подчиненных.

Конечный результат представляет собой объединение всех результатов первой и второй итерации, отмеченных нерекурсивными и рекурсивными частями:

```
 employee_id | manager_id |     full_name
-------------+------------+--------------------
           2 |          1 | Mary Burton
           5 |          2 | Elizabeth Tucker
           6 |          2 | Joseph Lewis
           7 |          2 | William Ferguson
          12 |          4 | Donald Carter
          13 |          7 | Elizabeth Collins
          10 |          3 | Daniel Gray
```

Порядок поиска

Если нужно применить к результатам запроса сортировку "в глубину" (depth-first order) или "в ширину" (breadth-first order), используйте операторы `SEARCH`.

Добавьте фразу `SEARCH DEPTH FIRST BY` чтобы результаты с сортировкой "в глубину". При этом в вашу запрос будет добавлен столбец, содержащий путь до записи:

```
WITH RECURSIVE subordinates(employee_id, manager_id, full_name) AS (
   SELECT employee_id, manager_id, full_name
   FROM employees WHERE employee_id = 2
   UNION
      SELECT e.employee_id, e.manager_id, e.full_name
      FROM employees e
   INNER JOIN subordinates s ON s.employee_id = e.manager_id
) SEARCH DEPTH FIRST BY employee_id SET ordercol
SELECT * FROM subordinates ORDER BY ordercol;
```

Результат:

```
 employee_id | manager_id |     full_name     |   ordercol
-------------+------------+-------------------+---------------
           2 |          1 | Mary Burton       | {(2)}
           5 |          2 | Elizabeth Tucker  | {(2),(5)}
          10 |          3 | Daniel Gray       | {(2),(5),(10)}
           6 |          2 | Joseph Lewis      | {(2),(6)}
           7 |          2 | William Ferguson  | {(2),(7)}
          12 |          4 | Donald Carter     | {(2),(7),(12)}
          13 |          7 | Elizabeth Collins | {(2),(7),(13)}
```

Используйте `SEARCH BREADTH FIRST BY`, чтобы показать результаты с сортировкой "в ширину". При этом в вашу запрос будет добавлен столбец с уровнем вложенности рекурсивного запроса.

```
WITH RECURSIVE subordinates(employee_id, manager_id, full_name) AS (
   SELECT employee_id, manager_id, full_name
   FROM employees WHERE employee_id = 2
   UNION
      SELECT e.employee_id, e.manager_id, e.full_name
      FROM employees e
   INNER JOIN subordinates s ON s.employee_id = e.manager_id
) SEARCH BREADTH FIRST BY employee_id SET ordercol
SELECT * FROM subordinates ORDER BY ordercol;
```

Результат:

```
 employee_id | manager_id |     full_name     | ordercol
-------------+------------+-------------------+----------
           2 |          1 | Mary Burton       | {0,2}
           5 |          2 | Elizabeth Tucker  | {1,5}
           6 |          2 | Joseph Lewis      | {1,6}
           7 |          2 | William Ferguson  | {1,7}
          10 |          3 | Daniel Gray       | {2,10}
          12 |          4 | Donald Carter     | {2,12}
          13 |          7 | Elizabeth Collins | {2,13}
```

Выявление бесконечных циклов

При работе с рекурсивными запросами, которые могут привести к бесконечному циклу, используйте `CYCLE` для обнаружения бесконечных циклов.

```
WITH RECURSIVE subordinates(employee_id, manager_id, full_name) AS (
   SELECT employee_id, manager_id, full_name
   FROM employees WHERE employee_id=2
   UNION
      SELECT e.employee_id, e.manager_id, e.full_name
      FROM employees e
   INNER JOIN subordinates s ON s.employee_id = e.manager_id
) CYCLE employee_id SET is_cycle USING path
SELECT * FROM subordinates;
```

Выражение `CYCLE` содержит список столбцов, которые нужно отслеживать для обнаружения бесконечных циклов (`employee_id` в нашем примере) и дополнительно обнаружен ли цикл (`is_cycle`), а также имя столбца, который отслеживает путь (`path`). Столбцы `is_cycle` и `path` добавляются в выходные данные запроса.

Результат:

```
 employee_id | manager_id |     full_name     | is_cycle |       path
-------------+------------+-------------------+----------+-------------------
           2 |          1 | Mary Burton       | f        | {(2)}
           5 |          2 | Elizabeth Tucker  | f        | {(2),(5)}
           6 |          2 | Joseph Lewis      | f        | {(2),(6)}
           7 |          2 | William Ferguson  | f        | {(2),(7)}
          12 |          4 | Donald Carter     | f        | {(2),(7),(12)}
          13 |          7 | Elizabeth Collins | f        | {(2),(7),(13)}
          10 |          3 | Daniel Gray       | f        | {(2),(5),(10)}
```

РЕКОМЕНДАЦИЯ

Запрос может включать как выражение `SEARCH`, так и `CYCLE`. Столбцы path цикла вычисляются так же, как столбцы сортировки "в глубину". Выражение `SEARCH DEPTH FIRST BY ... CYCLE` использовать вместе, создает избыточные вычисления. Чтобы избежать этого, используйте выражение `CYCLE` и интерпретируйте результаты по столбцу `path`. Если вам нужно применить сортировку "в ширину", используйте оба выражения, `SEARCH BREADTH FIRST BY ... CYCLE`.

Материализация CTE

Вы можете указать, будет ли **материализована** результат подзапроса CTE (начинающегося с AS). Это означает, что подзапрос выполнится отдельно от родительского запроса и только один раз во время его выполнения, а его результат будет записан во временную таблицу, которую и читает. Такое поведение может быть полезно, если подзапрос содержит ресурсоемкие вычисления.

Используйте конструкцию `MATERIALIZED`, чтобы принудительно вычислить подзапрос отдельно от родительского запроса:

```
WITH employees_data AS MATERIALIZED (
    SELECT * FROM employees
)
SELECT full_name FROM employees_data WHERE employee_id = 5;
```

Следующие подзапросы материализуются по умолчанию:

- подзапросы из рекурсивных запросов CTE;
- подзапросы, на которые ссылается более одного раза.

Поскольку материализованный подзапрос вычисляется и оптимизируется отдельно от родительского запроса, это может вызвать проблемы с производительностью. Поэтому вы можете принудительно встроить подзапрос в родительский запрос, используя конструкцию `NOT MATERIALIZED`. Это может привести к тому, что подзапрос не будет материализован. Оптимизатор обнаружит и устранит избыточные запросы и обрабатывает оба запроса одновременно. Например, оптимизатор может объединить условия для двух запросов в один фильтр.

Вы также можете использовать модификатор `NOT MATERIALIZED` для принудительной совместной оптимизации подзапроса и родительского запроса.

```
WITH employees_data AS NOT MATERIALIZED (
    SELECT * FROM employees
)
SELECT (
    SELECT full_name FROM employees_data WHERE employee_id = 5),
    (SELECT full_name FROM employees_data WHERE employee_id = 11);
```

Обратите внимание, что добавление модификатора `NOT MATERIALIZED` не влияет на рекурсивные CTE-запросы, они всегда материализуются.

Изменение данных

В выражении `WITH` можно также использовать операторы `INSERT`, `UPDATE` и `DELETE`. Это позволяет выполнять несколько различных операций в одном запросе. В приведенном ниже примере две строки перемещаются из таблицы `employees` в таблицу `retired_employees`:

```
WITH moved_rows AS (
    DELETE FROM employees
    WHERE
        employee_id > 10
        employee_id < 0
    RETURNING *
)
INSERT INTO retired_employee
SELECT * FROM moved_rows;
```

Здесь оператор `DELETE` удаляет указанные строки из `employees` и возвращает их содержимое с помощью `RETURNING`. Родительский запрос считывает выходные данные подзапроса и добавляет их в `retired_employee`.

Операторы `INSERT`, `UPDATE` и `DELETE`, используемые в `WITH`, должны содержать `RETURNING`, чтобы можно было сослаться на их результат в родительском запросе.

Обратите внимание, что порядок выполнения операторов, используемых в `WITH`, не определен. Гарантируется лишь выполнение всех из таблицы `employees`, поскольку оператор `UPDATE` выполняется позже, чем `SELECT`.

```
WITH employees_data AS (
    UPDATE employees SET manager_id = 1 WHERE employee_id =10
    RETURNING *
)
SELECT * FROM employees;
```

Чтобы получить измененные данные, обратитесь к результату подзапроса:

```
WITH employees_data AS (
    UPDATE employees SET manager_id = 1 WHERE employee_id =10
    RETURNING *
)
SELECT * FROM employees_data;
```

Рекурсивными ссылки в операторах, изменяющих данные, не допускаются, но вы можете включить `INSERT`, `UPDATE` или `DELETE` в родительский запрос. Следующий рекурсивный запрос удаляет сотрудника с `employee_id` 2 и всех его подчиненных из `employees`:

```
WITH RECURSIVE subordinates (employee_id, full_name, manager_id) AS (
   SELECT employee_id, manager_id, full_name
   FROM employees WHERE employee_id = 2
   UNION
      SELECT e.employee_id, e.manager_id, e.full_name
      FROM employees e
   INNER JOIN subordinates s ON s.employee_id = e.manager_id
)
DELETE FROM employees
   WHERE employee_id IN (SELECT employee_id FROM subordinates);
```

Агрегатные функции

Евгения Кузина

AD.PG

Содержание

Назад к содержанию

Обзор

Синтаксис агрегатных выражений

Обзор

Агрегатные функции вычисляют результат по набору входных значений. Например, можно вычислить количество, сумму, среднее, максимальное и минимальное значение по набору строк.

Пример:

```
SELECT MIN(public_year) FROM book;
```

Результат:

```
 min
------
 1869
```

Чтобы использовать агрегатную функцию в выражении `WHERE`, используйте подзапрос. Следующий пример возвращает название самой старой книги:

```
SELECT title FROM book WHERE public_year = (SELECT min(public_year) FROM book);
```

Результат:

```
     title
---------------
 War and Peace
```

Также можно комбинировать агрегатные функции с `GROUP BY`. Например, получим самую старую книгу каждого жанра:

```
SELECT genre, min(public_year) public_year FROM book GROUP BY genre;
```

Результат:

```
      genre      | public_year
-----------------+-------------
 science fiction |        1895
 fantasy         |        1937
 novel           |        1869
```

Чтобы фильтровать полученные значения, используйте `HAVING`.

```
SELECT genre, min(public_year) public_year
   FROM book GROUP BY genre HAVING min(public_year) < 1900;
```

Результат:

```
      genre      | min
-----------------+------
 science fiction | 1895
 novel           | 1869
```

ПРИМЕЧАНИЕ

Существует фундаментальное различие между выражениями `WHERE` и `HAVING`. `WHERE` выбирает входные строки перед группировкой и вычислением агрегатных функций. `WHERE` определяет, какие строки включаются в агрегатные вычисления. `HAVING` выбирает строки после группировки и расчета агрегатных функций. `WHERE` не может содержать агрегатные функции, а `HAVING` всегда их содержит.
Если вам нужно использовать дополнительные условия, добавьте их в `WHERE`, а не `HAVING`. Этот подход позволяет избежать агрегатных вычислений для строк, которые не соответствуют дополнительным условиям.

Полный список предопределенных агрегатных функций можно найти по ссылке: Aggregate functions.

Синтаксис агрегатных выражений

Агрегатное выражение может записываться одним из следующих способов:

1. Первая форма агрегатного выражения вызывает агрегатную функцию один раз для каждой входящей строки в соответствии с указанным выражением:

```
<название_агрегатной_функции> (<выражение> [ , ... ] [<order_by_выражение> ])
   [FILTER (WHERE <значение_filter>)]
```

   Где:

   - `<название_агрегатной_функции>` – название агрегатной функции;
   - `<выражение>` – выражение значения, которое не содержит какого-либо агрегатного выражения или вызова оконной функции;
   - `<order_by>` – необязательный параметр, который указывает порядок сортировки;
   - `<значение_filter>` – необязательный параметр, содержащий условие фильтра.

   Пример:

```
SELECT MIN(public_year) FROM book;
```

2. Вторая форма агрегатного выражения содержит модификатор `ALL`. Поскольку `ALL` является модификатором по умолчанию, вторая форма эквивалентна первой.

```
<название_агрегатной_функции> (ALL <выражение> [ , ... ] [<order_by_выражение>])
   [FILTER (WHERE <значение_filter>)]
```

3. Третья форма вызывает агрегатную функцию один раз для каждого уникального значения выражения или набора уникальных значений для нескольких выражений.

```
<название_агрегатной_функции> (DISTINCT <выражение> [ , ... ] [<order_by_выражение>])
   [FILTER (WHERE <значение_filter>)]
```

   Следующее выражение возвращает количество уникальных названий книг из таблицы с заказами:

```
SELECT COUNT (DISTINCT(book_title)) FROM orders;
```

4. Четвертая форма вызывает агрегатную функцию один раз для каждой строки. Поскольку выражение, задающее входящие значения, отсутствует, эта форма применима только к функции `COUNT(*)`.

```
<название_агрегатной_функции> ( * ) [FILTER (WHERE <значение_filter>)]
```

5. Пятая форма используется с агрегатными функциями, выполняемыми над упорядоченным набором значений (например, `array_agg`, `string_agg`, `rank` и `percentile`). Результат выполнения этих функций зависит от порядка входных строк. Для этих функций параметр `order_by` является обязательным.

```
<название_агрегатной_функции> ([<выражение> [ , ... ] ])
   WITHIN GROUP(<order_by_выражение>)
   [FILTER (WHERE <значение_filter>)]
```

   Выражение `order_by` имеет тот же синтаксис, что и `ORDER BY`, используемый на уровне запроса, но не может содержать номера и имена выходных столбцов.

   Пример:

```
SELECT array_agg(field1 ORDER BY field2 DESC) FROM table1;
```

   Если вы вызываете агрегатную функцию с несколькими аргументами, `ORDER BY` должен быть помещен после всех аргументов:

```
SELECT string_agg(field1, ',' ORDER BY field1) FROM table1;
```

   Если `DISTINCT` указан в дополнение к `ORDER BY`, выражение `ORDER BY` должно соответствовать аргументам агрегатной функции. Сортировка по значениям, не включенным в список `DISTINCT`, невозможна.

   Для агрегатной функции с упорядоченным набором входных значений, вы можете поместить выражение `order_by` внутрь выражения `WITHIN GROUP(…)`. В этом случае выражения в `order_by` вычисляются один раз для каждой входящей строки как обычные агрегатные аргументы, сортируются в соответствии с требованиями `order_by` и передаются агрегатной функции в качестве входных аргументов. Если `order_by` не находится в `WITHIN GROUP`, оно не рассматривается как аргумент агрегатной функции. Выражения аргументов, предшествующие `WITHIN GROUP`, если таковые имеются, называются прямыми аргументами, чтобы отличить их от агрегированных аргументов, перечисленных в выражении `order_by`. Прямые аргументы обрабатываются один раз за вызов агрегатной функции. Они могут содержать переменные, только если эти переменные сгруппированы по `GROUP BY`. Прямые аргументы обычно используются для единственного значения на расчет агрегации (например, доли процентилей). Если список прямых аргументов пуст, напишите `()`.

   Пример:

```
SELECT percentile_cont(0.5) WITHIN GROUP (ORDER BY income) FROM households;
```

Для всех типов агрегатных выражений, перечисленных выше, `значение_filter` является необязательным параметром. Если указан `FILTER`, в агрегатную функцию передаются только те входящие строки, для которых `значение_filter` вычисляется как `true`.

Например, имеется следующая таблица:

```
 id |            book_title              | public_year
----+------------------------------------+-------------
  1 | Mrs. Dalloway                      |        1925
  2 | To the Lighthouse                  |        1927
  3 | To Kill a Mockingbird              |        1960
  4 | The Great Gatsby                   |        1925
  5 | The Lord of the Rings              |        1955
  8 | War and Peace                      |        1869
  7 | The Hobbit, or There and Back Again |        1937
  9 | Hyperion                           |        1989
 10 | The Time Machine                   |        1895
  6 | 1984                               |        1949
```

Выполним запрос с вызовом агрегатной функции `COUNT(*)` и заданным фильтром:

```
SELECT
    COUNT(*) AS unfiltered,
    COUNT(*) FILTER (WHERE public_year < 1988) AS filtered
FROM book;
```

Результат:

```
 unfiltered | filtered
------------+----------
         10 |        2
```

Большинство агрегатных функций игнорирует входящее значение NULL, а строки, в которых одно или несколько значений содержат NULL, отбрасываются. Это верно для всех предопределенных агрегатных функций, если не указано иное. Например, `COUNT(*)` возвращает общее количество входных строк, `COUNT(field1)` возвращает количество входных строк, где `field1` не равно нулю, а `COUNT(DISTINCT field1)` возвращает количество уникальных строк с ненулевыми значениями `field1`.

Оконные функции

Евгений Кузина

AD.PG

Назад к содержанию

Оконные функции выполняют вычисления по набору строк. Они не возвращают одну результирующую строку как агрегатные функции, а включают дополнительную информацию в выборку.

Упрощенный синтаксис оконной функции можно записать следующим образом:

```
<имя функции> OVER (<окно>)
```

Где:

- `<имя функции>` – имя оконной функции;
- `<окно>` – выражение, описывающее набор строк для обработки и порядок обработки.

Для получения информации о расширенном синтаксисе оконных функций обратитесь к статье Window Function Calls.

Если вам нужно передать все строки результата выполнения в оконную функцию, используйте пустые круглые скобки `()`. Например, следующий запрос добавляет к результату номера строк:

```
SELECT id, title, evaluation,
    row_number() OVER () AS numbers
FROM books;
```

Результат:

```
 id |                 title                 | evaluation | numbers
----+---------------------------------------+------------+---------
  1 | Mrs. Dalloway                         |       7.82 |       1
  2 | To the Lighthouse                     |       8.45 |       2
  3 | To Kill a Mockingbird                 |       7.48 |       3
  4 | The Great Gatsby                      |       9.23 |       4
  5 | The Lord of the Rings                 |       9.49 |       5
  6 | 1984                                  |       8.17 |       6
  7 | The Hobbit, or There and Back Again   |       9.32 |       7
  8 | War and Peace                         |       9.69 |       8
  9 | Hyperion                              |       9.46 |       9
 10 | The Time Machine                      |       8.12 |      10
```

Вы можете добавить выражение `ORDER BY` к оконной функции, чтобы изменить порядок обработки:

```
SELECT id, title, genre, evaluation,
    row_number() OVER (ORDER BY evaluation DESC) AS rating
FROM books
ORDER BY id;
```

Результат:

```
 id |                 title                 |  genre  | evaluation | rating
----+---------------------------------------+---------+------------+--------
  1 | Mrs. Dalloway                         | novel   |       7.82 |      9
  2 | To the Lighthouse                     | novel   |       8.45 |      6
  3 | To Kill a Mockingbird                 | novel   |       7.48 |     10
  4 | The Great Gatsby                      | novel   |       9.23 |      5
  5 | The Lord of the Rings                 | fantasy |       9.49 |      2
  6 | 1984                                  | sci-fi  |       8.17 |      7
  7 | The Hobbit, or There and Back Again   | fantasy |       9.32 |      4
  8 | War and Peace                         | novel   |       9.69 |      1
  9 | Hyperion                              | sci-fi  |       9.46 |      3
 10 | The Time Machine                      | sci-fi  |       8.12 |      8
```

В приведенном выше примере в запрос добавлена сортировка на уровне запроса (`ORDER BY id`). PostgreSQL сначала оценивает результаты оконной функции, а затем сортирует результаты запроса вместе с ее результатами в соответствии с `ORDER BY id`. `ORDER BY` на уровне окна и `ORDER BY` на уровне запроса применяются корректно и не оказывают негативного влияния друг на друга.

Вы можете использовать выражение `PARTITION BY`, чтобы разделить строки на группы или **партиции** (partitions), которые используют одни и те же значения выражения `PARTITION BY`. Оконная функция вычисляет значение строки по строкам одной партиции. В следующем примере вычисляется рейтинг книг одного жанра:

```
SELECT id, title, genre, evaluation,
    row_number() OVER (PARTITION BY genre ORDER BY evaluation DESC) AS rating
FROM books
ORDER BY genre;
```

Результат:

```
 id |                 title                 |  genre  | evaluation | rating
----+---------------------------------------+---------+------------+--------
  5 | The Lord of the Rings                 | fantasy |       9.49 |      1
  7 | The Hobbit, or There and Back Again   | fantasy |       9.32 |      2
  8 | War and Peace                         | novel   |       9.69 |      1
  4 | The Great Gatsby                      | novel   |       9.23 |      2
  2 | To the Lighthouse                     | novel   |       8.45 |      3
  1 | Mrs. Dalloway                         | novel   |       7.82 |      4
  3 | To Kill a Mockingbird                 | novel   |       7.48 |      5
  9 | Hyperion                              | sci-fi  |       9.46 |      1
  6 | 1984                                  | sci-fi  |       8.17 |      2
 10 | The Time Machine                      | sci-fi  |       8.12 |      3
```

Если не указывать `PARTITION BY`, то партицией являются все строки.

Список предопределенных оконных функций доступен по следующей ссылке: General-purpose window functions.

Вы также можете использовать агрегатные функции `sum`, `count` и другие в качестве оконных функций. Если `ORDER BY` не указан в окне функции, функция вычисляется для всей партиции, и ее результат записывается во все строки партиции.

Например, имеется таблица `orders`:

```
 id | customer_id |  total
----+-------------+---------
  1 |          17 | 3500.00
  2 |          22 | 1800.00
  4 |          46 | 3300.00
 11 |          38 | 2800.00
 12 |          17 | 1750.00
  3 |          17 | 5600.00
  5 |          17 | 4700.00
  6 |          17 | 2800.00
  7 |          46 | 1900.00
  8 |          22 | 2100.00
  9 |          22 | 1400.00
 10 |          13 | 1400.00
 13 |          46 | 4600.00
 14 |          13 | 2250.00
```

Следующий запрос вычисляет сумму заказов каждого клиента:

```
SELECT id, customer_id,
    sum(total) OVER (PARTITION BY customer_id) as sum
FROM orders;
```

Результат:

```
 id | customer_id |   sum
----+-------------+----------
 10 |          13 |  3650.00
 14 |          13 |  3650.00
  1 |          17 | 16950.00
 12 |          17 | 16950.00
  3 |          17 | 16950.00
  5 |          17 | 16950.00
  6 |          17 | 16950.00
  2 |          22 |  4500.00
  9 |          22 |  4500.00
  8 |          22 |  4500.00
 11 |          38 |  2800.00
  7 |          46 |  9800.00
  4 |          46 |  9800.00
 13 |          46 |  9800.00
```

Если указан `ORDER BY`, функция вычисляется для строк от начала раздела до текущей строки. Если следующая строка содержит такое же значение поля, которое указано в `ORDER BY`, она включается в расчет. Этот набор строк называется **фреймом** (window frame). Некоторые оконные функции вычисляются на основании фрейма, а не всей партиции. Например, функция `SUM` работает с фреймами. Добавим `ORDER BY` к приведенному выше примеру:

```
SELECT id, customer_id,
    sum(total) OVER (PARTITION BY customer_id ORDER BY id) as sum
FROM orders;
```

Результат:

```
 id | customer_id |   sum
----+-------------+----------
 10 |          13 |  1400.00
 14 |          13 |  3650.00
  1 |          17 |  3500.00
  3 |          17 |  8500.00
  5 |          17 | 13200.00
  6 |          17 | 15200.00
 12 |          17 | 16950.00
  2 |          22 |  1800.00
  8 |          22 |  3100.00
  9 |          22 |  4500.00
 11 |          38 |  2800.00
  4 |          46 |  3300.00
  7 |          46 |  5200.00
 13 |          46 |  9800.00
```

Результат содержит сумму нарастающим итогом для каждой партиции.

Вы можете использовать оконные функции в выражениях `SELECT` и `ORDER BY`. Они не могут быть включены в `GROUP BY`, `HAVING` `WHERE` и другие, поскольку оконные функции вычисляются после обработки этих выражений. Кроме того, оконные функции выполняются после неоконных агрегатных функций, поэтому вы можете включить вызов агрегатной функции в аргументы оконной функции.

Чтобы отфильтровать или сгруппировать строки после расчета оконной функции, используйте подзапросы.

Следующий запрос использует подзапрос с фильтром для отображения строк, где `sum` меньше `5000`:

```
SELECT id, customer_id, sum
  FROM
    (SELECT id, customer_id, sum(total) OVER (PARTITION BY customer_id ORDER BY id) as sum
       FROM orders
    )AS orders_with_sum
  WHERE sum < 5000;
```

Результат:

```
 id | customer_id |   sum
----+-------------+---------
 10 |          13 | 1400.00
 14 |          13 | 3650.00
  1 |          17 | 3500.00
  2 |          22 | 1800.00
  8 |          22 | 3100.00
  9 |          22 | 4500.00
 11 |          38 | 2800.00
  4 |          46 | 3300.00
```

В одном запросе можно использовать несколько оконных функций:

```
SELECT id, customer_id,
    sum(total) OVER (PARTITION BY customer_id ORDER BY id) as sum,
    round(avg(total) OVER(PARTITION BY customer_id),2) as avg
FROM orders;
```

Результат:

```
 id | customer_id |   sum    |   avg
----+-------------+----------+---------
 10 |          13 |  1400.00 | 1825.00
 14 |          13 |  3650.00 | 1825.00
  1 |          17 |  3500.00 | 3390.00
  3 |          17 |  8500.00 | 3390.00
  5 |          17 | 13200.00 | 3390.00
  6 |          17 | 15200.00 | 3390.00
 12 |          17 | 16950.00 | 3390.00
  2 |          22 |  1800.00 | 1500.00
  8 |          22 |  3100.00 | 1500.00
  9 |          22 |  4500.00 | 1500.00
 11 |          38 |  2800.00 | 2800.00
  4 |          46 |  3300.00 | 3266.67
  7 |          46 |  5200.00 | 3266.67
 13 |          46 |  9800.00 | 3266.67
```

Если несколько оконных функций имеют одно и то же выражение `OVER`, можно дать ему имя и поместить отдельно с помощью ключевого слова `WINDOW`. Запрос, приведенный выше, можно переписать следующим образом:

```
SELECT id, customer_id,
    sum(total) OVER w as sum,
    round(avg(total) OVER(w),2) as avg
FROM orders
WINDOW w AS (PARTITION BY customer_id);
```

Результат:

```
 id | customer_id |   sum    |   avg
----+-------------+----------+---------
 10 |          13 |  1400.00 | 1825.00
 14 |          13 |  3650.00 | 1825.00
  1 |          17 |  3500.00 | 3390.00
  3 |          17 |  8500.00 | 3390.00
  5 |          17 | 13200.00 | 3390.00
  6 |          17 | 15200.00 | 3390.00
 12 |          17 | 16950.00 | 3390.00
  2 |          22 |  1800.00 | 1500.00
  8 |          22 |  3100.00 | 1500.00
  9 |          22 |  4500.00 | 1500.00
 11 |          38 |  2800.00 | 2800.00
  4 |          46 |  3300.00 | 3266.67
  7 |          46 |  5200.00 | 3266.67
 13 |          46 |  9800.00 | 3266.67
```

Типы данных JSON/JSONB

© Антон Бутко

Поддержать автора

Содержание

- Типы данных JSON и JSONB
  - Кодировка JSON и JSONB
  - Конвертация JSONB
- Синтаксис JSON и JSONB
- Сохранение данных JSON в таблицу
- Использование операторов в запросах данных JSON
- Использование операторов JSON в выражении WHERE
- Операторы "содержит" и "существует"
- Применение агрегатных функций к данным JSON
- Функции PostgreSQL для работы с типами JSON и JSONB
  - json_each/jsonb_each
  - json_object_keys/jsonb_object_keys
  - json_typeof/jsonb_typeof
  - json_extract_path/jsonb_extract_path
  - jsonb_pretty

PostgreSQL предоставляет два типа для хранения данных в стандарте JSON, определенного в RFC 7159 — JSON и JSONB. Чтобы реализовать эффективные запросы, PostgreSQL также предоставляет специальный тип данных JSONPATH.

Типы данных JSON и JSONB

JSON и JSONB — это типы данных, которые принимают на вход данные JSON. Оба типа данных используют одни и те же проверки входных данных. Разница между JSON и JSONB заключается в том, как эти данные хранятся.

JSONB хранит данные в бинарном формате, что делает его медленнее при вводе, но быстрее при обработке данных. Поскольку JSONB не хранит пробелы, дублирующиеся ключи и порядок ключей, он занимает меньше места на диске, чем JSON. Однако JSONB хранит данные немного дольше.

JSONB — это предпочтительный формат для работы с данными JSON, если только вам не нужно сохранить точный порядок ключей. Например, если вы хотите сохранить данные в том же порядке, в котором они были получены, или если вы используете функцию, которая зависит от порядка ключей.

JSONB поддерживает индексы. JSON — нет.

Оба типа данных JSON и JSONB имеют ограничения. JSON, в PostgreSQL, выполняет только проверки на корректность входных данных. Если PostgreSQL может преобразовать данные в JSON, они принимаются. Однако JSONB выполняет дополнительную проверку данных. Например, JSONB не принимает данные, которые содержат дублирующиеся ключи.

JSONB также имеет ограничение на размер данных. Он ограничен размером 255 МБ.

Кодировка JSON и JSONB

Стандарт RFC 7159 указывает, что строки JSON должны быть закодированы в UTF8. Это означает, только в том случае, если PostgreSQL был настроен для использования другой кодировки символов, строки JSON будут проверены, чтобы убедиться, что они являются допустимыми строками UTF8. Точно так же, двоичные данные, которые должны быть преобразованы в JSON, также должны использовать кодировку UTF8. Символы, которые могут быть представлены в одной кодировке, но не могут быть представлены в UTF8, не будут разрешены.

Стандарт JSON также указывает, что числа, которые хранятся в JSON, должны соответствовать спецификации IEEE 754. Это означает, что PostgreSQL, скорее всего, будет принимать числа, которые соответствуют этому стандарту, но некоторые специальные значения, такие как Infinity и NaN, могут быть отклонены.

Конвертация JSONB

Тип преобразования зависимого типа в JSONB применяется типы JSON-совместимого типа данных. Типы PostgreSQL, которые не совместимы с JSON, могут быть преобразованы в JSON, если они определяют приведение типа. Приведение типа — это функция, которая сообщает PostgreSQL, как преобразовать данные из одного типа данных в другой. Если приведение типа не определено, вы можете создать свое собственное приведение типа.

Примитивные типы JSON	Типы PostgreSQL	Примечания
string	text	Escape-последовательности \u0000 запрещены, как и последовательности этого вида, которые недопустимы в представлении символов, используемом базой данных
number	numeric	Значения NaN и Infinity не допускаются
boolean	boolean	Значения true и false допускаются только в нижнем регистре
null	(none)	NULL в SQL имеет другой смысл

Синтаксис JSON и JSONB

Следующие выражения JSON являются допустимыми типами JSON и JSONB:

- Простые скалярные/примитивные значения. Простые значения могут быть числа, строки, логическими и иметь тип NULL. Пример:

```
SELECT '1'::jsonb;
SELECT 'hard string'::jsonb;
```

- Массивы из нуля и более элементов. Элементы могут быть разных типов. Пример:

```
SELECT '[1, 2, "string element", null]'::jsonb;
```

- Объекты, содержащие пары «ключ/значение». Ключи — это строки в массиве. Пример:

```
SELECT '{"key": "key", "values": 7.77, "active": false}'::jsonb;
```

- Вложенные массивы и объекты. Пример:

```
SELECT '{"key": {true, "key"}, "key": 3, "k": null}'::jsonb;
```

Сохранение данных JSON в таблицу

Чтобы сохранить данные JSON в таблицу, используйте столбец типа JSONB или JSON. Например, создадим новую таблицу book_orders со столбцом JSONB:

```
CREATE TABLE book_orders (
    id serial NOT NULL PRIMARY KEY,
    content jsonb NOT NULL
);
```

Чтобы не хранить данные «Мусор», это данные перед вставкой в другую базу данных JSON. Следующий код добавляет новые строки в таблицу book_orders:

```
INSERT INTO book_orders (content)
VALUES ('{"customer": "John Doe", "items": {"book": "Learn PostgreSQL", "qty": 1}}'),
       ('{"customer": "Jane Doe", "items": {"book": "Learn PostgreSQL", "qty": 2}}'),
       ('{"customer": "Andrew Mellen", "items": {"book": "Star Wars", "qty": 1}}');
```

Использование операторов в запросах данных JSON

Используйте SELECT со столбцом данных JSON. Не так, как в случае с другими типами данных:

```
SELECT content FROM book_orders;
```

Результат:

```
                                  content
-------------------------------------------------------------------------
 {"items": {"qty": 1, "book": "Star and Peace"}, "customer": "Hope Brown"}
 {"items": {"qty": 1, "book": "1984"}, "customer": "Andrew Mellen"}
 {"items": {"qty": 2, "book": "Hyperion"}, "customer": "Jacob Johnson"}
(3 rows)
```

PostgreSQL также поддерживает операторы, для работы с типами данных JSON. Например, оператор -> возвращает объект JSON из ключа, а ->> возвращает объект JSON из ключа в виде текста.

В следующем запросе оператор -> используется для получения значения из внутри объекта JSON:

```
SELECT content -> 'customer' AS customer
FROM book_orders;
```

Результат:

```
   customer
---------------
 "John Doe"
 "Jane Doe"
 "Andrew Mellen"
(3 rows)
```

В приведенном выше запросе используется оператор ->, чтобы получить всех покупателей в текстовой форме:

```
SELECT content ->> 'customer' AS customer
FROM book_orders;
```

Результат:

```
   customer
---------------
 John Doe
 Jane Doe
 Andrew Mellen
(3 rows)
```

Поскольку операторы -> возвращают объект JSON, можно использовать его с операторами ->> для получения конкретных значений. Например, следующее выражение возвращает все названия книг:

```
SELECT content -> 'items' ->> 'book' AS book
FROM book_orders;
```

Результат:

```
     book
---------------
 Star and Peace
 1984
 Hyperion
(3 rows)
```

content -> 'items' возвращает JSONB, как объекты JSON. content ->> 'items' тоже возвращает значения типа text.

Следующий оператор возвращает объекты JSON по типу 1984 book:

```
SELECT content AS books FROM book_orders
WHERE content ->> 'items' ->> 'book' = '1984';
```

Результат:

```
                                  books
-------------------------------------------------------------------------
 {"items": {"qty": 1, "book": "1984"}, "customer": "Andrew Mellen"}
(1 row)
```

PostgreSQL также поддерживает операторы, для работы с типами данных JSON. Например, оператор @> возвращает объект JSON из ключа, а <@ возвращает объект JSON из ключа в виде текста.

В следующем запросе оператор @> используется для получения книги покупателя в виде объектов JSON:

```
SELECT content -> 'customer' AS customer,
content -> 'items' -> 'book' AS book
FROM book_orders
WHERE CAST ( content -> 'items' -> 'qty' AS INTEGER) = 2;
```

Результат:

```
   customer    |     book
---------------+---------------
 "Jane Doe"    | "Hyperion"
(1 row)
```

Вы также можете искать объекты JSON по указанному пути в виде текста. Для этого используйте оператор #>>:

```
SELECT content #>> '{items, book}' AS books
FROM book_orders;
```

Результат:

```
     books
---------------
 Star and Peace
 1984
 Hyperion
(3 rows)
```

Использование операторов JSON в выражении WHERE

Вы можете использовать операторы JSON в выражении WHERE для фильтрации строк. Например, следующий запрос получает купивших книгу "Звёзды и мир":

```
SELECT content AS "customer" FROM book_orders
WHERE content -> 'items' ->> 'book' = 'Star and Peace';
```

Результат:

```
                                  customer
-------------------------------------------------------------------------
 {"items": {"qty": 1, "book": "Star and Peace"}, "customer": "Hope Brown"}
(1 row)
```

Следующий запрос возвращает объекты JSON по типу 2 книги Hyperion:

```
SELECT content AS "customer", content -> 'items' -> 'qty' AS QUANTITY
FROM book_orders
WHERE CAST ( content -> 'items' -> 'qty' AS INTEGER) = 2;
```

Результат:

```
                                  customer                                  | quantity
----------------------------------------------------------------------------+----------
 {"items": {"qty": 2, "book": "Hyperion"}, "customer": "Jacob Johnson"}     | 2
(1 row)
```

Операторы "содержит" и "существует"

Можно проверить первое условие, WHERE с помощью дополнительного оператора JSONB @>, который определяет "содержит" по первым значениям JSONB второго:

```
SELECT content AS "customer" FROM book_orders
WHERE content -> 'items' -> 'book' @> 'Star and Peace';
```

Получим такой же результат:

```
                                  customer
-------------------------------------------------------------------------
 {"items": {"qty": 1, "book": "Star and Peace"}, "customer": "Hope Brown"}
(1 row)
```

Оператор @> использует условие возможности сравнить. Он определяет, содержит ли объект JSONB справа от него такое же объекта или вложенную форму. Следующий пример возвращает true:

```
SELECT content @ '{"customer":
FROM book_orders;
```

Для получения дополнительных операторов об операторах JSONB обратитесь в статью Additional JSONB operators.

Применение агрегатных функций к данным JSON

Агрегатные функции json_agg, чтобы агрегировать данные выражения в массив JSON, следующий выражение агрегирует книги покупателя, заказавшего в нескольких экземплярах. Следующий запрос использует функцию json_agg, чтобы агрегировать все книги в массив JSON и возвращает имя покупателя и массив книг, которые они заказали:

```
SELECT
  BOOKORDERS.CONT ->> 'customer' AS "CUSTOMER",
  JSON_AGG(BOOKORDERS.CONT -> 'items' -> 'book') AS BOOKS,
  SUM (CAST (BOOKORDERS.CONT -> 'items' -> 'qty' AS INTEGER)) AS TOTAL
FROM book_orders BOOKORDERS (CONT)
GROUP BY BOOKORDERS.CONT ->> 'customer';
```

Результат:

```
 customer |          books          | total
----------+-------------------------+-------
 Jane Doe | ["Hyperion", "Hyperion"] |     4
(1 row)
```

Функции PostgreSQL для работы с типами JSON и JSONB

PostgreSQL предоставляет несколько функций для работы с типами данных JSON и JSONB, специфичных для этих типов данных.

json_each/jsonb_each

Функции json_each и jsonb_each принимают объект JSON-объект верхнего уровня и набор пар ключ-значение.

Функция имеет следующий синтаксис:

```
json_each ([column] | value) [as] [key_set_name, value_set_name]
jsonb_each ([column] | value) [as] [key_set_name, value_set_name]
```

Пример:

```
SELECT jsonb_each(content)
FROM book_orders;
```

Результат:

```
                  jsonb_each
-----------------------------------------------
 (items,"{""qty"": 1, ""book"": ""Star and Peace""}")
 (customer,"""Hope Brown""")
 (items,"{""qty"": 1, ""book"": ""1984""}")
 (customer,"""Andrew Mellen""")
 (items,"{""qty"": 2, ""book"": ""Hyperion""}")
 (customer,"""Jacob Johnson""")
(6 rows)
```

Чтобы получить набор пар ключ-значение для объектов JSON, используйте функцию json_each или jsonb_each_text:

json_object_keys/jsonb_object_keys

Функции json_object_keys и jsonb_object_keys возвращают набор ключей объекта JSON верхнего уровня.

Выражение имеет следующий синтаксис:

```
json_object_keys([column] | value) [as] [key_set_name]
jsonb_object_keys([column] | value) [as] [key_set_name]
```

Выражение ниже отображает набор ключей для объектов JSONB:

```
SELECT jsonb_object_keys (content) FROM book_orders;
```

Результат:

```
 jsonb_object_keys
-------------------
 items
 customer
 items
 customer
 items
 customer
(6 rows)
```

json_typeof/jsonb_typeof

Функции json_typeof и jsonb_typeof возвращают тип значения. Функции JSON-операнд и возвращают строку. Значения возвращаемого: number, boolean, null, object, array, string.

Функция имеет следующий синтаксис:

```
json_typeof([column] | value)
jsonb_typeof([column] | value)
```

Выражение ниже отображает тип объектов qty:

```
SELECT jsonb_typeof (content -> 'items' -> 'qty')
FROM book_orders;
```

Результат:

```
 jsonb_typeof
--------------
 number
 number
 number
(3 rows)
```

json_extract_path/jsonb_extract_path

Функции json_extract_path и jsonb_extract_path извлекают объекты JSON по указанному пути. Это извлекается по операции #>.

Функция имеет следующий синтаксис:

```
json_extract_path ( <from_json> [key, <VARIADIC path_elems> text[] ) → json
jsonb_extract_path ( <from_json> [key, <VARIADIC path_elems> text[] ) → jsonb
```

Выражение ниже отображает значения объекта qty:

```
SELECT jsonb_extract_path (content, 'items', 'qty')
FROM book_orders;
```

Результат:

```
 jsonb_extract_path
--------------------
 1
 1
 2
(3 rows)
```

jsonb_pretty

Функция jsonb_pretty преобразует входящие значения JSON в форматную улучшенную текстовую представленность utf8/jsonb.

Функция имеет следующий синтаксис:

```
jsonb_pretty ([column] | text)
```

В следующем примере отображается объекты jsonb:

```
SELECT jsonb_pretty(content) as "json"
FROM book_orders;
```

Результат:

```
                  json
-----------------------------------------
 {                                      +
     "qty": 1,                          +
     "book": "Star and Peace"           +
 }
 {                                      +
     "qty": 1,                          +
     "book": "1984"                     +
 }
 {                                      +
     "qty": 2,                          +
     "book": "Hyperion"                 +
 }
(3 rows)
```

Тип JSONPath

Содержание

Назад к содержанию

Функции JSONPath
- jsonb_path_exists
- jsonb_path_match
- jsonb_path_query
- jsonb_path_query_array
- jsonb_path_query_first
- Функции JSONPath с постфиксом _tz
- Аргумент vars
- Аргумент silent

Операторы JSONPath
- Оператор совпадения @@
- Оператор проверки существования @?

Режимы strict и lax

Тип **JSONPath** реализует поддержку языка путей SQL/JSON в PostgreSQL для эффективных запросов данных типа JSONB. JSONPath является декларативным языком для поиска элементов по структуре JSON, которая напоминает элементы для извлечения в JavaScript.

JSONPath состоит из последовательности элементов (разделенных точкой `.`), которые могут быть:

- Ключи (без кавычек) или `*` для ссылки на ключи объектов JSON.
- Квадратные скобки `[]` используются для доступа к массиву.
- Функции элементов массива SQL/JSON начинаются с `.` и включают в обычных массивах PostgreSQL, которые начинают примерно элементом с `1`.

Все элементы записываются вместе (без разделителей) для создания выражения, чтобы использовать в запросе SQL. Чтобы использовать в выражении переменную значения, используйте `$variable`, где `variable` — это имя переменной, которая будет передана.

Выражение может быть заключено в скобки для группировки операторов. Подобные методы доступа могут использоваться в дробных числах. Они начинаются с `?` и используются для фильтрации элементов, которые соответствуют определенному условию. Внутри условия используйте `@` для ссылки на текущий элемент, который проверяется. Операторы сравнения включают `==`, `!=` (или `<>`), `<`, `>`, `<=` и `>=`. Булевы операторы `&&`, `||` и `!` также доступны для объединения условий. Шаблоны строк могут быть сопоставлены с помощью `like_regex`, который работает аналогично оператору `~` PostgreSQL. Функции `.double()`, `.ceiling()`, `.floor()` и `.abs()` доступны для числовых значений, определенных данными спецификациями JSON. Шаблон `.keyvalue()` может использоваться для доступа к парам ключ-значение объекта.

```
$.isvalid()
```

Выражение должно работать с логическим выражением `BOOLEAN`.

- Операторы и методы JSONPath: https://www.postgresql.org/docs/current/functions-json.html#FUNCTIONS-SQLJSON-PATH
- SQL/JSON Operators And Methods: https://www.postgresql.org/docs/current/functions-json.html#FUNCTIONS-SQLJSON-OPERATORS
- Кратные особые, применяющиеся для обращения выражений доступа к элементам порядка выполнения типа.

Доступ к данным, представленным в таблице ниже.

Оператор доступа	Описание	Пример
`.key` `."key_переменная"`	Оператор доступа к ключу по ведущему ключу. Если имя ключа совпадает с именем ключевого слова или содержит не буквенно-цифровые символы, используйте двойные кавычки. Двойные кавычки также используются в JavaScript-переменных и идентификаторах, где должны заключаться в двойные кавычки.	`$.menu`
`*` `[*]`	Текущий элемент доступа, который проходит по всем уровням иерархии JSON. Используется в функциях, которые поддерживают множественные значения, или для извлечения значений из `NULL`.	`$.menu.*`
`**{уровень}` `**{начальный_уровень to конечный_уровень}`	Этот оператор выбирает только указанные уровни иерархии JSON. Уровни вложенности задаются целыми числами, при этом нулевой уровень соответствует текущему объекту. Для обращения к вложенным уровням используется массив значений. Последние уровни могут быть включены в диапазон, если они больше, чем вложенность структуры.	`$.**{1}` `$.**{2 to 3}`
`[индекс]` `[начальный_индекс to конечный_индекс]`	Оператор обращается к элементам массива. Синтаксис может задаваться в двух формах: `[индекс]` или `[начальный_индекс to конечный_индекс]`. Полная форма выбирает определенный элемент или диапазон элементов. Второй формат выбирает один элемент массива. Индексы массива начинаются с `0`. Индексы могут быть числами или `last`, обозначающими индекс последнего элемента.	`$.a[1]` `$.a[0 to 10]`
`.*`	Оператор обращается к элементам массива, возвращающий все элементы массива.	`$.bing[*]`

Все элементы использования JSONPath для создания более простых запросов, чем позволяет синтаксис SQL.

Функции JSONPath

Функции JSONPath работают с данными в формате JSONB. Все эти функции имеют необязательный аргумент `vars` и `silent` `bool` позволяет передавать переменные в выражение JSONPath. `silent` определяет, должно ли выражение не обрабатывать ошибки. Это необязательный аргумент со значением `false` по умолчанию.

Создадим таблицу `book_orders` и `insert` для чистых примеров:

```
CREATE TABLE book_orders (
    id serial NOT NULL PRIMARY KEY,
    content JSONB NOT NULL
);

INSERT INTO book_orders (content)
VALUES ('{"email": "jacob_jensen", "items": {"item": "Hyperion","qty": 1, "price": 2111},
    { "supplier": "Hyde Brown", "items": {"item": "War and Peace","qty": 2, "price": 2413},
    { "supplier": "Andrew Nelson", "items": {"item": "Ulysses","qty": 1, "price": 812}}
;

CREATE TABLE deseret(name text, age_price int);

INSERT INTO deseret (name, age_price) VALUES ('Jane', 25), ('Alex', 27), ('Mike', 35);
```

jsonb_path_exists

```
jsonb_path_exists ( <значение> jsonb, <путь> jsonpath [, vars jsonb [, silent boolean ]] ) → boolean
```

Функция определяет возвращает `true` какой-либо элемент для значений JSON.

Примеры:

```
SELECT jsonb_path_exists('{"a": 1}', '$.a');
-- возвращает true

SELECT jsonb_path_exists('{"a": 1}', '$.b');
-- возвращает false

SELECT jsonb_path_exists('{"a":[1,2,3,4,5]}', '$.a[7 > 2]');
-- возвращает true

SELECT jsonb_path_exists('{"a":[1,2,3,4,5]}', 'strict$.a[7 > 2]' => $vars);
ERROR: Invalid SQL/JSON subscript
```

Следующий запрос строки содержит `Hyperion` в `content` таблицы `book_orders` строки со значением `Hyperion`.

```
SELECT jsonb_path_exists(content, '$ ? (@ == "Hyperion")')
FROM book_orders ;
```

`$.vr` передал от жирного значений JSON из поля `content` и возвращает все значения элементов, независимо от уровня вложенности. `@` ссылается на текущее значение элемента в выражении фильтра.

Результат:

```
jsonb_path_exists
-------------------
true
false
false
```

jsonb_path_match

```
jsonb_path_match ( <значение> jsonb, <путь> jsonpath [, vars jsonb [, silent boolean ]] ) → boolean
```

Функция возвращает результат проверки предиката `JSON` для значения JSON. Функция учитывает только первый элемент результата. Если результат не `boolean`, `jsonb_path_match` возвращает `NULL`.

Примеры:

```
SELECT jsonb_path_match('{"a": 1}', '$.a == 1');
-- возвращает true

SELECT jsonb_path_match('{"a": 1}', '$.a');
-- возвращает NULL

SELECT jsonb_path_match('{"a":[1,2,3,4,5]}', '$.a[7 > 2]');
-- возвращает true

SELECT jsonb_path_match('{"a":[1,2,3,4,5]}', 'strict$.a[7 > 2]' => $vars);
ERROR: Invalid SQL/JSON subscript
```

Следующий запрос также определяет, содержит ли столбец `content` таблицы `book_orders` строки со значением `Hyperion`.

```
SELECT jsonb_path_match(content, 'strict$.items.* ? (@ == "Hyperion")')
FROM book_orders ???? no qty;
```

Результат:

```
jsonb_path_match
-------------------
true
false
false
```

jsonb_path_query

```
jsonb_path_query ( <значение> jsonb, <путь> jsonpath [, vars jsonb [, silent boolean ]] ) → setof jsonb
```

Функция возвращает все элементы значения JSON. Функция учитывает только первый элемент результата.

Примеры:

```
SELECT jsonb_path_query('{"a": [1,2,3,4,5]}', '$.a[3 > 2]');
-- возвращает 3, 4, 5

SELECT jsonb_path_query('{"a": [1,2,3,4,5]}', '$.a[7 > 5]');
-- возвращает ничего

SELECT jsonb_path_query('{"a": [1,2,3,4,5]}', 'strict$.a[7 > 5]' => $vars);
ERROR: Invalid SQL/JSON subscript
```

Следующий запрос возвращает `items` цитаты `qty` больше 2.

```
SELECT jsonb_path_query(content, '$.items[*] ? (@.qty > 2)')
FROM book_orders;
```

Результат:

```
                jsonb_path_query
-----------------------------------------------
{"qty": 3, "item": "Hyperion", "price": 21}
{"qty": 4, "item": "Ulysses", "price": 812}
```

jsonb_path_query_array

```
jsonb_path_query_array ( <значение> jsonb, <путь> jsonpath [, vars jsonb [, silent boolean ]] ) → jsonb
```

Функция возвращает все элементы значения JSON, результат помещается в `JSONB` в виде массива JSON.

Примеры:

```
SELECT jsonb_path_query_array('{"a": [1,2,3,4,5]}', '$.a[7 > 2]');
-- возвращает [3, 4, 5]

SELECT jsonb_path_query_array('{"a": [1,2,3,4,5]}', '$.a[7 > 5]');
-- возвращает []

SELECT jsonb_path_query_array('{"a": [1,2,3,4,5]}', 'strict$.a[7 > 5]' => $vars);
ERROR: Invalid SQL/JSON subscript
```

jsonb_path_query_first

```
jsonb_path_query_first ( <значение> jsonb, <путь> jsonpath [, vars jsonb [, silent boolean ]] ) → jsonb
```

Функция возвращает первый элемент значения JSON, который соответствует значению `путь` `jsonb_path_query_first` возвращает `NULL` если соответствующее значение не найдено.

Примеры:

```
SELECT jsonb_path_query_first('{"a": [1,2,3,4,5]}', '$.a[7 > 2]');
-- возвращает 3

SELECT jsonb_path_query_first('{"a": [1,2,3,4,5]}', '$.a[7 > 5]');
-- возвращает ничего

SELECT jsonb_path_query_first('{"a": [1,2,3,4,5]}', 'strict$.a[7 > 5]' => $vars);
ERROR: Invalid SQL/JSON subscript
```

Функции JSONPath с постфиксом _tz

В PostgreSQL есть следующие функции с постфиксом `_tz`: `jsonb_path_exists_tz` `jsonb_path_match_tz` `jsonb_path_query_tz` `jsonb_path_query_array_tz` и `jsonb_path_query_first_tz`. Эти функции работают как аналогичные функции без `_tz`, но поддерживают сравнение дат и времени, которые включают часовой пояс. Часовой пояс учитывается при сравнении значений `timestamp`.

В приведенном ниже примере пример пути сравнивается дата значения `2023-08-02` и результатом у литтле человек поиск. Результат будет зависеть от часового пояса сервера. Функция `to_jsonb` для преобразования даты в значение JSON. Это означает, что это функция может быть использована в выражении пути.

Примеры:

```
SELECT jsonb_path_exists_tz('"2023-08-01 12:00:00 -05"', '$?(< ("2023-08-02" .datetime())');
-- возвращает true

SELECT jsonb_path_exists_tz('"2023-08-01 12:00:00"', '$?(< ("2023-08-02" .datetime())');
-- возвращает true
```

Аргумент vars

Аргумент `vars` позволяет передавать именованные переменные в выражение JSONPath. Значение `vars` должно быть объектом JSON, в его ключи должны содержать именованные значения для ссылки в выражении JSONPath.

Например, мы можем использовать `vars` для поиска книги с целой ценой, чем `age_price` которая указана для `Alex` в таблице `deseret`.

```
SELECT jsonb_path_query(content, '$.items[*] ? (@.price > $min_price)',
  vars => jsonb_build_object('min_price', deseret.age_price))
FROM book_orders, deseret WHERE name = 'Alex';
```

Аргумент `vars` передает результат запроса, который определяет `min_price` значением для `Alex` и функции `jsonb_path_query`.

Результат:

```
                jsonb_path_query
-----------------------------------------------
{"qty": 3, "item": "Hyperion", "price": 21}
{"qty": 2, "item": "War and Peace", "price": 24}
```

Аргумент silent

`silent` подавляет обработку ошибок. Выполним следующий запрос, в котором индексируется `[0:2]` может решим обработки ошибок.

```
SELECT jsonb_path_query('[]', 'strict $.a');
```

Произойдет ошибка

```
ERROR: jsonpath member accessor can only be applied to an object
```

Установим значение `true` для аргумента `silent`

```
SELECT jsonb_path_query('[]', 'strict $.a', silent => true);
```

Ошибка не отображается. Результат:

```
jsonb_path_query
------------------
(0 rows)
```

Операторы JSONPath

Оператор совпадения @@

Оператор `@@` возвращает результат проверки предиката JSONPath для указанного значения JSON. PostgreSQL учитывает только первый элемент результата. Если результат не `boolean`, `@@` возвращает `null`.

Следующий запрос определяет, содержит ли столбец `content` таблицы `book_orders` строки со значением `Hyperion`.

```
SELECT content @@ '$.** == "Hyperion"' AS hyperion FROM book_orders;
```

Результат:

```
hyperion
----------
true
null
null
```

Оператор проверки существования @?

Оператор `@?` определяет, возвращает ли выражение JSONPath какой-либо элемент для указанного значения JSON.

Следующий запрос использует `@?`, чтобы отобразить книги со значением `price` больше 25.

```
SELECT content @? '$.items[*] ? (@.price > 25)'
FROM book_orders;
```

Результат:

```
jsonb_path_query
------------------
"War and Peace"
"Ulysses"
```

Режимы strict и lax

Когда выполняется запросы над данными JSON выражение JSONPATH может соответствовать фактической структуре данных JSON. Попытка доступа к несуществующему элементу является ошибкой. По умолчанию выражение JSONPATH может для режимов обработки ошибок структуры.

- `lax` — запросом JSONPATH можно адаптирует запрашиваемые данные к указанному значению JSONPATH. Оставшаяся структурная ошибка подавляется и возвращается пустой результат по SQL/JSON. Этот режим используется по умолчанию.
- `strict` — если возникла ошибка структуры, происходит ошибка.

Для переключения режима используйте ключевое слово `lax` или `strict`.

Чтобы проиллюстрировать разницу между `strict` и `lax`, в значение JSON, которое не содержит этот ключ. В режиме `lax` результатом будет `false`

```
SELECT jsonb '{"a":2}' @* '$.b ? (@ > 1)';
```

Результат:

```
?column?
----------
false
```

В режиме `strict` возвращается `null`:

```
SELECT jsonb '{"a":2}' @* 'strict $.b ? (@ > 1)';
```

Результат:

```
?column?
----------
null
```

В режиме `lax` массив со сложной иерархией `[1,2,3,[4]]` обрабатывается как `[1,2,3,4]`

```
SELECT jsonb '[1,2,3,[4,5,6]]' @* 'lax $[4] > 0';
```

Результат: `true`

В режиме `strict` запрос вернет `false`

```
SELECT jsonb '[1,2,3,[4,5,6]]' @* 'strict $[4] ? (@ == 4)';
```

Значение `@ == @[4]`, чтобы получить результат `true` в режиме `strict`:

```
SELECT jsonb '[1,2,3,[4,5,6]]' @* 'strict $[4] ? (@[4] == 4)';
```

Для получения дополнительной информации о режимах обратитесь к статье Strict and lax modes.

Тип данных XML

Содержание

- Создание XML значений
  - Функция xmlparse
  - PostgreSQL преобразование
- Преобразование XML в строку символов
- Создание элементов XML
  - xmlconcat
  - xmlelement
  - xmlforest
  - xmlpi
  - xmlroot
  - xmlagg
- Проверка свойств XML значений
  - IS DOCUMENT / IS NOT DOCUMENT
  - xmlexists
  - xml_is_well_formed
- Получение XML значений
  - Получение XML значений
  - Получение XML из XML
- Экспорт таблиц в XML

Создание XML значений

Создание XML значений через `xmlparse` или приведение типа.

```
SELECT 'bar'::xml, xmlparse(DOCUMENT 'bar'), xmlparse(CONTENT 'bar');
```

Функция xmlparse

Функция `XMLPARSE` позволяет создать `XML` значение из символов. Значение определяется как `DOCUMENT` или `CONTENT`.

```
xmlparse ( { DOCUMENT | CONTENT } value )
```

Пример для данных содержащих `DOCTYPE`. Для данных без `DOCTYPE` значение `DOCUMENT` и `CONTENT` одинаково.

```
SELECT xml_is_well_formed('<!DOCTYPE t [<!ELEMENT t (#PCDATA)>]><t>bar</t>');

Результат:

 xml_is_well_formed
--------------------
 t

(1 строка)
```

```
SELECT xmlparse(DOCUMENT '<!DOCTYPE t [<!ELEMENT t (#PCDATA)>]><t>bar</t>');
SELECT xmlparse(CONTENT '<!DOCTYPE t [<!ELEMENT t (#PCDATA)>]><t>bar</t>');

Результат:

 xmlparse
----------
 t

(1 строка)

 xmlparse
----------
 t

(1 строка)
```

PostgreSQL преобразование

Преобразование значения символов в `XML` значение без использования функции `XMLPARSE`.

```
SELECT xml_is_well_formed('<!DOCTYPE t [<!ELEMENT t (#PCDATA)>]><t>bar</t>');

Результат:

 xml_is_well_formed
--------------------
 t

(1 строка)
```

```
SELECT 'bar'::xml;

Результат:

 xml
-----
 bar

(1 строка)
```

Преобразование XML в строку символов

Преобразование значения `XML` в значение символов.

```
SELECT xmlserialize(CONTENT 'bar'::xml AS text);
```

Функция `XMLSERIALIZE` преобразует значение `XML` в значение символов. Значение определяется как `DOCUMENT` или `CONTENT`. Тип данных символов может быть `character`, `character varying` или `text`.

```
xmlserialize ( { DOCUMENT | CONTENT } value AS type )
```

Пример:

```
SELECT xmlserialize(CONTENT 'bar'::xml AS text);

Результат:

 xmlserialize
--------------
 bar

(1 строка)
```

Создание элементов XML

В этой секции рассматриваются функции создающие элементы `XML`.

Функция `XMLCONCAT` создает `XML` значение из списка `XML` значений. Значение `NULL` игнорируется. Если все значения `NULL`, то результат `NULL`.

xmlconcat

Функция `XMLCONCAT` объединяет значения `XML` элементов.

```
xmlconcat ( value [, ...] )
```

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');

Результат:

  xmlconcat
--------------
 <abc/><bar>foo</bar>

(1 строка)
```

xmlelement

Функция `XMLELEMENT` создает `XML` элемент с заданным именем, атрибутами и значением.

```
xmlelement ( name name [, xmlattributes ( value [ AS attname ] [, ...] ) ] [, value [, ...] ] )
```

Где

- `name` - имя элемента
- `xmlattributes(value [ AS attname ] [, ...])` - задает атрибуты элемента. Имя атрибута `attname` берется из имени колонки, если оно не задано явно.
- `value [, ...]` - содержимое элемента

```
SELECT xmlelement(name test, xmlattributes(current_date as dt, 'abc' as "mi\"x"), 'bar & <baz/>');

Результат:

                    xmlelement
-----------------------------------------------
 <test dt="2016-06-22" mi="mi&quot;x">bar &amp; <baz/></test>

(1 строка)
```

Если значение имеет пустое значение, то элемент пустого значения не создается.

```
SELECT xmlelement(name test, xmlattributes(null as attr), 42, null, 'cat');

Результат:

          xmlelement
-------------------------------
 <test>42cat</test>

(1 строка)
```

xmlforest

Функция `XMLFOREST` создает последовательность `XML` элементов из заданных значений и названий.

```
xmlforest ( value [ AS name ] [, ...] )
```

Где `name` - имя элемента, `value` - значение элемента.

```
SELECT xmlforest(1 as one, 2.01 as two, 'three' as three, null as four);

Результат:

                 xmlforest
-------------------------------------------
 <one>1</one><two>2.01</two><three>three</three>

(1 строка)
```

Если имя элемента не задано явно, то оно берется из имени колонки. Если имя колонки не задано, то возникает ошибка.

```
SELECT xmlforest(1, 2.01, 'three', null);

Результат:

ERROR:  unnamed XML value
SQL state: 42846
Подсказка: XMLFOREST must have a name in each argument
```

xmlpi

Функция `XMLPI` создает `XML` инструкцию обработки.

```
xmlpi ( name name [, value ] )
```

Где `name` - имя инструкции, `value` - значение инструкции.

```
SELECT xmlpi(name php, 'echo "hello world";');

Результат:

            xmlpi
--------------------------------
 <?php echo "hello world";?>

(1 строка)
```

xmlroot

Функция `XMLROOT` изменяет свойства `XML` значения.

```
xmlroot ( xml, version { text | NO VALUE } [, STANDALONE { YES | NO | NO VALUE } ] )
```

Где `xml` - значение, `version` - версия, `STANDALONE` - значение `YES`, `NO` или `NO VALUE`.

```
SELECT xmlroot('<?xml version="1.1" standalone="yes"?><t>bar</t>', version '1.0', standalone yes);

Результат:

                    xmlroot
-----------------------------------------------
 <?xml version="1.0" standalone="yes"?><t>bar</t>

(1 строка)
```

xmlagg

Функция `XMLAGG` агрегирует `XML` значения.

```
xmlagg ( xml )
```

```
SELECT xmlagg(x) FROM (SELECT '<foo>bar</foo>'::xml AS x UNION ALL SELECT '<bar>foo</bar>'::xml) AS t;

Результат:

          xmlagg
-----------------------------
 <foo>bar</foo><bar>foo</bar>

(1 строка)
```

Проверка свойств XML значений

Функции `IS DOCUMENT` и `IS NOT DOCUMENT`.

IS DOCUMENT / IS NOT DOCUMENT

Функция `IS DOCUMENT` проверяет является ли `XML` значение документом. Функция `IS NOT DOCUMENT` проверяет является ли `XML` значение не документом.

```
xml IS DOCUMENT
xml IS NOT DOCUMENT
```

Если `XML` значение содержит `DOCTYPE` или `XML` узел, то `IS DOCUMENT` возвращает `true`. Если значение не содержит `DOCTYPE` или `XML` узел, то `IS DOCUMENT` возвращает `false`. Если значение не `XML`, то возникает ошибка.

```
SELECT '<foo>bar</foo>'::xml IS DOCUMENT;

Результат:

 ?column?
----------
 t

(1 строка)
```

```
SELECT 'bar'::xml IS NOT DOCUMENT;

Результат:

 ?column?
----------
 f

(1 строка)
```

xmlexists

Функция `XMLEXISTS` проверяет содержит ли `XML` значение заданный `XPath`. Значение `XPath` должно быть константой. Функция возвращает `true` если `XML` значение содержит `XPath`, иначе `false`. Если любой из аргументов `NULL`, то результат `NULL`.

```
xmlexists ( text PASSING [ BY { REF | VALUE } ] xml [ BY { REF | VALUE } ] )
```

```
SELECT xmlexists('//town[text() = ''Toronto'']' PASSING BY VALUE '<favorites><town>Toronto</town><town>Ottawa</town></favorites>');

Результат:

 xmlexists
-----------
 t

(1 строка)
```

xml_is_well_formed

Эта функция возвращает значение `true` если `XML` значение правильно сформировано и возвращает значение `false` иначе.

```
xml_is_well_formed ( text )
```

Функция `xml_is_well_formed` возвращает `true` если значение правильно сформировано для `DOCUMENT`, `xml_is_well_formed_document` возвращает `true` если значение правильно сформировано для `DOCUMENT`, `xml_is_well_formed_content` возвращает `true` если значение правильно сформировано для `CONTENT`. Функция `xml_is_well_formed` эквивалентна `xml_is_well_formed_document`.

```
SELECT xml_is_well_formed('<foo>bar</foo>');

Результат:

 xml_is_well_formed
--------------------
 t

(1 строка)
```

Получение XML значений

Функция `XPATH` позволяет получить `XML` значение по заданному `XPath`. Функция возвращает значение `XML[]`. Функция `XPATH_EXISTS` возвращает `true` если `XPath` существует, иначе `false`.

```
xpath ( xpath text, xml xml [, nsarray text[] ] )
xpath_exists ( xpath text, xml xml [, nsarray text[] ] )
```

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>', ARRAY[ARRAY['my', 'http://example.com']]);

Результат:

 xpath
--------
 {test}

(1 строка)
```

```
SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>', ARRAY[ARRAY['my', 'http://example.com']]);

Результат:

 xpath_exists
--------------
 t

(1 строка)
```

Получение XML из XML

Функция `XMLTABLE` позволяет получить `XML` значение из `XML` значения. Функция возвращает `SETOF record`.

```
XMLTABLE ( [ XMLNAMESPACES ( namespace_uri AS namespace_name [, ...] ) ,
row_expression PASSING [ BY { REF | VALUE } ] document_expression [ BY { REF | VALUE } ]
COLUMNS name { type [ PATH column_expression ] [ DEFAULT default_expression ] [ NOT NULL | NULL ] | FOR ORDINALITY } [, ...] )
```

```
SELECT * FROM XMLTABLE('/root/a' PASSING '<root><a>1</a><a>2</a></root>' COLUMNS x int);

Результат:

 x
---
 1
 2

(2 строки)
```

Экспорт таблиц в XML

Генерация XML из строк таблицы `XML` (в основе используется `xmlforest`).

```
TABLE_TO_XML ( tbl regclass, nulls boolean, tableforest boolean, targetns text )
QUERY_TO_XML ( query text, nulls boolean, tableforest boolean, targetns text )
CURSOR_TO_XML ( cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text )
```

Где `TABLE_TO_XML` - имя таблицы, `QUERY_TO_XML` - запрос, `CURSOR_TO_XML` - курсор, `nulls` - `true` если `NULL` значения должны быть включены, `tableforest` - `true` если `XML` должен быть создан как последовательность `xmlforest`, `targetns` - пространство имен, `count` - количество строк.

Функции `TABLE_TO_XML` имеют варианты `TABLE_TO_XMLSCHEMA` и `TABLE_TO_XML_AND_XMLSCHEMA` которые возвращают `XML` схему данных.

Функции `QUERY_TO_XML` имеют варианты `QUERY_TO_XMLSCHEMA` и `QUERY_TO_XML_AND_XMLSCHEMA` которые возвращают `XML` схему данных.

Функции `CURSOR_TO_XML` имеют варианты `CURSOR_TO_XMLSCHEMA` и `CURSOR_TO_XML_AND_XMLSCHEMA` которые возвращают `XML` схему данных.

Функции `DATABASE_TO_XML` и `DATABASE_TO_XMLSCHEMA` экспортируют базу данных.

Функции `SCHEMA_TO_XML` и `SCHEMA_TO_XMLSCHEMA` экспортируют схему.

Функции `DATABASE_TO_XML_AND_XMLSCHEMA` и `SCHEMA_TO_XML_AND_XMLSCHEMA` экспортируют базу данных и схему.

Функции `TABLE_TO_XML` имеют варианты `TABLE_TO_XML_AND_XMLSCHEMA` которые возвращают `XML` схему данных.

Функции `QUERY_TO_XML` имеют варианты `QUERY_TO_XML_AND_XMLSCHEMA` которые возвращают `XML` схему данных.

Функции `CURSOR_TO_XML` имеют варианты `CURSOR_TO_XML_AND_XMLSCHEMA` которые возвращают `XML` схему данных.

Функции `DATABASE_TO_XML` и `DATABASE_TO_XML_AND_XMLSCHEMA` экспортируют базу данных.

Функции `SCHEMA_TO_XML` и `SCHEMA_TO_XML_AND

Тип данных hstore

Egor Rogov

Содержание

Назад к содержанию

Операции со значениями hstore
- Получение значения по ключу
- Добавление пары ключ/значение к существующему значению hstore
- Изменение пары ключ/значение
- Добавление нескольких пар ключ/значение
- Удаление пары ключ/значение
- Проверка существования ключа
- Проверка существования пары ключ/значение
- Получение строк, содержащих указанные ключи
- Получение всех ключей
- Получение всех значений
- Преобразование данных hstore в набор записей
- Преобразование данных hstore в JSON/JSONB

Индексирование полей hstore

В PostgreSQL тип данных `hstore` позволяет хранить пары ключ/значение в одном значении поля. Тип `hstore` использовался, когда колонки не могут быть определены заранее или отличаются друг от друга.

Тип данных `hstore` реализуется модулем hstore. Чтобы использовать этот тип, нужно создать расширение `hstore`:

```
CREATE EXTENSION hstore;
```

Значение `hstore` содержит пары `ключ => значение`, разделенные запятыми. Ключи и значения хранятся в виде текстовых строк.

Создадим таблицу `products` по отношению `hstore`, чтобы посмотреть, как работать с этим типом:

```
CREATE TABLE products (
  id serial PRIMARY KEY,
  name text NOT NULL UNIQUE,
  attributes hstore
);
```

В этой таблице разные продукты могут иметь разные атрибуты. Добавим в таблицу следующие строки:

```
INSERT INTO products (name, attributes) VALUES (
  'The Time Machine',
  'author => "Herbert Wells",
  pages => 202,
  genre => "sci-fi"'),
('Hyperion',
  'author => "Dan Simmons",
  pages => 482,
  genre => "sci-fi"'),
('The Great Gatsby',
  'author => "F. Scott Fitzgerald",
  pages => 254,
  genre => "novel"'),
('Pericles',
  'publisher => "Time Inc",
  number => 32,
  date => "08.12.2022"'
);
```

Содержимое таблицы:

```
SELECT * FROM products;
```

```
 id |      name      |                                attributes
----+----------------+-------------------------------------------------------------------
  1 | The Time Machine | "genre"=>"sci-fi", "pages"=>"202", "author"=>"Herbert Wells"
  2 | Hyperion         | "genre"=>"sci-fi", "pages"=>"482", "author"=>"Dan Simmons"
  3 | The Great Gatsby | "genre"=>"novel", "pages"=>"254", "author"=>"F. Scott Fitzgerald"
  4 | Pericles         | "date"=>"08.12.2022", "number"=>"32", "publisher"=>"Time Inc"
```

Тип `hstore` имеет следующие особенности синтаксиса:

- Порядок пар `hstore` не сохраняется и может быть не воспроизведен на выводе.
- PostgreSQL игнорирует пробелы между парами или вокруг знака `=>`. Ключи и значения, содержащие пробелы, запятые, знаки `=>` или `"`, должны быть заключены в двойные кавычки. Символы двойных кавычек и обратной косой черты (`"` и `\`) в ключах или значениях, экранируются во обратной косой черты (`\"` и `\\`).
- Каждый ключ должен быть уникальным.
- Значения может содержать `NULL` - `key => NULL`. Ключевое слово `NULL` не зависит от регистра. Заключите его в двойные кавычки, чтобы обрабатывать как строковое значение ("NULL").

Операции со значениями hstore

PostgreSQL предоставляет полный комплект операторов и функций для работы с типом `hstore`. В этом разделе приведены примеры типовых операций, которые можно воспользоваться командами этого функционала. Для получения дополнительной информации об операторах и функциях `hstore` обратитесь к документации PostgreSQL: https://postgrespro.ru/docs/.

Получение значения по ключу

Вы можете использовать оператор `->`, чтобы получить значение определенного ключа из столбца `hstore`. Если ключ отсутствует, соответствующее значение:

```
SELECT name, attributes-> 'author' as author
FROM products;
```

Результат:

```
      name      |      author
----------------+-------------------
 The Time Machine | Herbert Wells
 Hyperion         | Dan Simmons
 The Great Gatsby | F. Scott Fitzgerald
```

Вы можете использовать оператор `->` и выражение `WHERE`. Следующий запрос выберет продукты, для которых указан жанр `sci-fi`:

```
SELECT name, attributes-> 'author' as author
FROM products
WHERE attributes -> 'genre' = 'sci-fi';
```

Результат:

```
      name      |      author
----------------+-------------------
 The Time Machine | Herbert Wells
 Hyperion         | Dan Simmons
```

Добавление пары ключ/значение к существующему значению hstore

Вы можете добавить новую пару ключ/значение в существующее строки. Для этого используйте оператор `||`, который объединяет два значения `hstore`. Следующий запрос добавляет `"freeshipping"=>"yes"` ко всем существующим строкам:

```
UPDATE products
SET attributes = attributes || '"freeshipping"=>"yes"'::hstore;
```

Изменение пары ключ/значение

Чтобы обновить существующую пару ключ/значение, используйте тот же оператор `||`. Следующий запрос изменяет значение `freeshipping` ключа `Hyperion` на `no`:

```
UPDATE products
SET attributes = attributes || '"freeshipping"=>"no"'::hstore
WHERE name = 'Hyperion';
```

Содержимое таблицы:

```
SELECT * FROM products;
```

```
 id |      name      |                                                attributes
----+----------------+-------------------------------------------------------------------------------------------
  1 | The Time Machine | "genre"=>"sci-fi", "pages"=>"202", "author"=>"Herbert Wells", "freeshipping"=>"yes"
  2 | Hyperion         | "genre"=>"sci-fi", "pages"=>"482", "author"=>"Dan Simmons", "freeshipping"=>"no"
  3 | The Great Gatsby | "genre"=>"novel", "pages"=>"254", "author"=>"F. Scott Fitzgerald", "freeshipping"=>"yes"
  4 | Pericles         | "date"=>"08.12.2022", "number"=>"32", "publisher"=>"Time Inc", "freeshipping"=>"yes"
```

Добавление нескольких пар ключ/значение

Чтобы добавить несколько пар ключ/значение, используйте двоичный массив, содержащий ключи и значения.
Следующее выражение изменяет значение поля `attributes` в строке, где `id = 4`:

```
UPDATE products SET attributes = hstore(ARRAY['pages', '32'],['media', 'ebook']) WHERE id = 4;
```

Запрос меняет добавляет новые значения в значение поля `attributes` в строке, где `id = 4`:

```
UPDATE products SET attributes = attributes || hstore(ARRAY['pages', '32'],['media', 'ebook']) WHERE id = 4;
```

Удаление пары ключ/значение

Используйте функцию `delete`, чтобы удалить пары ключ/значение из столбца `hstore`.

```
UPDATE products
SET attributes = delete(attributes, 'freeshipping');
```

Вы также можете использовать массив для удаления нескольких ключей:

```
UPDATE products
SET attributes = delete(attributes, ARRAY['genre', 'pages']);
```

Проверка существования ключа

Используйте оператор `?`, чтобы проверить, существует ли определенный ключ в столбце `hstore`:

```
SELECT name, attributes -> 'publisher' as publisher
FROM products
WHERE attributes ? 'publisher';
```

Результат:

```
   name    | publisher
-----------+-----------
 People    | Time Inc
```

Проверка существования пары ключ/значение

Чтобы проверить, существует ли пара ключ/значение, используйте оператор `@>`, который определяет, содержит ли левый операнд правый. Следующее выражение возвращает все строки, в которых столбец `attributes` содержит `"freeshipping"=>"yes"`:

```
SELECT name, attributes -> 'freeshipping' as freeshipping
FROM products
WHERE attributes @> '"freeshipping"=>"yes"' :: hstore;
```

Результат:

```
      name      | freeshipping
----------------+--------------
 The Time Machine | yes
 The Great Gatsby | yes
 People           | yes
```

Получение строк, содержащих указанные ключи

Используйте оператор `?&`, чтобы получить строки с несколькими указанными ключами. Следующий запрос возвращает строки с ключами `genre` и `pages`:

```
SELECT name FROM products
WHERE attributes ?& ARRAY [ 'genre', 'pages' ];
```

Результат:

```
      name
----------------
 The Time Machine
 The Great Gatsby
 Hyperion
```

Получение всех ключей

Используйте функцию `akeys`, чтобы получить все ключи `hstore` в виде массива:

```
SELECT akeys (attributes) FROM products;
```

Результат:

```
                akeys
-------------------------------------
 {genre,pages,author,freeshipping}
 {genre,pages,author,freeshipping}
 {genre,pages,author,freeshipping}
 {date,number,publisher,freeshipping}
```

Функция `skeys` возвращает все ключи в виде набора записей:

```
SELECT skeys(attributes) FROM products;
```

Результат:

```
 skeys
-------
 genre
 pages
 author
 freeshipping
 genre
 pages
 author
 freeshipping
 genre
 pages
 author
 freeshipping
 date
 number
 publisher
 freeshipping
 genre
 pages
 author
 freeshipping
```

Получение всех значений

Используйте функцию `avals`, чтобы получить все значения `hstore` в виде массива:

```
SELECT avals (attributes) FROM products;
```

Результат:

```
                avals
-------------------------------------
 {sci-fi,202,"Herbert Wells",yes}
 {novel,254,"F. Scott Fitzgerald",yes}
 {08.12.2022,32,"Time Inc",yes}
 {sci-fi,446,"Dan Simmons",no}
```

Функция `svals` возвращает все значения в виде набора записей:

```
SELECT svals (attributes) FROM products;
```

Результат:

```
 svals
-------
 sci-fi
 202
 Herbert Wells
 yes
 novel
 254
 F. Scott Fitzgerald
 yes
 08.12.2022
 32
 Time Inc
 yes
 sci-fi
 446
 Dan Simmons
 no
```

Преобразование данных hstore в набор записей

Используйте функцию `each` для извлечения данных `hstore` в виде набора записей:

```
SELECT
    name,
    (each(attributes)).*
FROM products;
```

Результат:

```
      name      |    key    |      value
----------------+-----------+-----------------
 The Time Machine | genre     | sci-fi
 The Time Machine | pages     | 202
 The Time Machine | author    | Herbert Wells
 The Time Machine | freeshipping | yes
 The Great Gatsby | genre     | novel
 The Great Gatsby | pages     | 254
 The Great Gatsby | author    | F. Scott Fitzgerald
 The Great Gatsby | freeshipping | yes
 People           | date      | 08.12.2022
 People           | number    | 32
 People           | publisher | Time Inc
 People           | freeshipping | yes
 Hyperion         | genre     | sci-fi
 Hyperion         | pages     | 446
 Hyperion         | author    | Dan Simmons
 Hyperion         | freeshipping | no
```

Преобразование данных hstore в JSON/JSONB

В PostgreSQL есть функции `hstore_to_json` и `hstore_to_jsonb` для преобразования данных `hstore` в JSON или JSONB. Следующие запросы преобразуют столбец `attributes` в `JSONB`:

```
SELECT hstore_to_jsonb (attributes) jsonb
FROM products;
```

Результат:

```
                                                jsonb
-------------------------------------------------------------------------------------------------------
 {"genre": "sci-fi", "pages": "202", "author": "Herbert Wells", "freeshipping": "yes"}
 {"genre": "novel", "pages": "254", "author": "F. Scott Fitzgerald", "freeshipping": "yes"}
 {"date": "08.12.2022", "number": "32", "publisher": "Time Inc", "freeshipping": "yes"}
 {"genre": "sci-fi", "pages": "446", "author": "Dan Simmons", "freeshipping": "no"}
```

Индексирование полей hstore

Вы можете использовать индексы, чтобы сократить время, необходимое PostgreSQL для поиска ключей и значений.

Тип `hstore` поддерживает GIN и GiST индексы с операторами `@>`, `?`, `?&` и `?|`.

Следующие операторы создают индексы `BTREE` и `HASH` для столбца `attributes`:

```
CREATE INDEX idx_gin_attributes ON products USING GIN(attributes);
```

```
CREATE INDEX idx_gist_attributes ON products USING GIST(attributes);
```

Вы также можете использовать индексацию `hash` и `btree` в операторе `=`.

Следующие выражения создают индексы `hash` и `btree` для столбца `attributes`:

```
CREATE INDEX idx_btree_attributes ON products USING BTREE(attributes);
```

```
CREATE INDEX idx_hash_attributes ON products USING HASH(attributes);
```

Тип данных bytea

Евгения Кузина

AD.PG

Содержание

Назад к содержанию

Формат bytea hex

Формат bytea escape

Функции и операторы для работы с двоичными строками

- SQL-функции и операторы
- Функции для преобразования текстовых/двоичных строк

Тип данных `bytea` позволяет хранить двоичные строки переменной длины. Двоичная строка представляет собой последовательность октетов (байтов). Двоичные строки отличаются от строк символов следующим:

- Двоичные строки позволяют хранить октеты нулевых значений и другие "непечатаемые" октеты. Строки символов не допускают нулевых октетов и любых других октетов, не соответствующих кодировке базы данных.
- Операции с двоичными строками обрабатывают байты, обработка символьных строк зависит от настроек языковых стандартов.

Двоичные строки предназначены для операций с "сырыми байтами" ("raw bytes"), строки символов — для работы с текстом.

Тип данных `bytea` добавляет 1 или 4 байта к размеру двоичной строки.

`Bytea` поддерживает два формата ввода и вывода `hex` и исторический формат PostgreSQL `escape`. Входящие данные могут быть в одном из этих форматов. Выходной формат определяет параметр конфигурации bytea_output, его значение по умолчанию — `hex`.

Формат bytea hex

Формат `hex` кодирует двоичные данные двумя шестнадцатеричными цифрами на байт. Он имеет следующий формат:

```
'\x<binary string>'
```

Пример:

```
SELECT '\xDEADBEEF';
```

Шестнадцатеричные цифры могут быть в верхнем или нижнем регистре, пробелы разрешены между парами цифр и запрещены внутри пары цифр и в начале последовательности `\x`. Формат `hex` совместим с широким спектром приложений и протоколов, и его преобразование происходит быстрее, чем формата `escape`.

Формат bytea escape

Формат `escape` — традиционный формат PostgreSQL для типа `bytea`. Он представляет двоичную строку как последовательность символов ASCII и преобразует байты, которые не могут быть представлены в виде символов ASCII, в спецпоследовательности.

Чтобы заэкранировать октет, преобразуйте его в трехзначное восьмеричное значение и поставьте перед ним обратную косую черту (`\`). Чтобы добавить символ обратной косой черты к двоичной строке, используйте двойную обратную косую черту (`\\`). В то время как любое значение октета может быть заэкранировано, октеты из таблицы ниже должны быть заэкранированы.

Десятичное значение байта	Описание	Входящая спецпоследовательность	Пример	Шестнадцатеричное представление
0	Нулевой байт	`'\000'`	`'\000'::bytea`	\x00
39	Одиночная кавычка	`''` или `'\047'`	`''::bytea`	\x27
92	Обратная косая черта	`'\\'` или `'\134'`	`'\\'::bytea`	\x5c
от 0 до 31 и от 127 до 255	"Непечатаемые" октеты	`'\xxx'` (восьмеричное значение)	`'\001'::bytea`	\x01

Требования экранирования "непечатаемых" символов определяются настройками языковых стандартов.

Октеты `bytea` по умолчанию выводятся в шестнадцатеричном формате.

Если вы измените `bytea_output` на `escape`, PostgreSQL преобразует "непечатаемые" октеты (десятичные значения октетов от 0 до 31 и от 127 до 255) в эквивалентные трехзначные восьмеричные значения. Печатные октеты выводятся в виде стандартного представления в кодировке клиента. Обратная косая черта при выводе дублируется.

Пример:

```
SET bytea_output = 'escape';

SELECT 'abc \\141\145\146 \052 \251 \134'::bytea;
```

Результат:

```
      bytea
------------------
abc def * \251 \\
```

РЕКОМЕНДАЦИЯ

Формат `escape` стирает различия между двоичными строками и символьными строками и имеет громоздкий механизм экранирования спецпоследовательностей. Используйте формат `hex` вместо `escape`, если это возможно.

Функции и операторы для работы с двоичными строками

PostgreSQL предоставляет множество функций и операторов для работы с двоичными строками. Многие из них эквивалентны функциям для работы с текстовыми строками. В этом разделе приведены несколько примеров. За дополнительной информацией обратитесь к документации PostgreSQL: Binary string functions and operators.

SQL-функции и операторы

Ниже приведены некоторые SQL-операторы и функции, которые можно использовать для работы с двоичными строками:

- Две двоичные строки можно объединить оператором `||`.

```
SELECT '\x123456'::bytea || '\x789a00bcde'::bytea;
```

  Результат: `\x123456789a00bcde`.

- Функция `position` возвращает начальный индекс первой найденной подстроки в байтах или ноль, если подстрока не найдена:

```
SELECT position('\x5678'::bytea in '\x1234567890'::bytea);
```

  Результат: `3`.

- Функция `substring` извлекает подстроку, которая начинается с указанного байта и имеет указанную длину в байтах. Функция имеет следующий синтаксис:

```
substring ( bytes bytea [ FROM start integer ] [ FOR count integer ] ) → bytea
```

  Пример:

```
SELECT substring('\x1234567890'::bytea from 3 for 2);
```

  Результат: `\x5678`.

Функции для преобразования текстовых/двоичных строк

PostgreSQL позволяет конвертировать строки между различными кодировками и преобразовывать двоичные данные в текстовую форму. Доступные преобразования перечислены в статье Available character set conversions. Вы можете использовать следующие функции для преобразования:

- `convert` преобразует двоичную строку, содержащую текст в исходной кодировке, в двоичную строку в результирующей кодировке.

```
convert ( <двоичная_строка> bytea, <исходная_кодировка>, <результирующая_кодировка> ) → bytea
```

  Пример:

```
SELECT convert('text_in_utf8', 'UTF8', 'LATIN1');
```

  Результат: `\x746578745f696e5f75746638`.

- `convert_from` преобразует двоичную строку, содержащую текст в исходной кодировке, в текст в кодировке базы данных.

```
convert_from ( <двоичная_строка> bytea, <исходная_кодировка> ) → text
```

  Пример:

```
SELECT convert_from('\x746578745f696e5f75746638', 'UTF8');
```

  Результат: `text_in_utf8`.

- `convert_to` преобразует текстовую строку в кодировке базы данных в двоичную строку в результирующей кодировке.

```
convert_to ( <текстовая_строка> text, <результирующая_кодировка> ) → bytea
```

  Пример:

```
SELECT convert_to('some_text', 'UTF8');
```

  Результат: `\x736f6d655f74657874`.

- `encode` кодирует двоичные данные в текстовое представление. Поддерживаемые форматы: base64, `escape`, `hex`.

```
encode ( <двоичная_строка> bytea, <формат> text ) → text
```

  Пример:

```
SELECT encode('\x41445047', 'base64');
```

  Результат: `QURQRw==`.

- `decode` декодирует двоичные данные из текстовой строки. Поддерживаемые форматы: base64, `escape`, `hex`.

```
decode ( <текстовая_строка> text, <формат> text ) → bytea
```

  Пример:

```
SELECT decode('QURQRw==', 'base64');
```

  Результат: `\x41445047`.

Содержание

Назад к содержанию

- Форматы tsvector и tsquery
  - tsvector
  - tsquery
- Операторы соответствия @@
- Увеличение скорости поиска
- Ранжирование результатов поиска

PostgreSQL поддерживает возможность полнотекстовых поисков, реализованных с помощью операторов сравнения типа tsvector (документ) и tsquery (запрос). Документ может быть простой текстовой строкой, но для эффективного поиска по ней возможности не ограничиваются.

Поиск по шаблону, который с самого начала стандарта SQL не отличал современных требований информационных систем. В PostgreSQL он поддерживается операторами ~, ~*, LIKE и ILIKE. Поиск по шаблону не учитывает словоформы, не может искать "соответствовать", слово "соответствующий" не будет включено в результат. Не учитывает синонимы, не может искать "соответствует", когда задано слово "соответствовать". Не учитывает порядок слов, не может искать "соответствует" когда задано "соответствовать". Не учитывает важность слов, не может выдать наиболее важные документы первыми. Не учитывает удобство чтения, не может выделить найденные слова в документе. Не учитывает морфологию языка, не может искать "соответствовать", когда задано "соответствует". Не учитывает грамматику языка, не может искать "соответствует", когда задано

Анализ запросов

Д. Евгений Кухарев

Содержание

Назад к содержанию

Использование кэша
Использование индексов

PostgreSQL создает план выполнения для каждого запроса. Поскольку результирующий план имеет решающее значение для производительности, в PostgreSQL имеется планировщик, оптимизирующий планы запросов, используя имеющуюся статистику каждой из доступных для выполнения команд. Вы можете использовать команду `EXPLAIN`, чтобы увидеть, какой план создает планировщик для запроса.

Создадим таблицу для последующих тестовых примеров

```
CREATE TABLE test_table (field1 integer, field2 text);
```

Добавим данные в таблицу и выполним команду `ANALYZE`, чтобы создать статистику таблицы после добавления данных:

```
INSERT INTO test_table
  SELECT g, md5(random()::text)
  FROM generate_series(1, 1000000) AS g;
ANALYZE test_table;
```

Выполним команду `EXPLAIN`, чтобы посмотреть план запроса:

```
EXPLAIN SELECT * FROM test_table;
```

Результат:

```
                          QUERY PLAN
---------------------------------------------------------------
Seq Scan on test_table  (cost=0.00..18334.00 rows=1000000 width=37)
```

PostgreSQL может считывать данные из таблицы несколькими способами. `Seq Scan` означает, что PostgreSQL считывает данные последовательно, блок за блоком. Параметр `cost` содержит два значения: предполагаемые начальные и общие затраты на исполнение запроса. Начальные затраты имеют значение `0.00`: это затраты до начала вывода, например, затраты на сортировку в процессе исполнения. Предполагаемые общие затраты (`18334.00`) показывают затраты, основанные для извлечения всех строк. Поскольку `cost` не учитывает затраты на отправку результатов клиенту, поскольку планировщик не может на них повлиять. Тем не менее, большие значения `cost` отражают стоимость последовательного доступа к страницам (`seq_page_cost`). Величина `cost` чаще всего говорит о потребляемом ресурсе, и из алгоритмов важнее, если эти величины корректно имеют собой. Обратитесь к статье Planner Cost Constants за дополнительными сведениями.

Параметр `rows` оказывается предполагаемым количеством строк, которое возвращает `Seq Scan`.

Параметр `width` представляет собой предполагаемый средний размер строки в байтах.

Когда вы выполняете `EXPLAIN`, планировщик использует `cost` запроса, вместо этого он создает предполагаемый план выполнения на основе доступной статистики. Фактический план может отличаться. Используйте параметр `ANALYZE`, чтобы выполнить запрос и получить актуальную статистику исполнения.

```
EXPLAIN (ANALYZE) SELECT * FROM test_table;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Seq Scan on test_table  (cost=0.00..18334.00 rows=1000000 width=37) (actual time=0.005..94.291 rows=1000000 loops=1)
Planning Time: 0.054 ms
Execution Time: 96.757 ms
```

Параметр `actual time` — время, затраченное на получение первой строки и всех строк, в миллисекундах.

Параметр `rows` — фактическое количество строк, возвращенное `Seq Scan`.

Параметр `loops` показывает, сколько раз PostgreSQL полностью отсканировал таблицу, т.е. выполнил `Seq Scan` над всеми данными таблицы.

`Execution Time` - общее время выполнения запроса.

Команда `EXPLAIN ANALYZE` выполняет запросы. Если вам нужно получить статистику по модифицирующим запросам `INSERT`, `DELETE` или `UPDATE` и не изменять данные, используйте команды `BEGIN` и `ROLLBACK`, чтобы выполнить `EXPLAIN ANALYZE` внутри блока транзакции с последующим откатом:

```
BEGIN;

EXPLAIN (ANALYZE) INSERT INTO test_table
  SELECT g, md5(random()::text)
  FROM generate_series(1, 100) AS g;

ROLLBACK;
```

Использование кеша

Команда `EXPLAIN` позволяет получить информацию об использовании кеша. Для этой цели нужно указать в запросе параметр `BUFFERS`.

Чтобы получить корректные результаты, перезапустите ADPG для очистки буферного кеша перед тестированием `BUFFERS`.

Выполним команду `EXPLAIN` с параметрами `ANALYZE` и `BUFFERS`:

```
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM test_table;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Seq Scan on test_table  (cost=0.00..18334.00 rows=1000000 width=37) (actual time=1.498..424.291 rows=1000000 loops=1)
  Buffers: shared read=6334
Planning:
  Buffers: shared hit=47 read=14
Planning Time: 35.535 ms
Execution Time: 663.332 ms
```

PostgreSQL читает таблицу частями, называемыми блоками. Параметр `Buffers: shared read` показывает количество блоков, считанных с диска. Таблица полностью считана с диска, считано 8334 блока. Так как ею ещё пуст.

Выполним этот же запрос:

```
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM test_table;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Seq Scan on test_table  (cost=0.00..18334.00 rows=1000000 width=37) (actual time=0.037..75.519 rows=1000000 loops=1)
  Buffers: shared hit=6334
  Rows Removed by Filter: 750
Planning Time: 0.033 ms
Execution Time: 108.073 ms
```

PostgreSQL добавил в результату параметр `Buffers: shared hit`. `Buffers: shared hit` - это количество блоков, считанных из кеша PostgreSQL. Если вы повторите запрос несколько раз, то увидите, что PostgreSQL использует только кеш. Количество блоков выполнения будет быть все больше и больше данных из кеша. PostgreSQL также добавляет данные в кеш при каждом запросе. Так как чтение из кеша быстрее, чем с диска, значение `Buffers: shared hit` увеличивается, а `Execution Time` сокращается при каждом выполнении запроса.

Конфигурационный параметр `shared_buffers` определяет размер кеша. Вы можете изменить его через ALTER SYSTEM. За дополнительной информацией обратитесь к статье Конфигурационные параметры.

Использование индексов

Добавим в запрос выражение `WHERE`:

```
EXPLAIN SELECT * FROM test_table WHERE field1 > 750;
```

Результат:

```
                          QUERY PLAN
---------------------------------------------------------------
Seq Scan on test_table  (cost=0.00..20854.00 rows=999288 width=37)
  Filter: (field1 > 750)
```

Таблица не имеет индексов, и PostgreSQL считывает каждую запись последовательно (`Seq Scan`).

`Filter: (field1 > 750)` означает, что к каждой записи применяется условие фильтрации `field1 > 750`. Если значение `field1` отвечает условию, PostgreSQL добавляет соответствующую строку в результат.

Значение `cost` увеличилось. Предполагаемое количество строк в результате запроса (`rows`) уменьшилось из-за выражения `WHERE`.

Создадим индекс для `field1` и выполним тот же запрос:

```
CREATE INDEX ON test_table(field1);
EXPLAIN (ANALYZE) SELECT * FROM test_table WHERE field1 > 750;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Seq Scan on test_table  (cost=0.00..20854.00 rows=999288 width=37) (actual time=0.133..89.711 rows=999250 loops=1)
   Filter: (field1 > 750)
   Rows Removed by Filter: 750
 Planning Time: 0.083 ms
 Execution Time: 121.254 ms
```

Было отфильтровано только 750 строк, запрос вернул более 99.9% таблицы. Несмотря на созданный индекс, планировщик выбрал `Seq Scan`.

С помощью опции `enable_seqscan` запретим `Seq Scan`. Планировщик запросов будет вынужден использовать индекс.

```
SET enable_seqscan TO off;
EXPLAIN (ANALYZE) SELECT * FROM test_table WHERE field1 > 750;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Index Scan using test_table_field1_idx on test_table  (cost=0.42..36794.81 rows=999288 width=37) (actual time=0.028..144.
  Index Cond: (field1 > 750)
Planning Time: 0.097 ms
Execution Time: 179.522 ms
```

`Index Scan` показывает, что PostgreSQL использует индекс. `Index Cond` содержит условие для поиска строк.

PostgreSQL использовал индекс, из-это уменьшило время выполнения. В предыдущем примере планировщик выбрал лучший план без индекса.

Включим опцию `enable_seqscan` и изменим запрос следующим образом:

```
SET enable_seqscan TO on;
EXPLAIN (ANALYZE) SELECT * FROM test_table WHERE field1 > 750;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Index Scan using test_table_field1_idx on test_table  (cost=0.42..36794.81 rows=999288 width=37) (actual time=0.028..144.
   Index Cond: (field1 > 750)
 Planning Time: 0.097 ms
 Execution Time: 179.522 ms
```

Индекс `test_table_field1_idx` был использован.

Добавим индекс для `field2`. Поскольку наша база данных хранит текстовые значения в кодировке UTF8, создадим индекс с помощью класса операторов `text_pattern_ops`:

```
CREATE INDEX ON test_table(field2 text_pattern_ops);
```

Добавим в запрос условие для текстового поля:

```
EXPLAIN (ANALYZE) SELECT * FROM test_table WHERE field2 LIKE 'ok';
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on test_table  (cost=7854.17..18711.87 rows=7874 width=37) (actual time=7.672..25.426 rows=42382 loops=1)
   Filter: (field2 ~~ 'ok'::text)
   Heap Blocks: exact=6332
   ->  Bitmap Index Scan on test_table_field2_idx1  (cost=0.00..7853.97 rows=43924 width=0) (actual time=6.548..6.548 rows=42382 loops=1)
         Index Cond: ((field2 >=~ 'o'::text) AND (field2 <~< 'p'::text))
 Planning Time: 0.097 ms
 Execution Time: 28.528 ms
```

В выводе команды `EXPLAIN` вижу `ON` и отступы отражают иерархическую структуру дерева плана.

`Bitmap Index Scan` - сначала битовых карт с позициями записей, соответствующих условию. Далее PostgreSQL в рамках оператора `Bitmap Heap Scan` последовательно сканирует целевую таблицу по этим позициям.

`Bitmap Heap Scan` эффективно работает с большим количеством строк, попадающих под условие.

PostgreSQL также поддерживает операцию `Index Only Scan`, которая используется, если запрос ссылается только на индексированные столбцы. Выполним запрос, который возвращает одно индексированное поле:

```
EXPLAIN SELECT field1 FROM test_table WHERE field1 < 750;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Index Only Scan using test_table_field1_idx on test_table
  (cost=0.42..29.69 rows=750 width=4)
  Index Cond: (field1 < 750)
```

`Index Only Scan` работает быстрее, чем `Index Scan`, так как `Index Only Scan` не требует чтения всей строки `width=4`, за исключением случая проверки видимости индексной записи.

Индексы повышают производительность различных операций, включая сортировку. Рассмотрим выражение `ORDER BY`.

Удалим индекс для `field1`:

```
DROP INDEX test_table_field1_idx;
```

Отсортируем таблицу по `field1`:

```
EXPLAIN (ANALYZE) SELECT * FROM test_table ORDER BY field1;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Gather Merge  (cost=143795.84..143913.70 rows=833614 width=37) (actual time=94.211..273.651 rows=1000000 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Sort  (cost=142795.82..143836.79 rows=416807 width=37) (actual time=91183..127.215 rows=333367 loops=3)
        Sort Key: field1
        Sort Method: external merge  Disk: 13540kB
        Worker 0:  Sort Method: external merge  Disk: 12040kB
        Worker 1:  Sort Method: external merge  Disk: 20504kB
        ->  Parallel Seq Scan on test_table  (cost=0.00..12582.08 rows=416807 width=37) (actual time=0.008..29.574 rows=333367 loops=3)
Planning Time: 0.099 ms
Execution Time: 314.971 ms
```

Когда планировщик определяет, что параллельный запрос является самой быстрой стратегией выполнения, он создает план, который включает узел `Gather` или `Gather Merge` с нашим случае `Gather Merge`. Эта нода содержит подробную плана выполнения, которая обрабатывается параллельных частичными процессами.

Узел операции - `Parallel Seq Scan` с таблицей `test_table`. Ранние последовательные блоки таблицы параллельно сканируются процессами `Worker 0` и `Worker 1`.

После этого PostgreSQL сортирует результаты. Sort Key содержит условие сортировки `field1`. Метод сортировки `external merge Disk` означает, что система создает на диске временные файлы.

План выполнения запроса, приведенный выше, содержит несколько операций и выглядит ресурсоэкстратным.

Создадим индекс для `field1` и выполним тот же запрос:

```
CREATE INDEX ON test_table(field1);
EXPLAIN (ANALYZE) SELECT * FROM test_table ORDER BY field1;
```

Результат:

```
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------
Index Scan using test_table_field1_idx on test_table  (cost=0.42..34319.93 rows=1000100 width=37) (actual time=0.018..356.251 rows=1000100 loops=1)
Planning Time: 0.140 ms
Execution Time: 148.662 ms
```

С индексом план выполнения включает только операцию `Index Scan`, и, как следствие, время выполнения запроса сокращается.

Работа с расширениями

Евгения Кузина

Содержание

Назад к содержанию

Просмотр доступных расширений

Управление расширениями

- Создание расширения
- Удаление расширения
- Изменение расширения

PostgreSQL дает возможность расширить функционал базы данных, группируя объекты SQL в один пакет, в результате чего эти объекты начинают использоваться совместно. Сгруппированные таким образом объекты называются расширениями PostgreSQL.

Расширения — это модули, предоставляющие дополнительную функциональность: функции, операторы, типы и т.д.

Просмотр доступных расширений

Вы можете просмотреть доступные расширения, используя psql:

- Выполните метакоманду `\dx`, чтобы просмотреть список установленных расширений.
- Вызовите метакоманду `\dx+`, чтобы просмотреть список установленных расширений и ассоциированных с ними объектов.

Представление pg_available_extensions содержит расширения, доступные для установки. Используйте следующий код, чтобы просмотреть их:

```
SELECT * FROM pg_available_extensions;
```

Список доступных расширений также можно найти в статье Поддерживаемые расширения.

Информация об установленных расширениях хранится в каталоге pg_extension. В ADPG для служебной базы данных `postgres` предустановлены расширения `plpgsql` и `pg_stat_statements`. При создании новой базы данных используются расширения, которые присутствуют в базе данных `template1` (по умолчанию только `plpgsql`). См. Template Databases.

Используйте следующий код, чтобы проверить установленные расширения:

```
SELECT * FROM pg_extension;
```

Результат:

```
  oid  |      extname       | extowner | extnamespace | extrelocatable | extversion | extconfig | extcondition
-------+--------------------+----------+--------------+----------------+------------+-----------+--------------
 14472 | plpgsql            |       10 |           11 | f              | 1.0        |           |
 16388 | pg_stat_statements |       10 |         2200 | t              | 1.9        |           |
```

Управление расширениями

Можно создавать, изменять и удалять расширения с помощью соответствующих команд.

ВНИМАНИЕ

Если вы используете процедурные языки PL/Perl, PL/PerlU, PL/Python3U, PL/Tcl или PL/TclU, которым необходимы дополнительные пакеты, убедитесь, что все пакеты, требуемые соответствующими расширениями, установлены на всех нодах сервиса ADPG — лидере и всех репликах. В противном случае кластер может быть поврежден во время минорного/мажорного обновления.

Создание расширения

`CREATE EXTENSION` загружает в текущую базу данных новое расширение. Расширение с таким же именем не должно быть уже загружено.

PostgreSQL запускает скрипт-файл расширения. Скрипт создает новые объекты SQL, например, функции, типы данных, операторы и методы поддержки индексов. `CREATE EXTENSION` также сохраняет идентификаторы всех созданных объектов, чтобы их можно было удалить, если вызвана `DROP EXTENSION`.

Пользователь, выполнивший `CREATE EXTENSION`, становится владельцем самого расширения, а также владельцем всех объектов, созданных скриптом расширения.

Если соответствующее расширение не найдено, то команда `CREATE EXTENSION` сообщит, что скрипт-файла расширения не существует.

Если расширение не найдено, установите соответствующий пакет, содержащий его.

Следующий код загружает модуль `pg_stat_statements`, который включен в расширение *contrib*:

```
CREATE EXTENSION pg_stat_statements;
```

Вы можете найти дополнительную информацию об этой команде в статье CREATE EXTENSION.

Удаление расширения

`DROP EXTENSION` удаляет расширение из базы данных. Удаление расширения также влечет удаление всех зависимых объектов.

В коде ниже используется параметр `CASCADE` для удаления расширения, всех зависимых объектов и объектов, зависящих от зависимых объектов расширения:

```
DROP EXTENSION plperlu CASCADE;
```

За дополнительной информацией обратитесь к статье DROP EXTENSION.

Изменение расширения

`ALTER EXTENSION` модифицирует расширение.

Следующий код изменяет схему расширения `hstore` на `schema1`:

```
ALTER EXTENSION hstore SET SCHEMA schema1;
```

Вы можете найти дополнительную информацию об этой команде в статье ALTER EXTENSION.

Содержание

Назад к содержанию

Обзор
Гипертаблицы
- Создание пустой гипертаблицы
- Преобразование таблицы с данными в гипертаблицу

Компрессия
Использование гиперфункций timescaledb для анализа данных
- first и last
- time_bucket
- histogram

Обзор

Расширение `timescaledb` предоставляет набор функций для обработки данных временных рядов. Оно позволяет выполнять масштабируемые вставки и сложные запросы, а также повышает производительность запросов над данными временных рядов.

Перед тем, как установить расширение `timescaledb`, настоятельно с ADPG. Чтобы использовать `timescaledb` необходимо только выполнить команду `CREATE EXTENSION` в той базе данных, в которой необходимо поставить расширение:

```
CREATE EXTENSION timescaledb;
```

> **ПРИМЕЧАНИЕ**
>
> Если опущение `timescaledb` создаст в базе данных `template1`, используемой в качестве шаблона по умолчанию, во всех новых создаваемых базах данных автоматически будет создано расширение `timescaledb`.

ADPG использует версию `timescaledb` 2.15.3. Чтобы это проверить, можно использовать следующий запрос:

```
SELECT extension FROM pg_extension
  WHERE extname = 'timescaledb';
```

Результат:

```
extension
-----------
2.15.3
```

Расширение `timescaledb` добавляет в базу данных ряд функций, представленных в таблице. За дополнительной информацией об API `timescaledb` обратитесь к статье timescaledb API reference.

Вы можете использовать команду `psql \dn` для просмотра списка схем, созданных `timescaledb`:

```
postgres=# \dn
                  List of schemas
        Name         |  Owner
---------------------+----------
 _timescaledb_cache  | postgres
 _timescaledb_catalog| postgres
 _timescaledb_config | postgres
 _timescaledb_debug  | postgres
 _timescaledb_functions | postgres
 _timescaledb_internal | postgres
 public              | pg_database_owner
 timescaledb_experimental | postgres
 timescaledb_information | postgres
```

Гипертаблицы

Расширение `timescaledb` предоставляет `abstraction`, которая является абстракцией высокого уровня, расширяющей традиционные таблицы PostgreSQL для более эффективной работы с временными рядами. Любые операции, которые можно выполнять с обычными таблицами PostgreSQL, можно выполнять с гипертаблицами `timescaledb`. Пользователь взаимодействует с гипертаблицами, создавая таблицы и выполняя запросы к ним, а не с отдельными дочерними таблицами. `timescaledb` разбивает гипертаблицу на `фрагменты (chunks)`.

Создание пустой гипертаблицы

Чтобы создать пустую гипертаблицу, необходимо определить схему таблицы и передать её в качестве параметра функции `create_hypertable`:

```
CREATE TABLE base_table(
    id bigint,
    time_column timestamptz not null default clock_timestamp(),
    data_column1 int default random()*1e3,
    data_column2 int default random()*1e3
);

SELECT create_hypertable('base_table', by_range('time_column'));
```

Результат:

```
create_hypertable
-------------------
 (7,1)
```

Часто из выводов (в данном случае `7`) будет использоваться далее в именах фрагментов, принадлежащих этой гипертаблице.

Вызов `create_hypertable` имеет два обязательных параметра:

- `relation` — идентификатор отношения для преобразования в гипертаблицу. Параметр `relation` должен быть типа `regclass` (`'base_table'` в примере выше).
- `dimension` — должен быть типа `dimension_info`. Для создания экземпляра `dimension_info` необходимо вызвать функцию `by_range` или `by_hash` (тот параметр определяет условие, в соответствии с которым гипертаблица будет разделена на фрагменты: `by_range('time_column')` в примере выше).

Дополнительно гипертаблицу допускается

```
INSERT INTO base_table(id, time_column) SELECT gen_random_uuid(), generate_series(
now() - INTERVAL '6 months',
now(),
INTERVAL '1 minute'
);
```

Используйте команду `psql \dt+` для отображения информации о полученной таблице `base_table`:

```
\dt+ base_table
```

Результат:

```
                                                
```

Использование PL/pgSQL

Содержание

- Обзор
- Структура PL/pgSQL
- Объявления
  - Объявление параметров функции
  - Псевдотипы
  - Копирование типов
  - Копирование строк
  - Алиасы
- Выражения SQL
  - Выполнение SQL-команд
  - Получение результата запроса в переменную
  - Выполнение динамических команд
  - Получение статуса результата
  - Получение данных диагностики
- Структуры управления
  - RETURN
  - Возврат из процедур
  - Возврат из функций, возвращающих множество
  - Возврат из функций, возвращающих курсор
  - Условия и циклы
  - Ошибки и сообщения
  - Курсоры
  - Триггеры

Назад к содержанию

Обзор

PL/pgSQL — процедурный язык для СУБД PostgreSQL, который позволяет выполнять сложные вычисления, недоступные в обычном SQL, такие как использование управляющих структур для контроля логики. Вы можете создавать пользовательские функции и триггеры, добавлять управляющие структуры к языку SQL, выполнять сложные вычисления, наследовать все пользовательские типы, функции и операторы, определять функции любого типа, процедуры и триггеры, а также использовать возможности модульности.

- Создание функций и триггеров
- Добавление управляющих структур к языку
- Выполнение сложных вычислений
- Наследование всех пользовательских типов, функций и операторов
- Определение функций любого типа, процедур и триггеров
- Использование возможностей модульности

Структура PL/pgSQL

PL/pgSQL предназначен для создания функций, процедур и триггеров. Структура языка выглядит следующим образом:

```
[ <<label>> ]
[ DECLARE
    declarations ]
BEGIN
    statements
END [ label ];
```

Каждое объявление и каждый оператор в блоке должны завершаться точкой с запятой. Блок может быть вложен в другой блок. Вы можете использовать `RAISE` для генерации сообщений и вызова ошибок. Ключевые слова `DECLARE` и `BEGIN` не завершаются точкой с запятой.

Любой текст после `--` и до конца строки является однострочным комментарием. Многострочные комментарии заключаются в `/* ... */`.

Вы можете использовать переменные для хранения результатов запросов и передачи параметров между командами. Переменные объявляются в секции `DECLARE` блока.

Все переменные, используемые в блоке, должны быть объявлены в секции `DECLARE`. Исключение составляют переменная цикла `FOR` (целочисленный цикл `FOR`), переменные, объявленные как `RECORD`, и переменные, объявленные как строки таблицы.

Переменные могут иметь любой тип данных, который существует в PostgreSQL, например `integer`, `numeric` или `varchar`. Вы можете объявить переменную, которая имеет тот же тип данных, что и столбец таблицы, используя `%TYPE`. Вы можете объявить переменную, которая представляет всю строку таблицы, используя составной тип, соответствующий структуре таблицы, или `%ROWTYPE`. Вы можете определить новый составной тип с помощью команды `CREATE TYPE`. Вы можете объявить переменную, которая может содержать строки различных структур, как `RECORD`.

Объявления

Все переменные, используемые в блоке, должны быть объявлены в секции `DECLARE`. Переменные могут иметь любой тип данных, который существует в PostgreSQL, например `integer`, `numeric` или `varchar`.

```
name [ CONSTANT ] type [ COLLATE collation_name ] [ NOT NULL ] [ { DEFAULT | := | = } expression ];
```

Параметр `CONSTANT` предотвращает изменение переменной, чтобы она оставалась постоянной на протяжении всего блока. Параметр `COLLATE` выбирает правило сортировки для переменной. Если `COLLATE` не указано, используется правило сортировки по умолчанию для типа данных переменной. Параметр `NOT NULL` обеспечивает, что переменная не будет содержать значение `NULL`. Если `NOT NULL` указано, переменная должна быть инициализирована. Выражение инициализации вычисляется каждый раз при входе в блок.

Вы можете объявить переменную, которая имеет тот же тип данных, что и столбец таблицы, используя `%TYPE`. Вы можете объявить переменную, которая представляет всю строку таблицы, используя составной тип, соответствующий структуре таблицы, или `%ROWTYPE`. Вы можете определить новый составной тип с помощью команды `CREATE TYPE`. Вы можете объявить переменную, которая может содержать строки различных структур, как `RECORD`.

Вы можете объявить переменную, которая имеет тот же тип данных, что и столбец таблицы, используя `%TYPE`. Вы можете объявить переменную, которая представляет всю строку таблицы, используя составной тип, соответствующий структуре таблицы, или `%ROWTYPE`. Вы можете определить новый составной тип с помощью команды `CREATE TYPE`. Вы можете объявить переменную, которая может содержать строки различных структур, как `RECORD`.

```
quantity integer;
url varchar;
myrow tablename%ROWTYPE;
myfield tablename.columnname%TYPE;
arow RECORD;
```

Вывод:

`CONSTANT`	Не изменяется
`COLLATE collation_name`	Правило сортировки
`NOT NULL`	Должно быть инициализировано
`DEFAULT value` или `:= value` или `= value`	Значение по умолчанию

Вы можете объявить переменную, которая имеет тот же тип данных, что и столбец таблицы, используя `%TYPE`. Вы можете объявить переменную, которая представляет всю строку таблицы, используя составной тип, соответствующий структуре таблицы, или `%ROWTYPE`. Вы можете определить новый составной тип с помощью команды `CREATE TYPE`. Вы можете объявить переменную, которая может содержать строки различных структур, как `RECORD`.

Вывод:

```
quantity integer;
url varchar := 'http://mysite.com';
user_id CONSTANT integer := 10;
```

Объявление параметров функции

Параметры функции передаются по имени и по позиции. Вы можете объявить параметр функции, используя `%TYPE`, чтобы он имел тот же тип данных, что и столбец таблицы.

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
DECLARE
    tax rate_type;
BEGIN
    RETURN subtotal * tax;
END;
$$ LANGUAGE plpgsql;
```

Псевдотипы

Псевдотип `ELEMENT TYPE` может быть использован для указания типа элемента массива.

```
CREATE FUNCTION sum_array(a anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Объявление параметров функции

Параметры функции передаются по имени и по позиции. Вы можете объявить параметр функции, используя `%TYPE`, чтобы он имел тот же тип данных, что и столбец таблицы.

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Копирование типов

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Копирование строк

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Алиасы

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Типы строк

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вывод:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Тип таблиц

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вывод:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Выполнение SQL-команд

Вы можете выполнять команды SQL, которые не возвращают строки, просто написав команду. Вы должны использовать `PERFORM` вместо `SELECT`, если вы не хотите возвращать строки. Вы можете использовать `EXECUTE` для выполнения динамических команд, которые строятся во время выполнения.

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Получение результата запроса в переменную

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Выполнение динамических команд

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Получение статуса результата

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Получение данных диагностики

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Структуры управления

Структуры управления позволяют вам управлять логикой вашей программы. PL/pgSQL предоставляет широкий набор структур управления, включая условные операторы, циклы и обработку ошибок.

RETURN

`RETURN` может использоваться для немедленного возврата из функции. Для функций, возвращающих скалярный тип, выражение будет вычислено и возвращено. Для функций, возвращающих составной (composite) тип или набор результатов, будет возвращена строка или набор строк. Для функций, возвращающих `void`, оператор `RETURN` может использоваться для выхода из функции.

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Возврат из процедур

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Возврат из функций, возвращающих множество

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Возврат из функций, возвращающих курсор

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Условия и циклы

- `IF ... THEN ... END IF;`
- `IF ... THEN ... ELSE ... END IF;`
- `IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF;`
- `CASE search-expression WHEN expression [, expression [ ... ]] THEN statements [ ... ] [ ELSE statements ] END CASE;`
- `CASE WHEN boolean-expression THEN statements [ ... ] [ ELSE statements ] END CASE;`
- `[ <<label>> ] LOOP statements END LOOP [ label ];`
- `EXIT [ label ] [ WHEN boolean-expression ];`
- `CONTINUE [ label ] [ WHEN boolean-expression ];`
- `WHILE boolean-expression LOOP statements END LOOP;`
- `FOR name IN [ REVERSE ] expression .. expression [ BY expression ] LOOP statements END LOOP;`

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вывод:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Ошибки и сообщения

Используйте `RAISE` для генерации сообщений и вызова ошибок. Уровни ошибок включают `DEBUG`, `LOG`, `INFO`, `NOTICE`, `WARNING` и `EXCEPTION`. По умолчанию `EXCEPTION` прерывает текущую транзакцию. Вы можете использовать `ASSERT` для проверки предположений.

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Курсоры

Курсоры позволяют вам перебирать результаты запроса по одной строке за раз. Вы можете объявить курсор, открыть его, получить из него строки и закрыть его. Вы можете использовать `FOR` для перебора всех строк курсора.

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Триггеры

Триггеры позволяют вам вызывать функцию при изменении данных в таблице. Вы можете использовать триггеры для проверки данных, поддержания целостности данных и выполнения сложных операций.

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Пример

Вот пример функции, которая суммирует элементы массива:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вы можете вызвать эту функцию, используя `SELECT sum_array(ARRAY[1,2,3,4,5]);`. Вы можете использовать `sum_array` в любом месте, где вы можете использовать встроенную функцию, такую как `sum`.

Вот пример функции, которая возвращает набор результатов:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вы можете вызвать эту функцию, используя `SELECT * FROM sum_array(ARRAY[1,2,3,4,5]);`. Вы можете использовать `sum_array` в любом месте, где вы можете использовать таблицу, такую как `SELECT * FROM sum_array(ARRAY[1,2,3,4,5]) WHERE sum_array > 10;`.

Вот пример функции, которая возвращает курсор:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вы можете вызвать эту функцию, используя `SELECT * FROM sum_array(ARRAY[1,2,3,4,5]);`. Вы можете использовать `sum_array` в любом месте, где вы можете использовать курсор, такую как `SELECT * FROM sum_array(ARRAY[1,2,3,4,5]) WHERE sum_array > 10;`.

Вот пример функции, которая возвращает триггер:

```
CREATE FUNCTION sum_array(anyarray) RETURNS anyelement AS $$
DECLARE
    s ALIAS FOR $0;
    i integer;
BEGIN
    s := 0;
    FOR i IN 1..array_upper(a, 1) LOOP
        s := s + a[i];
    END LOOP;
    RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Вы можете вызвать эту функцию, используя `SELECT * FROM sum_array(ARRAY[1,2,3,4,5]);`. Вы можете использовать `sum_array` в любом месте, где вы можете использовать триггер, такую как `SELECT * FROM sum_array(ARRAY[1,2,3,4,5]) WHERE sum_array > 10;`.

Использование PL/Python

Назад

Содержание

Обзор
Функции PL/Python
Типы данных
Совместное использование данных
Доступ к базе данных
Функции, возвращающие множество
Триггеры
Анонимные блоки кода
Совместное использование данных
Доступ к базе данных
Функции, возвращающие множество
Триггеры
Анонимные блоки кода

Открыть в приложении

Обзор

Процедурный язык PL/Python позволяет писать функции PostgreSQL на языке Python 3. Синтаксис функции задаётся командой CREATE FUNCTION, в которой указывается имя языка, имя функции, возвращаемый тип и исходный код функции. Если аргументы функции именованные, то они доступны по своим именам. Например, эта функция возвращает сумму двух своих целочисленных аргументов:

```
CREATE FUNCTION pymax (a integer, b integer)
  RETURNS integer
AS $$
  if a > b:
    return a
  return b
$$ LANGUAGE plpython3u;
```

Обратите внимание

Синтаксис функции **отличается** от синтаксиса функций, написанных на других процедурных языках. Вместо ключевых слов **BEGIN** и **END** используются двоеточие и отступы. Тело функции — это просто скрипт, который интерпретатор Python выполняет при вызове функции. Аргументы передаются скрипту как переменные, а результат возвращается с помощью **return**.

Функции PL/Python

Функции, написанные на PL/Python, могут использовать любые стандартные возможности Python. Это означает, что в них можно импортировать и использовать **любые** модули, установленные в системе. Например, можно написать функцию, которая использует регулярные выражения из модуля **re**:

```
CREATE FUNCTION is_email(email text)
  RETURNS boolean
AS $$
  import re
  return re.match(r'^[^@]+@[^@]+\.[^@]+$', email) is not None
$$ LANGUAGE plpython3u;
```

Если функция возвращает значение типа **boolean**, интерпретатор Python автоматически преобразует результат выражения в истинность или ложность. Значения **None** преобразуются в **NULL**. Функция может возвращать составные типы, списки и множества строк — подробности ниже.

Функции, написанные на PL/Python, могут использовать любые стандартные возможности Python. Это означает, что в них можно импортировать и использовать **любые** модули, установленные в системе. Например, можно написать функцию, которая использует регулярные выражения из модуля **re**:

```
CREATE FUNCTION is_email(email text)
  RETURNS boolean
AS $$
  import re
  return re.match(r'^[^@]+@[^@]+\.[^@]+$', email) is not None
$$ LANGUAGE plpython3u;
```

Если функция возвращает значение типа **boolean**, интерпретатор Python автоматически преобразует результат выражения в истинность или ложность. Значения **None** преобразуются в **NULL**. Функция может возвращать составные типы, списки и множества строк — подробности ниже.

Типы данных

Соответствие типов аргументов

Тип PostgreSQL	Тип Python
integer	int
double precision	float
boolean	bool
text	str
bytea	bytes
date	datetime.date
timestamp	datetime.datetime
numeric	decimal.Decimal
array	list
composite type	dict или sequence

Обратное преобразование выполняется при возврате значения из функции. Для составных типов можно возвращать словарь, последовательность или объект с атрибутами, имена которых совпадают с именами полей.

Тип PostgreSQL	Тип возвращаемого значения Python
composite type	dict, sequence или объект с нужными атрибутами
array	list или другой итерируемый объект
set of	итерируемый объект, элементы которого соответствуют базовому типу
void	None (возвращается автоматически)
trigger	dict с ключами NEW, OLD, TG_OP и другими

NULL и None

Значения **NULL** в SQL передаются в функцию как **None**. Чтобы функция корректно обрабатывала отсутствующие данные, проверяйте аргументы на **None** перед использованием. Если функция объявлена как **STRICT**, PostgreSQL сам вернёт **NULL**, не вызывая функцию, если хотя бы один аргумент равен **NULL**.

```
CREATE FUNCTION safe_upper(s text)
  RETURNS text
AS $$
  if s is None:
    return None
  return s.upper()
$$ LANGUAGE plpython3u;

SELECT safe_upper('hello'), safe_upper(NULL);
```

Результат

```
 safe_upper | safe_upper
------------+------------
 HELLO      |
(1 row)
```

Arrays и списки

Массивы PostgreSQL автоматически преобразуются в списки Python и обратно. Многомерные массивы представляются как вложенные списки. Пустой массив — это пустой список.

```
CREATE FUNCTION sum_array(a integer[])
  RETURNS integer
AS $$
  return sum(a)
$$ LANGUAGE plpython3u;

SELECT sum_array(ARRAY[1, 2, 3, 4]);
```

Результат

```
 sum_array
-----------
        10
(1 row)
```

Составные типы

Функция может принимать составной тип как словарь, где ключи — имена полей. Для возврата составного типа можно вернуть словарь, последовательность или объект с нужными атрибутами. Имена полей должны совпадать с именами столбцов.

```
CREATE TABLE point (x int, y int);

CREATE FUNCTION make_point(x int, y int)
  RETURNS point
AS $$
  return {'x': x, 'y': y}
$$ LANGUAGE plpython3u;

SELECT * FROM make_point(3, 7);
```

Результат

```
 x | y
---+---
 3 | 7
(1 row)
```

Совместное использование данных

Каждая функция PL/Python выполняется в собственном пространстве имён, поэтому глобальные переменные между вызовами не сохраняются. Для хранения состояния между вызовами используйте словарь **SD** — он доступен всем функциям в рамках одной сессии.

```
CREATE FUNCTION counter_next() RETURNS int
AS $$
  if 'n' not in SD:
    SD['n'] = 0
  SD['n'] += 1
  return SD['n']
$$ LANGUAGE plpython3u;

SELECT counter_next(), counter_next(), counter_next();
```

Результат

```
 counter_next | counter_next | counter_next
--------------+--------------+--------------
            1 |            2 |            3
(1 row)
```

Для общих данных между разными функциями используйте словарь **GD**. Он работает аналогично **SD**, но доступен всем функциям в базе данных.

```
CREATE FUNCTION set_global(k text, v text) RETURNS void
AS $$
  GD[k] = v
$$ LANGUAGE plpython3u;

CREATE FUNCTION get_global(k text) RETURNS text
AS $$
  return GD.get(k)
$$ LANGUAGE plpython3u;

SELECT set_global('mode', 'fast'), get_global('mode');
```

Результат

```
 set_global | get_global
------------+------------
            | fast
(1 row)
```

Доступ к базе данных

Внутри функции PL/Python можно выполнять SQL-запросы через модуль **plpy**. Функция **plpy.execute** принимает текст запроса и возвращает результат в виде списка словарей, где ключи — имена столбцов.

```
CREATE FUNCTION count_users() RETURNS int
AS $$
  rv = plpy.execute("SELECT count(*) AS n FROM users")
  return rv[0]["n"]
$$ LANGUAGE plpython3u;

SELECT count_users();
```

Результат

```
 count_users
-------------
           3
(1 row)
```

Для подготовленных запросов используйте **plpy.prepare** и **plpy.execute**. Это позволяет передавать параметры безопасно, без риска SQL-инъекций.

```
CREATE FUNCTION user_by_id(uid int) RETURNS text
AS $$
  plan = plpy.prepare("SELECT name FROM users WHERE id = $1", ["int"])
  rv = plpy.execute(plan, [uid])
  return rv[0]["name"] if rv else None
$$ LANGUAGE plpython3u;

SELECT user_by_id(1), user_by_id(999);
```

Результат

```
 user_by_id | user_by_id
------------+------------
 alice      |
(1 row)
```

Функции, возвращающие множество

Функция может возвращать несколько строк, если объявлена с **RETURNS SETOF** или **RETURNS TABLE**. Для этого используйте **yield** вместо **return**. Каждый вызов **yield** добавляет одну строку в результат.

```
CREATE FUNCTION squares(n int) RETURNS TABLE(x int, sq int)
AS $$
  for i in range(1, n + 1):
    yield (i, i * i)
$$ LANGUAGE plpython3u;

SELECT * FROM squares(4);
```

Результат

```
 x | sq
---+----
 1 |  1
 2 |  4
 3 |  9
 4 | 16
(4 rows)
```

Можно возвращать и составные типы, передавая словарь на каждой итерации.

```
CREATE FUNCTION numbered_names(names text[])
  RETURNS TABLE(num int, name text)
AS $$
  for i, n in enumerate(names, 1):
    yield (i, n)
$$ LANGUAGE plpython3u;

SELECT * FROM numbered_names(ARRAY['a', 'b', 'c']);
```

Результат

```
 num | name
-----+------
   1 | a
   2 | b
   3 | c
(3 rows)
```

Триггеры

Функция-триггер вызывается автоматически при операциях INSERT, UPDATE или DELETE. Данные о текущей строке доступны через словарь **TD**: **TD["new"]** — новая строка, **TD["old"]** — старая, **TD["event"]** — тип операции.

```
CREATE FUNCTION log_changes() RETURNS trigger
AS $$
  plpy.notice("%s on %s" % (TD["event"], TD["table_name"]))
  return None
$$ LANGUAGE plpython3u;

CREATE TRIGGER t_log AFTER INSERT OR UPDATE ON users
  FOR EACH ROW EXECUTE FUNCTION log_changes();
```

Результат

```
NOTICE:  INSERT on users
```

Анонимные блоки кода

Команда **DO** позволяет выполнить код на PL/Python без создания функции. Удобно для разовых операций.

```
DO $$
  plpy.notice("hello from anonymous block")
$$ LANGUAGE plpython3u;
```

Результат

```
NOTICE:  hello from anonymous block
DO
```

Совместное использование данных

Каждая функция выполняется в собственном пространстве имён, поэтому глобальные переменные между вызовами не сохраняются. Для хранения состояния между вызовами используйте словарь **SD** — он доступен всем функциям в рамках одной сессии.

```
CREATE FUNCTION counter_next() RETURNS int
AS $$
  if 'n' not in SD:
    SD['n'] = 0
  SD['n'] += 1
  return SD['n']
$$ LANGUAGE plpython3u;

SELECT counter_next(), counter_next(), counter_next();
```

Результат

```
 counter_next | counter_next | counter_next
--------------+--------------+--------------
            1 |            2 |            3
(1 row)
```

Для общих данных между разными функциями используйте словарь **GD**. Он работает аналогично **SD**, но доступен всем функциям в базе данных.

```
CREATE FUNCTION set_global(k text, v text) RETURNS void
AS $$
  GD[k] = v
$$ LANGUAGE plpython3u;

CREATE FUNCTION get_global(k text) RETURNS text
AS $$
  return GD.get(k)
$$ LANGUAGE plpython3u;

SELECT set_global('mode', 'fast'), get_global('mode');
```

Результат

```
 set_global | get_global
------------+------------
            | fast
(1 row)
```

Доступ к базе данных

Внутри функции PL/Python можно выполнять SQL-запросы через модуль **plpy**. Функция **plpy.execute** принимает текст запроса и возвращает результат в виде списка словарей, где ключи — имена столбцов.

```
CREATE FUNCTION count_users() RETURNS int
AS $$
  rv = plpy.execute("SELECT count(*) AS n FROM users")
  return rv[0]["n"]
$$ LANGUAGE plpython3u;

SELECT count_users();
```

Результат

```
 count_users
-------------
           3
(1 row)
```

Для подготовленных запросов используйте **plpy.prepare** и **plpy.execute**. Это позволяет передавать параметры безопасно, без риска SQL-инъекций.

```
CREATE FUNCTION user_by_id(uid int) RETURNS text
AS $$
  plan = plpy.prepare("SELECT name FROM users WHERE id = $1", ["int"])
  rv = plpy.execute(plan, [uid])
  return rv[0]["name"] if rv else None
$$ LANGUAGE plpython3u;

SELECT user_by_id(1), user_by_id(999);
```

Результат

```
 user_by_id | user_by_id
------------+------------
 alice      |
(1 row)
```

Функции, возвращающие множество

Функция может возвращать несколько строк, если объявлена с **RETURNS SETOF** или **RETURNS TABLE**. Для этого используйте **yield** вместо **return**. Каждый вызов **yield** добавляет одну строку в результат.

```
CREATE FUNCTION squares(n int) RETURNS TABLE(x int, sq int)
AS $$
  for i in range(1, n + 1):
    yield (i, i * i)
$$ LANGUAGE plpython3u;

SELECT * FROM squares(4);
```

Результат

```
 x | sq
---+----
 1 |  1
 2 |  4
 3 |  9
 4 | 16
(4 rows)
```

Можно возвращать и составные типы, передавая словарь на каждой итерации.

```
CREATE FUNCTION numbered_names(names text[])
  RETURNS TABLE(num int, name text)
AS $$
  for i, n in enumerate(names, 1):
    yield (i, n)
$$ LANGUAGE plpython3u;

SELECT * FROM numbered_names(ARRAY['a', 'b', 'c']);
```

Результат

```
 num | name
-----+------
   1 | a
   2 | b
   3 | c
(3 rows)
```

Триггеры

Функция-триггер вызывается автоматически при операциях INSERT, UPDATE или DELETE. Данные о текущей строке доступны через словарь **TD**: **TD["new"]** — новая строка, **TD["old"]** — старая, **TD["event"]** — тип операции.

```
CREATE FUNCTION log_changes() RETURNS trigger
AS $$
  plpy.notice("%s on %s" % (TD["event"], TD["table_name"]))
  return None
$$ LANGUAGE plpython3u;

CREATE TRIGGER t_log AFTER INSERT OR UPDATE ON users
  FOR EACH ROW EXECUTE FUNCTION log_changes();
```

Результат

```
NOTICE:  INSERT on users
```

Анонимные блоки кода

Команда **DO** позволяет выполнить код на PL/Python без создания функции. Удобно для разовых операций.

```
DO $$
  plpy.notice("hello from anonymous block")
$$ LANGUAGE plpython3u;
```

Результат

```
NOTICE:  hello from anonymous block
DO
```

Совместное использование данных

Каждая функция выполняется в собственном пространстве имён, поэтому глобальные переменные между вызовами не сохраняются. Для хранения состояния между вызовами используйте словарь **SD** — он доступен всем функциям в рамках одной сессии.

```
CREATE FUNCTION counter_next() RETURNS int
AS $$
  if 'n' not in SD:
    SD['n'] = 0
  SD['n'] += 1
  return SD['n']
$$ LANGUAGE plpython3u;

SELECT counter_next(), counter_next(), counter_next();
```

Результат

```
 counter_next | counter_next | counter_next
--------------+--------------+--------------
            1 |            2 |            3
(1 row)
```

Для общих данных между разными функциями используйте словарь **GD**. Он работает аналогично **SD**, но доступен всем функциям в базе данных.

```
CREATE FUNCTION set_global(k text, v text) RETURNS void
AS $$
  GD[k] = v
$$ LANGUAGE plpython3u;

CREATE FUNCTION get_global(k text) RETURNS text
AS $$
  return GD.get(k)
$$ LANGUAGE plpython3u;

SELECT set_global('mode', 'fast'), get_global('mode');
```

Результат

```
 set_global | get_global
------------+------------
            | fast
(1 row)
```

Доступ к базе данных

Внутри функции PL/Python можно выполнять SQL-запросы через модуль **plpy**. Функция **plpy.execute** принимает текст запроса и возвращает результат в виде списка словарей, где ключи — имена столбцов.

```
CREATE FUNCTION count_users() RETURNS int
AS $$
  rv = plpy.execute("SELECT count(*) AS n FROM users")
  return rv[0]["n"]
$$ LANGUAGE plpython3u;

SELECT count_users();
```

Результат

```
 count_users
-------------
           3
(1 row)
```

Для подготовленных запросов используйте **plpy.prepare** и **plpy.execute**. Это позволяет передавать параметры безопасно, без риска SQL-инъекций.

```
CREATE FUNCTION user_by_id(uid int) RETURNS text
AS $$
  plan = plpy.prepare("SELECT name FROM users WHERE id = $1", ["int"])
  rv = plpy.execute(plan, [uid])
  return rv[0]["name"] if rv else None
$$ LANGUAGE plpython3u;

SELECT user_by_id(1), user_by_id(999);
```

Результат

```
 user_by_id | user_by_id
------------+------------
 alice      |
(1 row)
```

Функции, возвращающие множество

Функция может возвращать несколько строк, если объявлена с **RETURNS SETOF** или **RETURNS TABLE**. Для этого используйте **yield** вместо **return**. Каждый вызов **yield** добавляет одну строку в результат.

```
CREATE FUNCTION squares(n int) RETURNS TABLE(x int, sq int)
AS $$
  for i in range(1, n + 1):
    yield (i, i * i)
$$ LANGUAGE plpython3u;

SELECT * FROM squares(4);
```

Результат

```
 x | sq
---+----
 1 |  1
 2 |  4
 3 |  9
 4 | 16
(4 rows)
```

Можно возвращать и составные типы, передавая словарь на каждой итерации.

```
CREATE FUNCTION numbered_names(names text[])
  RETURNS TABLE(num int, name text)
AS $$
  for i, n in enumerate(names, 1):
    yield (i, n)
$$ LANGUAGE plpython3u;

SELECT * FROM numbered_names(ARRAY['a', 'b', 'c']);
```

Результат

```
 num | name
-----+------
   1 | a
   2 | b
   3 | c
(3 rows)
```

Триггеры

Функция-триггер вызывается автоматически при операциях INSERT, UPDATE или DELETE. Данные о текущей строке доступны через словарь **TD**: **TD["new"]** — новая строка, **TD["old"]** — старая, **TD["event"]** — тип операции.

```
CREATE FUNCTION log_changes() RETURNS trigger
AS $$
  plpy.notice("%s on %s" % (TD["event"], TD["table_name"]))
  return None
$$ LANGUAGE plpython3u;

CREATE TRIGGER t_log AFTER INSERT OR UPDATE ON users
  FOR EACH ROW EXECUTE FUNCTION log_changes();
```

Результат

```
NOTICE:  INSERT on users
```

Анонимные блоки кода

Команда **DO** позволяет выполнить код на PL/Python без создания функции. Удобно для разовых операций.

```
DO $$
  plpy.notice("hello from anonymous block")
$$ LANGUAGE plpython3u;
```

Результат

```
NOTICE:  hello from anonymous block
DO
```

Использование функций на языке C

Кэшируется

Содержание

- Динамическая загрузка
- Ручной создание функции
- Передача списка в функцию языка C
- Соглашение о вызовах версии 1
- Основные правила написания кода
- Аргументы соглашается типа
- Возврат множества

Динамическая загрузка

Для динамической загрузки функции в языке C используется библиотека `dlfcn.h`. Основные функции для работы с динамическими библиотеками:

- `dlopen(const char *filename, int flag)` — открывает динамическую библиотеку.
- `dlsym(void *handle, const char *symbol)` — получает адрес функции из библиотеки.
- `dlclose(void *handle)` — закрывает библиотеку.
- `dlerror()` — возвращает строку с описанием последней ошибки.

Пример загрузки функции из динамической библиотеки:

```c
#include <dlfcn.h>
#include <stdio.h>

int main() {
    void *handle;
    int (*add)(int, int);
    char *error;

    handle = dlopen("./mylib.so", RTLD_LAZY);
    if (!handle) {
        fprintf(stderr, "%s\n", dlerror());
        return 1;
    }

    add = dlsym(handle, "add");
    if ((error = dlerror()) != NULL) {
        fprintf(stderr, "%s\n", error);
        return 1;
    }

    printf("%d\n", add(2, 3));
    dlclose(handle);
    return 0;
}
```

Ручной создание функции

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int main() {
    int (*func)(int, int) = add;
    printf("%d\n", func(2, 3));
    return 0;
}
```

API

```c
// Создать функцию
int (*func)(int, int) = add;

// Вызвать функцию
int result = func(2, 3);

// Передать функцию в качестве аргумента
void apply(int (*f)(int, int), int a, int b) {
    printf("%d\n", f(a, b));
}

apply(add, 2, 3);
```

```c
// Создать указатель на функцию
int (*func)(int, int) = add;

// Вызвать функцию через указатель
int result = func(2, 3);

// Передать функцию в качестве аргумента
void apply(int (*f)(int, int), int a, int b) {
    printf("%d\n", f(a, b));
}

apply(add, 2, 3);
```

Передача списка в функцию языка C

Для передачи списка в функцию языка C можно использовать массив или указатель на первый элемент списка.

```c
#include <stdio.h>

void print_list(int *list, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", list[i]);
    }
    printf("\n");
}

int main() {
    int my_list[] = {1, 2, 3, 4, 5};
    print_list(my_list, 5);
    return 0;
}
```

Также можно передать список в функцию с помощью указателя на первый элемент списка и размера списка.

```c
#include <stdio.h>

void print_list(int *list, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", list[i]);
    }
    printf("\n");
}

int main() {
    int my_list[] = {1, 2, 3, 4, 5};
    print_list(my_list, 5);
    return 0;
}
```

Для передачи списка в функцию языка C можно использовать массив или указатель на первый элемент списка.

```c
#include <stdio.h>

void print_list(int *list, int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", list[i]);
    }
    printf("\n");
}

int main() {
    int my_list[] = {1, 2, 3, 4, 5};
    print_list(my_list, 5);
    return 0;
}
```

Базовые типы и функции языка C

```c
#include <stdio.h>

int main() {
    int a = 10;
    int b = 20;
    int sum = a + b;
    printf("Sum: %d\n", sum);
    return 0;
}
```

Использование PL/Perl

Содержание

Обзор

Как писать функции на языке PL/Perl

Аргументы и результаты функций

Данные в базе данных и на языке PL/Perl

Полезные скалярные строки

Получение большого набора строк

Полезные массивы

Использование предварительно подготовленных планов запросов

Управление транзакциями

Встроенные статические функции

Содержание

- Обзор
- Как писать функции на языке PL/Perl
  - Аргументы и результаты функций
  - Триггеры
  - Триггеры событий
  - Использование SPI из функций и процедур
- Аргументы и результаты функций
  - Функции, процедуры и аргументы PL/Perl
  - Триггеры
  - Триггеры событий
- Данные в базе данных и на языке PL/Perl
  - Полезные скалярные строки
  - Получение большого набора строк
  - Полезные массивы
  - Использование предварительно подготовленных планов запросов
  - Управление транзакциями
  - Встроенные статические функции

Обзор

В PL/Perl — это загружаемый процедурный язык, который позволяет писать функции PostgreSQL на языке Perl. Основное преимущество PL/Perl заключается в том, что он позволяет использовать мощные возможности Perl для манипулирования данными. Обычно функции PL/Perl компилируются и выполняются быстрее, чем процедуры, написанные на PL/pgSQL, поскольку интерпретатор Perl уже загружен, и код выполняется в виде встроенного интерпретатора. Кроме того, PL/Perl предоставляет доступ к большому количеству модулей Perl, написанных на CPAN, и может использовать их для выполнения задач, которые было бы сложно или невозможно выполнить на PL/pgSQL.

Внимание

Если пользователь не является суперпользователем, PL/Perl использует безопасную версию Perl, которая ограничивает доступ к файловой системе, сетевым вызовам и выполнению внешних программ. Это предотвращает злоупотребление языком для получения несанкционированного доступа к операционной системе. Только доверенные языки, такие как PL/Perl, могут быть установлены суперпользователями. Функции PL/Perl выполняются в ограниченной среде, которая не позволяет им получать доступ к файловой системе, если только они не используют SPI для взаимодействия с базой данных.

Чтобы установить PL/Perl для конкретной базы данных, выполните [CREATE EXTENSION plperl;](#).

Совет

Чтобы установить версию PL/Perl, которая не является доверенной, установите расширение [plperlu](#). Подробности см. в разделе [Доверенные и недоверенные языки](#).

Как писать функции на языке PL/Perl

Функции PL/Perl объявляются с помощью стандартного [CREATE FUNCTION](#). Тело функции — это просто код Perl. Переменная [$_TD](#) содержит ссылку на структуру данных, описывающую вызов триггера, поэтому код триггера может использовать [$_TD->{new}](#) и [$_TD->{old}](#) для доступа к строкам данных. Функции также могут возвращать [undef](#) в скалярном контексте, чтобы указать, что значение SQL NULL должно быть возвращено из функции.

Аргументы и результаты функций

Аргументы функции PL/Perl передаются в [@_](#) как обычно. Имена аргументов SQL также доступны в виде отдельных переменных, если они не конфликтуют с существующими переменными. Если аргумент имеет значение SQL NULL, соответствующая переменная будет [undef](#). Для функций, возвращающих несколько значений, можно возвращать ссылку на массив, ссылку на хеш или, в контексте списка, простой список.

```
CREATE FUNCTION concat_lower(text, text) RETURNS text AS $$
  my ($a, $b) = @_;
  return lc($a . $b);
$$ LANGUAGE plperl;
```

Для возврата результирующего набора функция может возвращать ссылку на массив строк, где каждая строка является ссылкой на хеш, или она может использовать [return_next](#) для потоковой передачи строк.

```
CREATE FUNCTION numbers(integer) RETURNS SETOF integer AS $$
  for my $i (1 .. $_[0]) {
    return_next($i);
  }
  return undef;
$$ LANGUAGE plperl;
```

Триггеры

Триггеры PL/Perl получают свои данные через [$_TD](#). Поле [$_TD->{new}](#) содержит новую строку для операций INSERT и UPDATE, а [$_TD->{old}](#) содержит старую строку для UPDATE и DELETE. Поле [$_TD->{event}](#) сообщает, какая операция вызвала триггер, а [$_TD->{when}](#) указывает, является ли это BEFORE, AFTER или INSTEAD OF. Триггеры могут изменять строку, возвращая измененную ссылку на хеш, или они могут пропустить операцию, возвращая [undef](#).

```
CREATE FUNCTION audit_trigger() RETURNS trigger AS $$
  my $row = $_TD->{new} // $_TD->{old};
  elog(NOTICE, "$_TD->{when} $_TD->{event} on id=$row->{id}");
  return undef;
$$ LANGUAGE plperl;
```

Триггеры событий

Триггеры событий PL/Perl получают имя события в [$_TD->{event}](#) и имя тега в [$_TD->{tag_name}](#). Они полезны для аудита изменений схемы или ограничения того, какие команды DDL разрешены.

```
CREATE FUNCTION block_drop() RETURNS event_trigger AS $$
  if ($_TD->{tag_name} =~ /^DROP/) {
    elog(ERROR, "DROP commands are not allowed");
  }
$$ LANGUAGE plperl;

CREATE EVENT TRIGGER no_drop ON ddl_command_start
  EXECUTE FUNCTION block_drop();
```

Использование SPI из функций и процедур

Функции PL/Perl могут выполнять запросы SQL с помощью [spi_exec_query](#) для буферизованных результатов или [spi_query](#) / [spi_fetch](#) для потоковой передачи больших наборов. Функция [spi_prepare](#) создает подготовленные операторы, которые можно повторно использовать с [spi_exec_prepared](#). Всегда освобождайте курсоры с помощью [spi_cursor_close](#), когда закончите.

```
CREATE FUNCTION count_rows(text) RETURNS integer AS $$
  my $rv = spi_exec_query("SELECT count(*) AS n FROM $_[0]");
  return $rv->{rows}[0]->{n};
$$ LANGUAGE plperl;
```

Аргументы и результаты функций

Функции, процедуры и аргументы PL/Perl

Функции PL/Perl объявляются с помощью стандартного [CREATE FUNCTION](#). Тело функции — это просто код Perl. Переменная [$_TD](#) содержит ссылку на структуру данных, описывающую вызов триггера, поэтому код триггера может использовать [$_TD->{new}](#) и [$_TD->{old}](#) для доступа к строкам данных. Функции также могут возвращать [undef](#) в скалярном контексте, чтобы указать, что значение SQL NULL должно быть возвращено из функции.

```
CREATE FUNCTION concat_lower(text, text) RETURNS text AS $$
  my ($a, $b) = @_;
  return lc($a . $b);
$$ LANGUAGE plperl;
```

Аргументы функции PL/Perl передаются в [@_](#) как обычно. Имена аргументов SQL также доступны в виде отдельных переменных, если они не конфликтуют с существующими переменными. Если аргумент имеет значение SQL NULL, соответствующая переменная будет [undef](#). Для функций, возвращающих несколько значений, можно возвращать ссылку на массив, ссылку на хеш или, в контексте списка, простой список.

```
CREATE FUNCTION numbers(integer) RETURNS SETOF integer AS $$
  for my $i (1 .. $_[0]) {
    return_next($i);
  }
  return undef;
$$ LANGUAGE plperl;
```

Для возврата результирующего набора функция может возвращать ссылку на массив строк, где каждая строка является ссылкой на хеш, или она может использовать [return_next](#) для потоковой передачи строк.

```
CREATE FUNCTION audit_trigger() RETURNS trigger AS $$
  my $row = $_TD->{new} // $_TD->{old};
  elog(NOTICE, "$_TD->{when} $_TD->{event} on id=$row->{id}");
  return undef;
$$ LANGUAGE plperl;
```

Триггеры PL/Perl получают свои данные через [$_TD](#). Поле [$_TD->{new}](#) содержит новую строку для операций INSERT и UPDATE, а [$_TD->{old}](#) содержит старую строку для UPDATE и DELETE. Поле [$_TD->{event}](#) сообщает, какая операция вызвала триггер, а [$_TD->{when}](#) указывает, является ли это BEFORE, AFTER или INSTEAD OF. Триггеры могут изменять строку, возвращая измененную ссылку на хеш, или они могут пропустить операцию, возвращая [undef](#).

```
CREATE FUNCTION block_drop() RETURNS event_trigger AS $$
  if ($_TD->{tag_name} =~ /^DROP/) {
    elog(ERROR, "DROP commands are not allowed");
  }
$$ LANGUAGE plperl;

CREATE EVENT TRIGGER no_drop ON ddl_command_start
  EXECUTE FUNCTION block_drop();
```

Триггеры событий PL/Perl получают имя события в [$_TD->{event}](#) и имя тега в [$_TD->{tag_name}](#). Они полезны для аудита изменений схемы или ограничения того, какие команды DDL разрешены.

```
CREATE FUNCTION count_rows(text) RETURNS integer AS $$
  my $rv = spi_exec_query("SELECT count(*) AS n FROM $_[0]");
  return $rv->{rows}[0]->{n};
$$ LANGUAGE plperl;
```

Функции PL/Perl могут выполнять запросы SQL с помощью [spi_exec_query](#) для буферизованных результатов или [spi_query](#) / [spi_fetch](#) для потоковой передачи больших наборов. Функция [spi_prepare](#) создает подготовленные операторы, которые можно повторно использовать с [spi_exec_prepared](#). Всегда освобождайте курсоры с помощью [spi_cursor_close](#), когда закончите.

Данные в базе данных и на языке PL/Perl

Полезные скалярные строки

Скалярные строки PL/Perl отображаются на типы SQL text и varchar. Значения SQL NULL отображаются в [undef](#). Любой тип данных, который имеет строковое представление, может быть передан в функцию PL/Perl как строка, и любой результат может быть возвращен как строка, которую PostgreSQL затем проанализирует.

```
CREATE FUNCTION trim_both(text) RETURNS text AS $$
  my $s = shift;
  $s =~ s/^\s+|\s+$//g;
  return $s;
$$ LANGUAGE plperl;
```

Получение большого набора строк

Скалярные строки PL/Perl отображаются на типы SQL text и varchar. Значения SQL NULL отображаются в [undef](#). Любой тип данных, который имеет строковое представление, может быть передан в функцию PL/Perl как строка, и любой результат может быть возвращен как строка, которую PostgreSQL затем проанализирует.

```
CREATE FUNCTION trim_both(text) RETURNS text AS $$
  my $s = shift;
  $s =~ s/^\s+|\s+$//g;
  return $s;
$$ LANGUAGE plperl;
```

Полезные массивы

Массивы PostgreSQL передаются в PL/Perl как ссылки на массивы Perl. Многомерные массивы отображаются на вложенные ссылки на массивы. Для возврата массивов функция может возвращать ссылку на массив, и PL/Perl преобразует ее обратно в массив PostgreSQL.

```
CREATE FUNCTION sum_array(integer[]) RETURNS integer AS $$
  my $arr = shift;
  my $sum = 0;
  $sum += $_ for @$arr;
  return $sum;
$$ LANGUAGE plperl;

SELECT sum_array(ARRAY[1,2,3,4]);
```

Массивы PostgreSQL передаются в PL/Perl как ссылки на массивы Perl. Многомерные массивы отображаются на вложенные ссылки на массивы. Для возврата массивов функция может возвращать ссылку на массив, и PL/Perl преобразует ее обратно в массив PostgreSQL.

Использование предварительно подготовленных планов запросов

Для часто выполняемых запросов [spi_prepare](#) может кэшировать план запроса, чтобы избежать повторного анализа. Подготовленный план сохраняется в переменной Perl и может быть повторно использован в последующих вызовах функции. Используйте [spi_exec_prepared](#) с подготовленным планом и списком параметров.

```
CREATE FUNCTION lookup_user(integer) RETURNS text AS $$
  my $plan = $_SHARED{plan} //= spi_prepare('SELECT name FROM users WHERE id = $1', 'integer');
  my $rv = spi_exec_prepared($plan, $_[0]);
  return $rv->{rows}[0]->{name};
$$ LANGUAGE plperl;
```

Управление транзакциями

Функции PL/Perl не могут напрямую управлять транзакциями, но они могут использовать [SAVEPOINT](#) через SPI. Функция [spi_exec_query("SAVEPOINT sp1")](#) создает точку сохранения, [spi_exec_query("ROLLBACK TO sp1")](#) откатывается к ней, а [spi_exec_query("RELEASE sp1")](#) освобождает ее. Это позволяет функции восстанавливаться после частичных сбоев, не прерывая всю транзакцию.

```
CREATE FUNCTION safe_insert(integer) RETURNS boolean AS $$
  spi_exec_query("SAVEPOINT sp1");
  my $rv = spi_exec_query("INSERT INTO t(id) VALUES ($1)", 0, $_[0]);
  if ($rv->{status} eq 'SPI_ERROR') {
    spi_exec_query("ROLLBACK TO sp1");
    return 0;
  }
  spi_exec_query("RELEASE sp1");
  return 1;
$$ LANGUAGE plperl;
```

Встроенные статические функции

Функция	Описание
[elog(level, msg)](#)	Записывает сообщение в журнал сервера. Уровни: [DEBUG](#), [LOG](#), [INFO](#), [NOTICE](#), [WARNING](#), [ERROR](#). [ERROR](#) прерывает текущую транзакцию.
[quote_literal(str)](#)	Возвращает строку, заключенную в кавычки, подходящую для использования в качестве литерала SQL. Возвращает [undef](#), если [str](#) равно [undef](#).
[quote_nullable(str)](#)	Как [quote_literal](#), но возвращает строку [NULL](#), если [str](#) равно [undef](#).
[quote_ident(str)](#)	Возвращает строку, заключенную в кавычки, подходящую для использования в качестве идентификатора SQL, такого как имя таблицы или столбца.
[decode_bytea(str)](#)	Декодирует шестнадцатеричное представление данных [bytea](#).
[encode_bytea(str)](#)	Кодирует данные [bytea](#) в их шестнадцатеричное представление.
[encode_array_literal(arr)](#)	Преобразует ссылку на массив в строковое представление массива PostgreSQL.
[encode_array_literal(arr, delim)](#)	Как [encode_array_literal](#), но использует [delim](#) в качестве разделителя между элементами.
[encode_typed_literal(val, type)](#)	Преобразует значение Perl в строковое представление указанного типа PostgreSQL.
[looks_like_number(str)](#)	Возвращает истину, если [str](#) выглядит как допустимое числовое значение.
[spi_exec_query(query)](#)	Выполняет запрос SQL и возвращает все строки в виде ссылки на хеш.
[spi_exec_query(query, limit)](#)	Как [spi_exec_query](#), но ограничивает количество возвращаемых строк.
[spi_query(query)](#)	Начинает потоковый запрос SQL и возвращает курсор.
[spi_fetch(cursor)](#)	Получает следующую строку из курсора, возвращенного [spi_query](#).
[spi_cursor_close(cursor)](#)	Освобождает курсор, возвращенный [spi_query](#).
[spi_prepare(query, types...)](#)	Подготавливает оператор SQL с заполнителями [$1](#), [$2](#) и т. д.
[spi_exec_prepared(plan, args...)](#)	Выполняет подготовленный план с указанными значениями параметров.
[spi_prepare(query, types...)](#)	Подготавливает оператор SQL с заполнителями [$1](#), [$2](#) и т. д.
[spi_exec_prepared(plan, args...)](#)	Выполняет подготовленный план с указанными значениями параметров.
[spi_exec_prepared(plan, {limit => n}, args...)](#)	Как [spi_exec_prepared](#), но ограничивает количество возвращаемых строк.
[spi_exec_prepared(plan, {binary => 1}, args...)](#)	Как [spi_exec_prepared](#), но возвращает значения в двоичном формате.
[spi_exec_prepared(plan, {limit => n, binary => 1}, args...)](#)	Как [spi_exec_prepared](#), но ограничивает количество возвращаемых строк.
[spi_exec_prepared(plan, {limit => n, binary => 1}, args...)](#)	Как [spi_exec_prepared](#), но возвращает значения в двоичном формате.
[spi_exec_prepared(plan, {limit => n, binary => 1}, args...)](#)	Как [spi_exec_prepared](#), но ограничивает количество возвращаемых строк.
[spi_exec_prepared(plan, {limit => n, binary => 1}, args...)](#)	Как [spi_exec_prepared](#), но возвращает значения в двоичном формате.

Использование PL/Tcl

Автор: Евгений Смирнов

AD.PG

Назад к содержанию

Содержание

Обзор
Аргументы и функции PL/Tcl
Глобальные данные
Обращение к базе данных из PL/Tcl
spi_exec
spi_execp
quote
subtransaction
elog

Обзор

PL/Tcl — серверный процедурный язык для СУБД PostgreSQL, который позволяет использовать язык Tcl для написания функций и процедур.

Для создания функции PL/Tcl используется команда CREATE FUNCTION, в которой указывается, что функция будет написана на языке pltcl. Тело функции — это просто исходный код на языке Tcl. В документации к PostgreSQL рекомендуется заключать тело функции в доллар-кавычки (dollar-quoting), чтобы не приходилось экранировать одинарные кавычки внутри функции. Подробнее о доллар-кавычках можно прочитать в документации PostgreSQL.

```
CREATE FUNCTION имя_функции(аргумент1 тип1, аргумент2 тип2, ...)
RETURNS возвращаемый_тип
AS $$
    # Код функции на Tcl
$$ LANGUAGE pltcl;
```

Вместо $$ можно использовать любой другой тег, например $body$. Главное — чтобы открывающий и закрывающий теги совпадали.

Если функция возвращает значение, оно передаётся командой return. Если функция ничего не возвращает, можно указать RETURNS void или вообще опустить RETURNS.

Пример простой функции, которая складывает два числа:

```
CREATE FUNCTION add_numbers(a integer, b integer) RETURNS integer AS $$
    return [expr $1 + $2]
$$ LANGUAGE pltcl;
```

Аргументы функции доступны внутри тела как переменные $1, $2 и так далее — по порядку, в том порядке, в котором они были перечислены при создании функции. Переменная $1 содержит значение первого аргумента, $2 — второго, и так далее.

Вызвать эту функцию можно так:

```
SELECT add_numbers(3, 5);
```

Результат:

```
 add_numbers
-------------
           8
(1 row)
```

Также можно объявить имена для аргументов, например:

```
CREATE FUNCTION add_numbers_named(first integer, second integer) RETURNS integer AS $$
    return [expr $first + $second]
$$ LANGUAGE pltcl;
```

Теперь вместо $1 и $2 можно использовать first и second, что делает код более читаемым.

Функция может возвращать не только числа, но и строки, булевы значения, массивы и составные типы. Для этого достаточно указать нужный тип в RETURNS и вернуть значение командой return. Например, функция, возвращающая приветствие:

```
CREATE FUNCTION greet(name text) RETURNS text AS $$
    return "Привет, $name!"
$$ LANGUAGE pltcl;
```

```
SELECT greet('Мир');
```

Результат:

```
    greet
-------------
 Привет, Мир!
(1 row)
```

Функция может возвращать несколько строк. Для этого нужно вызвать return_next для каждой строки, а в конце — return. Пример функции, которая возвращает числа от 1 до заданного:

```
CREATE FUNCTION count_to(n integer) RETURNS SETOF integer AS $$
    for {set i 1} {$i <= $n} {incr i} {
        return_next $i
    }
$$ LANGUAGE pltcl;
```

```
SELECT * FROM count_to(5);
```

Результат:

```
 count_to
----------
        1
        2
        3
        4
        5
(5 rows)
```

Если нужно вернуть несколько столбцов, используется return_next с массивом, где ключи — имена столбцов:

```
CREATE FUNCTION get_user(user_id integer) RETURNS TABLE (id integer, name text, email text) AS $$
    spi_exec -array row "SELECT id, name, email FROM users WHERE id = $user_id" {
        return_next [array get row]
    }
$$ LANGUAGE pltcl;
```

```
SELECT * FROM get_user(1);
```

Результат:

```
 id |  name  |      email
----+--------+------------------
  1 | Алиса  | alice@mail.ru
(1 row)
```

Функция может возвращать NULL, если нужно явно указать отсутствие значения. Для этого используется return_null:

```
CREATE FUNCTION find_user(user_id integer) RETURNS text AS $$
    spi_exec "SELECT name FROM users WHERE id = $user_id"
    if {$SPI_processed == 0} {
        return_null
    }
    return $name
$$ LANGUAGE pltcl;
```

```
SELECT find_user(999);
```

Результат:

```
 find_user
-----------

(1 row)
```

Обратите внимание, что результат выглядит как пустая строка, а не как NULL. Чтобы проверить, действительно ли функция вернула NULL, можно использовать IS NULL:

```
SELECT find_user(999) IS NULL;
```

Результат:

```
 ?column?
----------
 t
(1 row)
```

Теперь результат — t (true), что подтверждает, что функция действительно вернула NULL.

Аргументы и функции PL/Tcl

Чтобы создать функцию на языке PL/Tcl используется стандартная команда CREATE FUNCTION.

```
CREATE FUNCTION название_функции(аргумент1 тип1, аргумент2 тип2, ...)
RETURNS возвращаемый_тип
AS $$
    # Код функции на Tcl
$$ LANGUAGE pltcl;
```

Тело функции — это обычный скрипт на Tcl. При вызове функции аргументы передаются в переменные $1, $2 и так далее. Результат возвращается с помощью return.

Функция PL/Tcl может вернуть скалярное значение (число, строку, булево значение), составной тип (строку таблицы) или несколько строк. Для этого используется return_next.

Пример функции, которая складывает два числа:

```
CREATE FUNCTION add_numbers(a integer, b integer) RETURNS integer AS $$
    return [expr $1 + $2]
$$ LANGUAGE pltcl;
```

```
SELECT add_numbers(3, 5);
```

Результат:

```
 add_numbers
-------------
           8
(1 row)
```

Также можно объявить имена для аргументов, например:

```
CREATE FUNCTION add_numbers_named(first integer, second integer)
RETURNS integer AS $$
    return [expr $first + $second]
$$ LANGUAGE pltcl;
```

Теперь вместо $1 и $2 можно использовать first и second, что делает код более читаемым.

Функция может возвращать не только числа, но и строки, булевы значения, массивы и составные типы. Для этого достаточно указать нужный тип в RETURNS и вернуть значение командой return. Например, функция, возвращающая приветствие:

```
CREATE FUNCTION greet(name text) RETURNS text AS $$
    return "Привет, $name!"
$$ LANGUAGE pltcl;
```

```
SELECT greet('Мир');
```

Результат:

```
    greet
-------------
 Привет, Мир!
(1 row)
```

Функция может возвращать несколько строк. Для этого нужно вызвать return_next для каждой строки, а в конце — return. Пример функции, которая возвращает числа от 1 до заданного:

```
CREATE FUNCTION count_to(n integer) RETURNS SETOF integer AS $$
    for {set i 1} {$i <= $n} {incr i} {
        return_next $i
    }
$$ LANGUAGE pltcl;
```

```
SELECT * FROM count_to(5);
```

Результат:

```
 count_to
----------
        1
        2
        3
        4
        5
(5 rows)
```

Если нужно вернуть несколько столбцов, используется return_next с массивом, где ключи — имена столбцов:

```
CREATE FUNCTION get_user(user_id integer)
RETURNS TABLE (id integer, name text, email text) AS $$
    spi_exec -array row "SELECT id, name, email FROM users WHERE id = $user_id" {
        return_next [array get row]
    }
$$ LANGUAGE pltcl;
```

```
SELECT * FROM get_user(1);
```

Результат:

```
 id |  name  |      email
----+--------+------------------
  1 | Алиса  | alice@mail.ru
(1 row)
```

Функция PL/Tcl также может возвращать результат одного запроса. Для этого нужно вызвать spi_exec, передав ему текст запроса. Если запрос возвращает несколько строк, они будут обработаны по очереди. Если запрос ничего не возвращает, функция вернёт NULL.

```
CREATE FUNCTION get_user_email(user_id integer)
RETURNS text AS $$
    spi_exec "SELECT email FROM users WHERE id = $user_id"
    return $email
$$ LANGUAGE pltcl;
```

```
SELECT get_user_email(1);
```

Результат:

```
 get_user_email
----------------
 alice@mail.ru
(1 row)
```

Если запрос возвращает несколько строк, можно использовать цикл foreach:

```
CREATE FUNCTION get_all_emails()
RETURNS SETOF text AS $$
    spi_exec "SELECT email FROM users"
    foreach email $email {
        return_next $email
    }
$$ LANGUAGE pltcl;
```

```
SELECT * FROM get_all_emails();
```

Результат:

```
 get_all_emails
----------------
 alice@mail.ru
 bob@mail.ru
 carol@mail.ru
(3 rows)
```

Функция может возвращать NULL, если нужно явно указать отсутствие значения. Для этого используется return_null:

```
CREATE FUNCTION find_user(user_id integer)
RETURNS text AS $$
    spi_exec "SELECT name FROM users WHERE id = $user_id"
    if {$SPI_processed == 0} {
        return_null
    }
    return $name
$$ LANGUAGE pltcl;
```

```
SELECT find_user(999);
```

Результат:

```
 find_user
-----------

(1 row)
```

Обратите внимание, что результат выглядит как пустая строка, а не как NULL. Чтобы проверить, действительно ли функция вернула NULL, можно использовать IS NULL:

```
SELECT find_user(999) IS NULL;
```

Результат:

```
 ?column?
----------
 t
(1 row)
```

Теперь результат — t (true), что подтверждает, что функция действительно вернула NULL.

Глобальные данные

Иногда возникает необходимость сохранить данные между вызовами функции или совместно использовать данные между несколькими функциями. Это можно сделать с помощью глобальных переменных Tcl. Такие переменные доступны во всех функциях, выполняющихся в рамках одной сессии. Для этого используется команда global, которая объявляет переменную глобальной. Например, функция counter будет увеличивать значение переменной $::counter при каждом вызове:

```
CREATE FUNCTION counter() RETURNS integer AS $$
    global counter
    if {![info exists counter]} {
        set counter 0
    }
    incr counter
    return $counter
$$ LANGUAGE pltcl;
```

```
SELECT counter(), counter(), counter();
```

Результат:

```
 counter | counter | counter
---------+---------+---------
       1 |       2 |       3
(1 row)
```

Глобальные переменные сохраняются между вызовами функций в рамках одной сессии, но не между разными сессиями. Это означает, что если вы подключитесь к базе данных в другом окне терминала, переменная counter будет начинаться с 1. Также стоит помнить, что глобальные переменные могут привести к неожиданному поведению, если несколько функций используют одно и то же имя переменной. Поэтому рекомендуется использовать уникальные имена или пространства имён.

Обращение к базе данных из PL/Tcl

Для выполнения SQL-запросов в PL/Tcl используются команды spi_exec и spi_execp. Они позволяют выполнять запросы и получать результаты. Рассмотрим их подробнее.

spi_exec

Выполняет команду SQL, переданную в виде строки. Синтаксис SQL команды проверяется и выполняется сразу. Результаты можно сохранить в переменную, указанную после имени команды. Например, spi_exec "SELECT * FROM users" выполнит запрос и сохранит результаты в переменную SPI_result.

```
spi_exec ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_exec "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

Обращение к базе данных из PL/Tcl

Для выполнения SQL-запросов в PL/Tcl используются команды spi_exec и spi_execp. В этом разделе я расскажу подробно о синтаксисе, параметрах и примерах использования этих команд. Начнём с spi_exec.

spi_exec

Выполняет команду SQL, переданную в виде строки. Синтаксис SQL команды проверяется и выполняется сразу. Результаты можно сохранить в переменную, указанную после имени команды. Например, spi_exec "SELECT * FROM users" выполнит запрос и сохранит результаты в переменную SPI_result.

```
spi_exec ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_exec "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

spi_execp

Функция spi_execp работает аналогично spi_exec, но поддерживает подготовленные операторы. Подготовленный оператор — это заранее подготовленный запрос, который можно выполнять многократно с разными параметрами. Это ускоряет работу, если один и тот же запрос выполняется много раз.

```
spi_execp ?-count num? ?-array name? ?-array_numeric? ?-interp name? ?-variable? query ?loop_body?
```

Где:

- -count num — ограничивает количество возвращаемых строк числом num.
- -array name — сохраняет результаты в ассоциативный массив с именем name. Ключами будут имена столбцов, а значениями — данные.
- -array_numeric — аналогично -array, но ключами будут числовые индексы столбцов, начиная с 0.
- -interp name — выполняет команду в указанном интерпретаторе name.
- -variable — сохраняет результаты в переменную SPI_result (по умолчанию).
- query — текст SQL-запроса.
- loop_body — код, который будет выполнен для каждой строки результата.

Пример:

```
spi_execp "SELECT id, name FROM users WHERE active = true" {
    puts "ID: $id, Name: $name"
}
```

Здесь id и name — имена столбцов, которые автоматически становятся переменными в теле цикла.

quote

Функция quote добавляет или удаляет кавычки вокруг строки, экранируя специальные символы. Это полезно, когда нужно вставить данные в строку запроса.

```
quote строка
```

Пример:

```
quote "O'Brien"
```

Результат: O''Brien

Вызов quote "O'Brien" эквивалентен quote {O'Brien}.

В документации говорится, что SELECT 'O''Brien' = 'O''Brien' AS t; будет true (экранирование) и SELECT 'O'Brien' = 'O''Brien' AS t; — false.

subtransaction

Функция subtransaction позволяет указанную команду в рамках подтранзакции SQL.

```
subtransaction командная_строка
```

Где командная_строка — скрипт Tcl.

Если скрипт Tcl возвращает ошибку, подтранзакция откатывается, в противном случае — фиксируется.

Пример:

```
CREATE TABLE accounts(
    id integer PRIMARY KEY,
    balance numeric
);

CREATE TABLE transfer_log(
    t_time timestamp DEFAULT now(),
    msg text
);

INSERT INTO accounts VALUES
    (1,100.00),
    (2,200.00);

CREATE FUNCTION account_transfer() RETURNS void AS $$
    set from_id 1
    set to_id 2
    set amount '150.00'
    spi_exec "UPDATE accounts SET balance = balance - 150 WHERE id=$from_id"
    spi_exec "UPDATE accounts SET balance = balance + 150 WHERE id=$to_id"
    if {$to_id == 2} {
        error "Не хватает денег на счёте $from_id для перевода $amount"
    }
    spi_exec "INSERT INTO transfer_log(msg) VALUES ('Сделать $result')"
$$ LANGUAGE pltcl;

SELECT account_transfer();
SELECT * FROM accounts;
```

Результат:

```
 id | balance
----+---------
  1 |  100.00
  2 |  200.00
(2 rows)
```

elog

Функция elog выдаёт пользовательское или системное сообщение.

```
elog уровень сообщение
```

Где:

- уровень — уровень серьезности сообщения. Возможные значения: DEBUG, LOG, INFO, NOTICE, WARNING, ERROR, FATAL.
- сообщение — текст сообщения.

Пример:

```
elog DEBUG "The database contains table $table_name"
```

Команда DEBUG пишет сообщение в лог. Если задано log_min_messages равное DEBUG или ниже, сообщение будет также передано клиенту. Команда ERROR прерывает текущую транзакцию и возвращает управление в самый внешний уровень. Команда FATAL прерывает сеанс с сервером PL/Tcl, но не базой данных. Команда elog может иметь дополнительные параметры, которые влияют на детали отправленного сообщения, но они не связаны с PL/Tcl, поэтому здесь не документируются. Обратитесь к документации PostgreSQL.

Назад к содержанию

Шифрование с помощью pgcrypto

Alexander Korotkov

Postgres Professional

Содержание

- Ограничение безопасности
- Установка pgcrypto
- Поддерживаемые алгоритмы
- Функции pgcrypto
  - Стандартные функции хеширования
    - Функция хеширования паролей
    - Функции шифрования PGP
    - Генерация случайных данных
    - Функции получения случайных данных

Postgres Professional

Postgres Professional

Вы можете использовать расширение pgcrypto для шифрования данных в базе данных Postgres. Данное расширение предоставляет функции для хеширования паролей, общего хеширования и шифрования. Поддерживаются как симметричное, так и асимметричное шифрование данных. Данная статья не является введением в криптографию, а предполагает, что читатель уже знает, что такое PGP, симметричные и асимметричные ключи и т.п.

Ограничение безопасности

Все функции работают внутри сервера баз данных. Это означает, что весь трафик между клиентом и сервером PostgreSQL может быть виден в открытом виде, если не используется SSL. Также сервер может видеть открытые ключи и пароли. Если это неприемлемо, то криптографию необходимо выполнять на стороне клиента.

Установка pgcrypto

Пустая версия pgcrypto включена в contrib, которая поставляется вместе с сервером баз данных PostgreSQL.

Чтобы установить pgcrypto, выполните команду CREATE EXTENSION:

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
```

Поддерживаемые алгоритмы

Поддерживаются следующие алгоритмы:

- MD5
- SHA1
- SHA224/256/384/512
- алгоритмы семейства Blowfish, поддерживаются OpenSSL, включены в версии pgcrypto, поставляемой с Postgres Pro
- AES
- DES/3DES/CAST5
- нейтральный генератор данных
- алгоритмы семейства Modular Crypt Format
- PGP шифрование и спецификации OpenPGP

Функции pgcrypto

Функции pgcrypto описаны в документации PostgreSQL. Здесь мы приведем краткое описание и несколько примеров.

Стандартные функции хеширования

Пример:

```
digest(data text, type text) returns bytea
digest(data bytea, type text) returns bytea
```

```
CREATE TABLE tbltest( key_code int4, data bytea );

INSERT INTO tbltest VALUES(1, digest('Hello World', 'sha1'));
SELECT
    key_code,
    digest(data, 'sha1') AS "sha1",
    data
FROM tbltest;

DROP TABLE tbltest;
```

```
CREATE TABLE tbltest( key_code int4, data bytea );
INSERT INTO tbltest VALUES(1, 'Hello World');
SELECT
    key_code,
    digest(data, 'sha1') AS "sha1",
    data
FROM tbltest;

DROP TABLE tbltest;
```

```
key_code | sha1                                       | data
---------+--------------------------------------------+-------------
       1 | \x0a4d55a8d778e5022fab701977c03a5b7e7832b1 | Hello World
```

```
key_code | sha1                                       | data
---------+--------------------------------------------+-------------
       1 | \x0a4d55a8d778e5022fab701977c03a5b7e7832b1 | \x48656c6c6f20576f726c64
```

```
digest(data text, type text) returns bytea
digest(data bytea, type text) returns bytea
```

Функция хеширования паролей

Функции crypt() и gen_salt() специально разработаны для хеширования паролей. Функция crypt() выполняет хеширование, а gen_salt() подготавливает параметры для неё.

Алгоритм хеширования паролей crypt() отличается от обычных алгоритмов хеширования:

- Он медленный. Так как задача решается подбором паролей, то чем медленнее алгоритм, тем сложнее подобрать пароль.
- Он использует случайное значение, называемое солью, поэтому два одинаковых пароля дают разный результат.
- Он имеет настраиваемый коэффициент, определяющий, насколько медленным будет алгоритм.

Функции crypt() и gen_salt() поддерживают следующие алгоритмы:

Название	Максимальная длина пароля	Адаптивность	Размер соли (бит)	Длина хеширования	Описание
bf	72	да	128	60	На базе Blowfish, вариант 2a
md5	без ограничений	нет	48	34	На базе MD5
xdes	8	да	24	20	Расширенный DES
des	8	нет	12	13	Классический DES
ext	8	да	24	20	Расширенный DES

crypt()

```
crypt(password text, salt text) returns text
```

Шифрует password, используя salt. Результат может быть сохранен в базе данных. Позже crypt() может использоваться для проверки пароля, сравнивая сохраненное значение с результатом crypt() для введенного пароля.

```
UPDATE tbltest SET password_hash = crypt('new_password', gen_salt('md5'));
```

Запрос, проверяющий имя пользователя и пароль:

```
SELECT (password_hash = crypt('entered_password', password_hash)) AS password_match FROM tbltest WHERE name=...;
```

gen_salt()

```
gen_salt(type text) returns text
gen_salt(type text, iter_count integer) returns text
```

Генерирует новое случайное значение соли для использования в crypt(). Функция gen_salt() подготавливает параметры для crypt(). Параметр type определяет используемый алгоритм. Параметр iter_count позволяет пользователю определить, насколько медленным будет алгоритм crypt() (если поддерживается). Чем больше это значение, тем больше времени потребуется.

Алгоритм	Значение по умолчанию	Минимум	Максимум
bf	6	4	31
md5	1000	1	[2^31]-1
xdes	725	1	16777215
des	25	25	25

Функции шифрования PGP

Эти функции реализуют часть стандарта OpenPGP. Они позволяют шифровать данные симметричными ключами и ключами открытого шифрования. Поддерживаются 3 типа данных:

- Симметричный ключ: быстрый, использует только пароль.
- Открытый ключ: медленный, использует пару ключей.
- Текст с цифровой подписью: открытый ключ, используется для проверки целостности, но не для секретности.

Поддерживаемые возможности PGP включают все стандартные функции PGP:

- Шифрование открытым ключом и симметричным ключом.
- Различные длины ключей. (Поддерживаются ключи любой длины.)
- Защита ключей паролем.
- Подписывание, проверка подписи.
- Сжатие.
- Кодирование в ASCII.
- Использование именованных смещений.
- Генерация ключей. (Не рекомендуется для медленных платформ, таких как 486.)

Не поддерживаются:

- Подписывание ключей (ключи подписываются, но подписи ключей не проверяются).
- Несколько подключей.
- Различные предпочтения (сжатие, алгоритмы шифрования).
- Ключевые серверы.

```
pgp_sym_encrypt(data text, psw text) returns bytea
pgp_sym_encrypt(data text, psw text, options text) returns bytea
```

Название	Описание	Значение по умолчанию	Допустимые значения
cipher-algo	Алгоритм шифрования	aes128	bf, aes128, aes192, aes256, 3des, cast5, blowfish, idea, none, aes, twofish, camellia128, camellia192, camellia256
compress-algo	Алгоритм сжатия	0	0 = без сжатия, 1 = сжатие ZIP, 2 = сжатие ZLIB
compress-level	Уровень сжатия. Чем больше значение, тем выше степень сжатия. 0 означает, что данные не будут сжаты.	6	0-9
s2k-mode	Алгоритм S2K	3	0 = не использовать соль S2K, 1 = использовать только соль, 3 = использовать соль и итерации
s2k-digest-algo	Алгоритм дайджеста, используемый в S2K	sha1	md5, sha1
s2k-cipher-algo	Алгоритм шифрования, используемый для шифрования ключей S2K	использовать тот же, что и cipher-algo	bf, aes, aes128, aes192, aes256
unicode-mode	Определяет, преобразовать ли текст в UTF-8 при шифровании. После расшифровки данные будут преобразованы из UTF-8 в исходную кодировку сервера. Полезно, если клиент использует другую кодировку.	0	0 = не использовать преобразование, 1 = использовать преобразование

Пример:

```
CREATE TABLE users( user_id serial PRIMARY KEY, username VARCHAR(50), password TEXT);

INSERT INTO users(username, password) VALUES ('Peter', pgp_sym_encrypt('password', 'key_key', 'compress-algo=1, cipher-algo=aes256'));
```

pgp_sym_decrypt()

```
pgp_sym_decrypt(msg bytea, psw text) returns text
pgp_sym_decrypt(msg bytea, psw text, options text) returns text
```

Расшифровывает данные, зашифрованные pgp_sym_encrypt(). Параметр options имеет такой же формат, как и для pgp_sym_encrypt().

Название	Описание	Значение по умолчанию	Допустимые значения
cipher-algo	Алгоритм шифрования	aes128	bf, aes128, aes192, aes256, 3des, cast5, blowfish, idea, none, aes, twofish, camellia128, camellia192, camellia256
compress-algo	Алгоритм сжатия	0	0 = без сжатия, 1 = сжатие ZIP, 2 = сжатие ZLIB
compress-level	Уровень сжатия. Чем больше значение, тем выше степень сжатия. 0 означает, что данные не будут сжаты.	6	0-9
unicode-mode	Определяет, преобразовать ли текст в UTF-8 при шифровании. После расшифровки данные будут преобразованы из UTF-8 в исходную кодировку сервера. Полезно, если клиент использует другую кодировку.	0	0 = не использовать преобразование, 1 = использовать преобразование

Пример:

```
SELECT user_id, pgp_sym_decrypt(password::bytea, 'key_key') AS password
FROM users WHERE username = 'Peter';
```

pgp_pub_encrypt()

```
pgp_pub_encrypt(data text, key bytea) returns bytea
pgp_pub_encrypt(data text, key bytea, options text) returns bytea
```

Шифрует данные открытым ключом. Этой функции не нужен пароль, поскольку для каждого сеанса генерируется случайный ключ. Однако для ключа должен быть указан подходящий открытый ключ.

Название	Описание	Значение по умолчанию	Допустимые значения
cipher-algo	Алгоритм шифрования	aes128	bf, aes128, aes192, aes256, 3des, cast5, blowfish, idea, none, aes, twofish, camellia128, camellia192, camellia256
compress-algo	Алгоритм сжатия	0	0 = без сжатия, 1 = сжатие ZIP, 2 = сжатие ZLIB
compress-level	Уровень сжатия. Чем больше значение, тем выше степень сжатия. 0 означает, что данные не будут сжаты.	6	0-9
unicode-mode	Определяет, преобразовать ли текст в UTF-8 при шифровании. После расшифровки данные будут преобразованы из UTF-8 в исходную кодировку сервера. Полезно, если клиент использует другую кодировку.	0	0 = не использовать преобразование, 1 = использовать преобразование

Пример:

```
INSERT INTO users (username, password, public_key) VALUES ('Peter', pgp_sym_encrypt('password', 'key_key'), pgp_pub_encrypt('password', dearmor('-----BEGIN PGP PUBLIC KEY BLOCK-----
...
-----END PGP PUBLIC KEY BLOCK-----')));
```

pgp_pub_decrypt()

```
pgp_pub_decrypt(msg bytea, key bytea) returns text
pgp_pub_decrypt(msg bytea, key bytea, psw text) returns text
pgp_pub_decrypt(msg bytea, key bytea, psw text, options text) returns text
```

Расшифровывает данные, зашифрованные pgp_pub_encrypt(). Для ключа должен быть указан подходящий закрытый ключ. Если ключ защищен паролем, то он должен быть указан в psw. Параметр options имеет такой же формат, как и для pgp_sym_encrypt().

Название	Описание	Значение по умолчанию	Допустимые значения
unicode-mode	Определяет, преобразовать ли текст в UTF-8 при шифровании. После расшифровки данные будут преобразованы из UTF-8 в исходную кодировку сервера. Полезно, если клиент использует другую кодировку.	0	0 = не использовать преобразование, 1 = использовать преобразование

Пример:

```
SELECT user_id, pgp_pub_decrypt(password, public_key, 'key_key') AS password
FROM users WHERE username = 'Peter';
```

pgp_key_id()

```
pgp_key_id(bytea) returns text
```

Извлекает идентификатор ключа, использованного для шифрования данных. Возможные результаты:

- SYMKEY - использовался симметричный ключ
- ANYKEY - использовался открытый ключ, но идентификатор ключа не указан

Некоторые сообщения могут быть созданы двумя ключами, поэтому pgp_key_id() может вернуть более одного значения.

```
pgp_key_id(bytea) returns text
```

```
pgp_armor_headers(data text, key out text, value out text) returns setof record
```

Возвращает список ключей и значений, закодированных в ASCII. Ключи, которые не распознаются, будут возвращены с префиксом Unknown-.

armor(), dearmor()

```
armor(data bytea [, keys text[], values text[]]) returns text
dearmor(data text) returns bytea
```

Эти функции реализуют часть формата PGP ASCII Armor. Дополнительные данные, такие как контрольные суммы, не проверяются.

Генерация случайных данных

```
gen_salt(type text [, iter_count integer]) returns text
```

Функция gen_salt() используется для генерации соли для функции crypt(). Параметр type определяет используемый алгоритм. Параметр iter_count позволяет пользователю определить, насколько медленным будет алгоритм crypt() (если поддерживается). Чем больше это значение, тем больше времени потребуется.

Возвращает следующую соль, готовую к вызову crypt():

```
gen_salt('md5')
```

Эквивалентно вызову функции с параметром по умолчанию:

```
gen_salt('md5', 1000)
```

Создает случайную соль длиной 4 байта:

```
gen_salt('bf', 8)
```

Создает случайную соль длиной 16 байт:

```
gen_salt('bf', 10)
```

Создает случайную соль длиной 24 бита:

```
gen_salt('xdes', 249)
```

Создает случайную соль длиной 12 бит:

```
gen_salt('des')
```

Нетекстовые функции шифрования

Эти функции выполняют симметричное шифрование данных. Они не используют дополнительные возможности шифрования OpenPGP. Поэтому обратите, с какими ограничениями вы работаете:

- Используемые ключи полностью случайные, а не произвольные последовательности символов.
- Для проверки целостности ключа используйте собственный механизм.
- Функции не распознают, какой алгоритм использовался для шифрования данных.
- Наилучший алгоритм по умолчанию - Blowfish.

Низкоуровневые функции шифрования

Эти функции выполняют только шифрование, без дополнительных возможностей. Они используют поля, как двоичные строки.

```
encrypt(data bytea, key bytea, type text) returns bytea
decrypt(data bytea, key bytea, type text) returns bytea
```

```
encrypt_iv(data bytea, key bytea, iv bytea, type text) returns bytea
decrypt_iv(data bytea, key bytea, iv bytea, type text) returns bytea
```

Также `type` имеет следующий формат:

```
algorithm [ - mode ] [ / pad: padname ]
```

В качестве имени примера основанный значения параметров:

Название	Значения
algorithm	bf - Blowfish aes - AES (Rijndael-128, -192 или -256)
mode	cbc - определяется более низкий от предыдущего (значение по умолчанию) ecb - определяется шифрование открытого текста для последующего
padname	pkcs - значение по умолчанию (определяется в RFC 2298) none - заполнение данных должно быть нулями размером кратно размеру

Следующие вызовы функций эквивалентны:

```
encrypt(data, 'fooz', 'bf')
encrypt(data, 'fooz', 'bf-cbc/pad:pkcs')
```

Для функций encrypt_iv и decrypt_iv вектор инициализации, или IV, по умолчанию - это нулевой. Эти функции позволяют переопределить это значение. Если предоставленный IV слишком короткий, он будет дополнен нулями.

Функции получения случайных данных

```
gen_random_bytes(count integer) returns bytea
```

Возвращает заданное количество случайных байтов в качестве bytea, до 1024.

Эта функция может быть полезна для создания ключей. Пример:

```
gen_random_uuid() returns uuid
```

Функция генерирует UUID версии 4 (случайный).

Конфигурационные параметры

Разделы параметров

1 Сервер
2 Прокси
3 Агент
4 Агент 2
5 Веб-сервис Java
6 Веб-сервис Zabbix
7 Сервер журналов
8 Прокси журналов
9 Параметры
10 Пользовательские параметры
11 Удаленные команды
12 Настройки SNMP-трапов
13 Примеры
14 Шаблоны

1 Сервер

Файл конфигурации: /etc/zabbix/zabbix_server.conf

Параметры

Параметр	Описание	Значение по умолчанию
AllowRoot	Разрешить сервер запускать от имени root. Если запущен от root и эта опция отключена, сервер попытается переключиться на пользователя, указанного в параметре User.	0

ADPS

Параметр	Описание	Значение по умолчанию
ADPSFile	Полный путь к файлу ADPS.	/var/lib/zabbix/adps.bin

Basic parameters

Параметр	Описание	Значение по умолчанию
AllowUnsupportedDBVersions	Разрешить работу с неподдерживаемыми версиями базы данных.	0

Connection to Vault

Параметр	Описание	Значение по умолчанию
VaultCert	Полный путь к файлу SSL-сертификата клиента, используемого для аутентификации в Vault. Поддерживаются только сертификаты в формате PEM.	
VaultDBPath	Расположение, из которого будут извлекаться учетные данные базы данных, например, secret/zabbix/database. Ключ должен иметь имена свойств password и username.	
VaultKey	Полный путь к файлу SSL-ключа клиента, используемого для аутентификации в Vault. Поддерживаются только ключи в формате PEM.	
VaultTLSCertFile	Полный путь к файлу SSL-сертификата клиента, используемого для аутентификации в Vault. Поддерживаются только сертификаты в формате PEM.	
VaultTLSKeyFile	Полный путь к файлу SSL-ключа клиента, используемого для аутентификации в Vault. Поддерживаются только ключи в формате PEM.	

Database monitor

Параметр	Описание	Значение по умолчанию
DBHost	Имя хоста или IP-адрес базы данных. Если установлено значение localhost, при подключении к MySQL будет использоваться сокет. Если установлено значение пустой строки, при подключении к PostgreSQL будет использоваться сокет.	localhost
DBName	Имя базы данных.	
DBPassword	Пароль базы данных. Комментируйте эту строку, если пароль не используется.	
DBPort	Порт базы данных при использовании не сокета.	3306
DBSchema	Имя схемы используется для PostgreSQL.	
DBSocket	Путь к сокету MySQL.	
DBTLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
DBTLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
DBTLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
DBTLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
DBTLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	
DBUser	Пользователь базы данных.	
DBVerifyHost	Проверяет, что имя хоста, указанное для клиента, совпадает с именем хоста, указанным в сертификате.	0
ExportDir	Каталог, в который будут экспортироваться файлы с историей.	
ExportFileSize	Максимальный размер файлов экспорта в байтах.	1G
ExportType	Список модулей, которые будут экспортировать данные в реальном времени.	
HistoryStorageDateIndex	Включает предварительное создание индексов для таблиц истории.	0
HistoryStorageTypes	Список типов истории, которые будут храниться в хранилищах истории.	
HistoryStorageURL	URL-адрес хранилища истории.	
VaultDBPath	Расположение, из которого будут извлекаться учетные данные базы данных, например, secret/zabbix/database. Ключ должен иметь имена свойств password и username.	

Debug hardware

Параметр	Описание	Значение по умолчанию
StartHistoryPollers	Количество предварительно запущенных процессов опроса истории.	5

Escalator

Параметр	Описание	Значение по умолчанию
StartEscalators	Количество предварительно запущенных процессов эскалации.	5

Ha cluster

Параметр	Описание	Значение по умолчанию
FailoverDelay	Время, которое должно пройти, прежде чем узел будет помечен как недоступный.	60
HANodeName	Имя узла кластера.	
NodeAddress	Адрес узла кластера.	localhost:10051

Housekeeper

Параметр	Описание	Значение по умолчанию
HousekeepingFrequency	Как часто Zabbix будет выполнять процедуру очистки.	1
MaxHousekeeperDelete	Максимальное количество строк, которое будет удалено за одну задачу очистки.	5000

Housekeeping of problems

Параметр	Описание	Значение по умолчанию
DisableHousekeeping	Отключает очистку данных.	0

Housekeeping of trends

Параметр	Описание	Значение по умолчанию
DisableHousekeeping	Отключает очистку данных.	0

Internal monitoring

Параметр	Описание	Значение по умолчанию
StartInternalPollers	Количество предварительно запущенных внутренних процессов опроса.	1

Log files

Параметр	Описание	Значение по умолчанию
DebugLevel	Уровень отладки.	3
LogFile	Имя файла журнала.	
LogFileSize	Максимальный размер файла журнала в МБ.	1
LogType	Тип журнала.	file

Network discovery

Параметр	Описание	Значение по умолчанию
StartDiscoverers	Количество предварительно запущенных процессов обнаружения.	1

Node watcher

Параметр	Описание	Значение по умолчанию
NodeWatcherFrequency	Как часто Zabbix будет проверять состояние узлов кластера.	5

Proxy data sender

Параметр	Описание	Значение по умолчанию
ProxyConfigFrequency	Как часто прокси получает данные конфигурации с сервера.	10
ProxyDataFrequency	Как часто прокси получает данные с сервера.	1

Proxy poller

Параметр	Описание	Значение по умолчанию
ProxyPollers	Количество предварительно запущенных процессов опроса прокси.	1
ProxyUnreachableDelay	Как часто прокси опрашивается после ошибки.	60

Proxy server

Параметр	Описание	Значение по умолчанию
ProxyHeartbeatFrequency	Как часто прокси отправляет heartbeat-сообщения на сервер.	60
ProxyLocalBuffer	Количество часов, в течение которых данные будут храниться локально на прокси.	0
ProxyMode	Режим работы прокси.	0
ProxyOfflineBuffer	Количество часов, в течение которых данные будут храниться на прокси, если связь с сервером потеряна.	1

Self-monitoring

Параметр	Описание	Значение по умолчанию
StartStatusPollers	Количество предварительно запущенных процессов опроса состояния.	1

Snmp trapper

Параметр	Описание	Значение по умолчанию
SNMPTrapperFile	Временный файл, используемый для передачи данных SNMP-трапов от процесса SNMP-трапперов.	/tmp/zabbix_traps.tmp
StartSNMPTrapper	Если установлено значение 1, будет запущен процесс SNMP-траппера.	0

Timeouts

Параметр	Описание	Значение по умолчанию
TrapperTimeout	Время ожидания обработки данных.	300
UnreachablePeriod	После скольких секунд недоступности узел сети будет считаться недоступным.	45

Trigger housekeeper

Параметр	Описание	Значение по умолчанию
DeleteOrphanedProblems	Удалять проблемы, у которых отсутствуют связанные триггеры.	0

Value cache

Параметр	Описание	Значение по умолчанию
ValueCacheSize	Размер кэша значений в байтах.	8M

Vmware monitoring

Параметр	Описание	Значение по умолчанию
StartVMwareCollectors	Количество предварительно запущенных процессов сборщиков VMware.	0
VMwareCacheSize	Размер кэша VMware в байтах.	8M
VMwareFrequency	Как часто Zabbix будет подключаться к сервису VMware.	60
VMwarePerfFrequency	Как часто Zabbix будет собирать данные производительности VMware.	60
VMwareTimeout	Время ожидания, используемое при выполнении SOAP-операций VMware.	10

Zabbix cluster

Параметр	Описание	Значение по умолчанию
ClusterAllowedIP	Список разрешенных IP-адресов, разделенных запятыми, для узлов кластера.	
ClusterDatabaseHost	Имя хоста или IP-адрес базы данных.	
ClusterDatabaseName	Имя базы данных.	
ClusterDatabasePassword	Пароль базы данных.	
ClusterDatabasePort	Порт базы данных при использовании не сокета.	3306
ClusterDatabaseSocket	Путь к сокету MySQL.	
ClusterDatabaseTLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
ClusterDatabaseTLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
ClusterDatabaseTLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
ClusterDatabaseTLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
ClusterDatabaseTLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	
ClusterDatabaseUser	Пользователь базы данных.	
ClusterDatabaseVerifyHost	Проверяет, что имя хоста, указанное для клиента, совпадает с именем хоста, указанным в сертификате.	0
ClusterId	Идентификатор узла кластера.	
ClusterLocalAddress	IP-адрес или DNS-имя для подключения к узлу кластера.	
ClusterLocalPort	Порт для подключения к узлу кластера.	10051
ClusterNodeTimeout	Время, которое должно пройти, прежде чем узел будет помечен как недоступный.	60
ClusterStandbyNodes	Список разрешенных IP-адресов, разделенных запятыми, для резервных узлов кластера.	
ClusterStartupTime	Время, которое должно пройти, прежде чем узел будет помечен как доступный.	0
StartClusterNodeWatchers	Количество предварительно запущенных процессов наблюдения за узлами кластера.	0

Zabbix server

Параметр	Описание	Значение по умолчанию
StartTimers	Количество предварительно запущенных процессов таймеров.	1

Zabbix server high availability

Параметр	Описание	Значение по умолчанию
HANodeName	Имя узла кластера.	
NodeAddress	Адрес узла кластера.	localhost:10051

Zabbix trapper

Параметр	Описание	Значение по умолчанию
StartTrappers	Количество предварительно запущенных процессов трапперов.	5

Zabbix web service

Параметр	Описание	Значение по умолчанию
StartReportWriters	Количество предварительно запущенных процессов генерации отчетов.	0
WebServiceURL	URL-адрес службы веб-отчетов.	

Forks (APS only)

Параметр	Описание	Значение по умолчанию
StartAlerters	Количество предварительно запущенных процессов оповещения.	3
StartConnectors	Количество предварительно запущенных процессов коннекторов.	1
StartDBSyncers	Количество предварительно запущенных процессов синхронизации базы данных.	4
StartDiscoverers	Количество предварительно запущенных процессов обнаружения.	1
StartEscalators	Количество предварительно запущенных процессов эскалации.	5
StartHTTPPollers	Количество предварительно запущенных процессов опроса HTTP.	1
StartHistoryPollers	Количество предварительно запущенных процессов опроса истории.	5
StartIPMIPollers	Количество предварительно запущенных процессов опроса IPMI.	0
StartJavaPollers	Количество предварительно запущенных процессов опроса Java.	0
StartLLDProcessors	Количество предварительно запущенных процессов обработки LLD.	2
StartODBCPollers	Количество предварительно запущенных процессов опроса ODBC.	1
StartPingers	Количество предварительно запущенных процессов пинга.	1
StartPollers	Количество предварительно запущенных процессов опроса.	5
StartPollersUnreachable	Количество предварительно запущенных процессов опроса недоступных узлов сети.	1
StartPreprocessors	Количество предварительно запущенных процессов предобработки.	3
StartSNMPPollers	Количество предварительно запущенных процессов опроса SNMP.	1
StartTimers	Количество предварительно запущенных процессов таймеров.	1
StartTrappers	Количество предварительно запущенных процессов трапперов.	5
StartVMwareCollectors	Количество предварительно запущенных процессов сборщиков VMware.	0

Forks (APS only)

Параметр	Описание	Значение по умолчанию
StartAlerters	Количество предварительно запущенных процессов оповещения.	3
StartConnectors	Количество предварительно запущенных процессов коннекторов.	1
StartDBSyncers	Количество предварительно запущенных процессов синхронизации базы данных.	4
StartDiscoverers	Количество предварительно запущенных процессов обнаружения.	1
StartEscalators	Количество предварительно запущенных процессов эскалации.	5
StartHTTPPollers	Количество предварительно запущенных процессов опроса HTTP.	1
StartHistoryPollers	Количество предварительно запущенных процессов опроса истории.	5
StartIPMIPollers	Количество предварительно запущенных процессов опроса IPMI.	0
StartJavaPollers	Количество предварительно запущенных процессов опроса Java.	0
StartLLDProcessors	Количество предварительно запущенных процессов обработки LLD.	2
StartODBCPollers	Количество предварительно запущенных процессов опроса ODBC.	1
StartPingers	Количество предварительно запущенных процессов пинга.	1
StartPollers	Количество предварительно запущенных процессов опроса.	5
StartPollersUnreachable	Количество предварительно запущенных процессов опроса недоступных узлов сети.	1
StartPreprocessors	Количество предварительно запущенных процессов предобработки.	3
StartSNMPPollers	Количество предварительно запущенных процессов опроса SNMP.	1
StartTimers	Количество предварительно запущенных процессов таймеров.	1
StartTrappers	Количество предварительно запущенных процессов трапперов.	5
StartVMwareCollectors	Количество предварительно запущенных процессов сборщиков VMware.	0

GnuTLS configuration

Параметр	Описание	Значение по умолчанию
GnuTLSPriorities	Строка приоритетов, используемая для настройки GnuTLS.	

Internal TLS configuration

Параметр	Описание	Значение по умолчанию
TLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
TLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
TLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
TLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
TLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	

OpenSSL configuration

Параметр	Описание	Значение по умолчанию
SSLCertLocation	Расположение файлов SSL-сертификатов.	
SSLKeyLocation	Расположение файлов SSL-ключей.	

Parameters for secure communication between Zabbix components

Параметр	Описание	Значение по умолчанию
TLSAccept	Тип соединений, которые должен принимать сервер.	unencrypted
TLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
TLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
TLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
TLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
TLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	
TLSPSKFile	Полный путь к файлу, содержащему ключ PSK.	
TLSPSKIdentity	Идентификатор PSK.	

SSL configuration

Параметр	Описание	Значение по умолчанию
SSLCertLocation	Расположение файлов SSL-сертификатов.	
SSLKeyLocation	Расположение файлов SSL-ключей.	

TLS_CRL

Параметр	Описание	Значение по умолчанию
TLSCRLFile	Полный путь к файлу, содержащему отозванные сертификаты.	

ADPS Control Agents

Параметр	Описание	Значение по умолчанию
ADPSControlAgents	Список разрешенных IP-адресов, разделенных запятыми, для управляющих агентов ADPS.	

ADPS OpenVPN settings

Параметр	Описание	Значение по умолчанию
OpenVPNCipher	Алгоритм шифрования, используемый для OpenVPN.	AES-128-CBC, AES-192-CBC, AES-256-CBC
OpenVPNPort	Порт, используемый для OpenVPN.	1194

2 Прокси

Файл конфигурации: /etc/zabbix/zabbix_proxy.conf

Параметры

Параметр	Описание	Значение по умолчанию
AllowRoot	Разрешить прокси запускать от имени root. Если запущен от root и эта опция отключена, прокси попытается переключиться на пользователя, указанного в параметре User.	0

Basic parameters

Параметр	Описание	Значение по умолчанию
AllowUnsupportedDBVersions	Разрешить работу с неподдерживаемыми версиями базы данных.	0

Connection to Vault

Параметр	Описание	Значение по умолчанию
VaultCert	Полный путь к файлу SSL-сертификата клиента, используемого для аутентификации в Vault. Поддерживаются только сертификаты в формате PEM.	
VaultDBPath	Расположение, из которого будут извлекаться учетные данные базы данных, например, secret/zabbix/database. Ключ должен иметь имена свойств password и username.	
VaultKey	Полный путь к файлу SSL-ключа клиента, используемого для аутентификации в Vault. Поддерживаются только ключи в формате PEM.	
VaultTLSCertFile	Полный путь к файлу SSL-сертификата клиента, используемого для аутентификации в Vault. Поддерживаются только сертификаты в формате PEM.	
VaultTLSKeyFile	Полный путь к файлу SSL-ключа клиента, используемого для аутентификации в Vault. Поддерживаются только ключи в формате PEM.	

Database monitor

Параметр	Описание	Значение по умолчанию
DBHost	Имя хоста или IP-адрес базы данных. Если установлено значение localhost, при подключении к MySQL будет использоваться сокет. Если установлено значение пустой строки, при подключении к PostgreSQL будет использоваться сокет.	localhost
DBName	Имя базы данных.	
DBPassword	Пароль базы данных. Комментируйте эту строку, если пароль не используется.	
DBPort	Порт базы данных при использовании не сокета.	3306
DBSchema	Имя схемы используется для PostgreSQL.	
DBSocket	Путь к сокету MySQL.	
DBTLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
DBTLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
DBTLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
DBTLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
DBTLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	
DBUser	Пользователь базы данных.	
DBVerifyHost	Проверяет, что имя хоста, указанное для клиента, совпадает с именем хоста, указанным в сертификате.	0
HistoryStorageDateIndex	Включает предварительное создание индексов для таблиц истории.	0
HistoryStorageTypes	Список типов истории, которые будут храниться в хранилищах истории.	
HistoryStorageURL	URL-адрес хранилища истории.	
VaultDBPath	Расположение, из которого будут извлекаться учетные данные базы данных, например, secret/zabbix/database. Ключ должен иметь имена свойств password и username.	

Debug hardware

Параметр	Описание	Значение по умолчанию
StartHistoryPollers	Количество предварительно запущенных процессов опроса истории.	5

Escalator

Параметр	Описание	Значение по умолчанию
StartEscalators	Количество предварительно запущенных процессов эскалации.	5

Ha cluster

Параметр	Описание	Значение по умолчанию
FailoverDelay	Время, которое должно пройти, прежде чем узел будет помечен как недоступный.	60
HANodeName	Имя узла кластера.	
NodeAddress	Адрес узла кластера.	localhost:10051

Housekeeper

Параметр	Описание	Значение по умолчанию
HousekeepingFrequency	Как часто Zabbix будет выполнять процедуру очистки.	1
MaxHousekeeperDelete	Максимальное количество строк, которое будет удалено за одну задачу очистки.	5000

Housekeeping of problems

Параметр	Описание	Значение по умолчанию
DisableHousekeeping	Отключает очистку данных.	0

Housekeeping of trends

Параметр	Описание	Значение по умолчанию
DisableHousekeeping	Отключает очистку данных.	0

Internal monitoring

Параметр	Описание	Значение по умолчанию
StartInternalPollers	Количество предварительно запущенных внутренних процессов опроса.	1

Log files

Параметр	Описание	Значение по умолчанию
DebugLevel	Уровень отладки.	3
LogFile	Имя файла журнала.	
LogFileSize	Максимальный размер файла журнала в МБ.	1
LogType	Тип журнала.	file

Network discovery

Параметр	Описание	Значение по умолчанию
StartDiscoverers	Количество предварительно запущенных процессов обнаружения.	1

Node watcher

Параметр	Описание	Значение по умолчанию
NodeWatcherFrequency	Как часто Zabbix будет проверять состояние узлов кластера.	5

Proxy data sender

Параметр	Описание	Значение по умолчанию
ProxyConfigFrequency	Как часто прокси получает данные конфигурации с сервера.	10
ProxyDataFrequency	Как часто прокси получает данные с сервера.	1

Proxy poller

Параметр	Описание	Значение по умолчанию
ProxyPollers	Количество предварительно запущенных процессов опроса прокси.	1
ProxyUnreachableDelay	Как часто прокси опрашивается после ошибки.	60

Proxy server

Параметр	Описание	Значение по умолчанию
ProxyHeartbeatFrequency	Как часто прокси отправляет heartbeat-сообщения на сервер.	60
ProxyLocalBuffer	Количество часов, в течение которых данные будут храниться локально на прокси.	0
ProxyMode	Режим работы прокси.	0
ProxyOfflineBuffer	Количество часов, в течение которых данные будут храниться на прокси, если связь с сервером потеряна.	1

Self-monitoring

Параметр	Описание	Значение по умолчанию
StartStatusPollers	Количество предварительно запущенных процессов опроса состояния.	1

Snmp trapper

Параметр	Описание	Значение по умолчанию
SNMPTrapperFile	Временный файл, используемый для передачи данных SNMP-трапов от процесса SNMP-трапперов.	/tmp/zabbix_traps.tmp
StartSNMPTrapper	Если установлено значение 1, будет запущен процесс SNMP-траппера.	0

Timeouts

Параметр	Описание	Значение по умолчанию
TrapperTimeout	Время ожидания обработки данных.	300
UnreachablePeriod	После скольких секунд недоступности узел сети будет считаться недоступным.	45

Trigger housekeeper

Параметр	Описание	Значение по умолчанию
DeleteOrphanedProblems	Удалять проблемы, у которых отсутствуют связанные триггеры.	0

Value cache

Параметр	Описание	Значение по умолчанию
ValueCacheSize	Размер кэша значений в байтах.	8M

Vmware monitoring

Параметр	Описание	Значение по умолчанию
StartVMwareCollectors	Количество предварительно запущенных процессов сборщиков VMware.	0
VMwareCacheSize	Размер кэша VMware в байтах.	8M
VMwareFrequency	Как часто Zabbix будет подключаться к сервису VMware.	60
VMwarePerfFrequency	Как часто Zabbix будет собирать данные производительности VMware.	60
VMwareTimeout	Время ожидания, используемое при выполнении SOAP-операций VMware.	10

Zabbix cluster

Параметр	Описание	Значение по умолчанию
ClusterAllowedIP	Список разрешенных IP-адресов, разделенных запятыми, для узлов кластера.	
ClusterDatabaseHost	Имя хоста или IP-адрес базы данных.	
ClusterDatabaseName	Имя базы данных.	
ClusterDatabasePassword	Пароль базы данных.	
ClusterDatabasePort	Порт базы данных при использовании не сокета.	3306
ClusterDatabaseSocket	Путь к сокету MySQL.	
ClusterDatabaseTLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
ClusterDatabaseTLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
ClusterDatabaseTLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
ClusterDatabaseTLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
ClusterDatabaseTLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	
ClusterDatabaseUser	Пользователь базы данных.	
ClusterDatabaseVerifyHost	Проверяет, что имя хоста, указанное для клиента, совпадает с именем хоста, указанным в сертификате.	0
ClusterId	Идентификатор узла кластера.	
ClusterLocalAddress	IP-адрес или DNS-имя для подключения к узлу кластера.	
ClusterLocalPort	Порт для подключения к узлу кластера.	10051
ClusterNodeTimeout	Время, которое должно пройти, прежде чем узел будет помечен как недоступный.	60
ClusterStandbyNodes	Список разрешенных IP-адресов, разделенных запятыми, для резервных узлов кластера.	
ClusterStartupTime	Время, которое должно пройти, прежде чем узел будет помечен как доступный.	0
StartClusterNodeWatchers	Количество предварительно запущенных процессов наблюдения за узлами кластера.	0

Zabbix server high availability

Параметр	Описание	Значение по умолчанию
HANodeName	Имя узла кластера.	
NodeAddress	Адрес узла кластера.	localhost:10051

Zabbix trapper

Параметр	Описание	Значение по умолчанию
StartTrappers	Количество предварительно запущенных процессов трапперов.	5

Zabbix web service

Параметр	Описание	Значение по умолчанию
StartReportWriters	Количество предварительно запущенных процессов генерации отчетов.	0
WebServiceURL	URL-адрес службы веб-отчетов.	

Forks (APS only)

Параметр	Описание	Значение по умолчанию
StartAlerters	Количество предварительно запущенных процессов оповещения.	3
StartConnectors	Количество предварительно запущенных процессов коннекторов.	1
StartDBSyncers	Количество предварительно запущенных процессов синхронизации базы данных.	4
StartDiscoverers	Количество предварительно запущенных процессов обнаружения.	1
StartEscalators	Количество предварительно запущенных процессов эскалации.	5
StartHTTPPollers	Количество предварительно запущенных процессов опроса HTTP.	1
StartHistoryPollers	Количество предварительно запущенных процессов опроса истории.	5
StartIPMIPollers	Количество предварительно запущенных процессов опроса IPMI.	0
StartJavaPollers	Количество предварительно запущенных процессов опроса Java.	0
StartLLDProcessors	Количество предварительно запущенных процессов обработки LLD.	2
StartODBCPollers	Количество предварительно запущенных процессов опроса ODBC.	1
StartPingers	Количество предварительно запущенных процессов пинга.	1
StartPollers	Количество предварительно запущенных процессов опроса.	5
StartPollersUnreachable	Количество предварительно запущенных процессов опроса недоступных узлов сети.	1
StartPreprocessors	Количество предварительно запущенных процессов предобработки.	3
StartSNMPPollers	Количество предварительно запущенных процессов опроса SNMP.	1
StartTimers	Количество предварительно запущенных процессов таймеров.	1
StartTrappers	Количество предварительно запущенных процессов трапперов.	5
StartVMwareCollectors	Количество предварительно запущенных процессов сборщиков VMware.	0

Forks (APS only)

Параметр	Описание	Значение по умолчанию
StartAlerters	Количество предварительно запущенных процессов оповещения.	3
StartConnectors	Количество предварительно запущенных процессов коннекторов.	1
StartDBSyncers	Количество предварительно запущенных процессов синхронизации базы данных.	4
StartDiscoverers	Количество предварительно запущенных процессов обнаружения.	1
StartEscalators	Количество предварительно запущенных процессов эскалации.	5
StartHTTPPollers	Количество предварительно запущенных процессов опроса HTTP.	1
StartHistoryPollers	Количество предварительно запущенных процессов опроса истории.	5
StartIPMIPollers	Количество предварительно запущенных процессов опроса IPMI.	0
StartJavaPollers	Количество предварительно запущенных процессов опроса Java.	0
StartLLDProcessors	Количество предварительно запущенных процессов обработки LLD.	2
StartODBCPollers	Количество предварительно запущенных процессов опроса ODBC.	1
StartPingers	Количество предварительно запущенных процессов пинга.	1
StartPollers	Количество предварительно запущенных процессов опроса.	5
StartPollersUnreachable	Количество предварительно запущенных процессов опроса недоступных узлов сети.	1
StartPreprocessors	Количество предварительно запущенных процессов предобработки.	3
StartSNMPPollers	Количество предварительно запущенных процессов опроса SNMP.	1
StartTimers	Количество предварительно запущенных процессов таймеров.	1
StartTrappers	Количество предварительно запущенных процессов трапперов.	5
StartVMwareCollectors	Количество предварительно запущенных процессов сборщиков VMware.	0

GnuTLS configuration

Параметр	Описание	Значение по умолчанию
GnuTLSPriorities	Строка приоритетов, используемая для настройки GnuTLS.	

Internal TLS configuration

Параметр	Описание	Значение по умолчанию
TLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
TLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
TLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
TLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
TLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	

OpenSSL configuration

Параметр	Описание	Значение по умолчанию
SSLCertLocation	Расположение файлов SSL-сертификатов.	
SSLKeyLocation	Расположение файлов SSL-ключей.	

Parameters for secure communication between Zabbix components

Параметр	Описание	Значение по умолчанию
TLSAccept	Тип соединений, которые должен принимать прокси.	unencrypted
TLSCAFile	Полный путь к файлу, содержащему центры сертификации (CA) для проверки сертификата сервера.	
TLSCertFile	Полный путь к файлу, содержащему сертификат клиента или цепочку сертификатов.	
TLSCipher	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.2 или более ранних версий.	
TLSCipher13	Переопределяет набор или наборы шифров, доступных для локальных соединений, установленных с помощью протокола TLSv1.3.	
TLSKeyFile	Полный путь к файлу, содержащему закрытый ключ клиента.	
TLSPSKFile	Полный путь к файлу, содержащему ключ PSK.	
TLSPSKIdentity	Идентификатор PSK.	

SSL configuration

Параметр	Описание	Значение по умолчанию
SSLCertLocation	Расположение файлов SSL-сертификатов.	
SSLKeyLocation	Расположение файлов SSL-ключей.	

TLS_CRL

Параметр	Описание	Значение по умолчанию
TLSCRLFile	Полный путь к файлу, содержащему отозванные сертификаты.	

ADPS Control Agents

Параметр	Описание	Значение по умолчанию
ADPSControlAgents	Список разрешенных IP-адресов, разделенных запятыми, для управляющих агентов ADPS.	

ADPS OpenVPN settings

Параметр	Описание	Значение по умолчанию
OpenVPNCipher	Алгоритм шифрования, используемый для OpenVPN.	AES-128-CBC, AES-192-CBC, AES-256-CBC
OpenVPNPort	Порт, используемый для OpenVPN.	1194

3 Агент

Файл конфигурации: /etc/zabbix/zabbix_agentd.conf

Параметры

Параметр	Описание	Значение по умолчанию
AllowRoot	Разрешить агенту запускать от имени root. Если запущен от root и эта опция отключена, агент попытается переключиться на пользователя, указанного в параметре User.	0

Active checks

Параметр	Описание	Значение по умолчанию
BufferSend	Как часто данные будут отправляться с агента на сервер.	5
BufferSize	Максимальное количество значений, которое будет храниться в буфере памяти.	100
EnableRemoteCommands	Разрешить выполнение удаленных команд.	0
HostInterface	Интерфейс узла сети.	
HostInterfaceItem	Элемент данных, используемый для получения интерфейса узла сети.	
HostMetadata	Метаданные узла сети.	
HostMetadataItem	Элемент данных, используемый для получения метаданных узла сети.	
RefreshActiveChecks	Как часто будет обновляться список активных проверок.	120
UnsafeUserParameters	Разрешить все символы в пользовательских параметрах.	0

Basic parameters

Параметр	Описание	Значение по умолчанию
Hostname	Имя узла сети.	
HostnameItem	Элемент данных, используемый для получения имени узла сети.	
PidFile	Имя файла PID.	/var/run/zabbix/zabbix_agentd.pid
SourceIP	Исходный IP-адрес.	

Debug hardware

Параметр	Описание	Значение по умолчанию
DebugLevel	Уровень отладки.	3
LogFile	Имя файла журнала.	
LogFileSize	Максимальный размер файла журнала в МБ.	1
LogType	Тип журнала.	file

Log files

Параметр	Описание	Значение по умолчанию
LogFile	Имя файла журнала.	
LogFileSize	Максимальный размер файла журнала в МБ.	1
LogType	Тип журнала.	file

Passive checks

Параметр	Описание	Значение по умолчанию
ListenIP	Список разрешенных IP-адресов, разделенных запятыми, для входящих соединений.	
ListenPort	Порт для входящих соединений.	10050
StartAgents	Количество предварительно запущенных процессов агента.	3

###

Поддерживаемые расширения

Евгения Кузина

Назад к содержанию

Расширения — это специальные модули, которые обеспечивают дополнительные функциональные возможности базы данных путем объединения нескольких объектов SQL в пакет для использования их как единого целого. Расширения, загруженные в базу данных, могут работать как встроенные функции. См. Работа с расширениями.

В таблице ниже перечислены расширения, доступные для установки.

Имя расширения	Версия по умолчанию	Описание
address_standardizer	3.4.1	Основанный на правилах (rules) стандартизатор, который используется для парсинга адреса на составляющие элементы
address_standardizer_data_us	3.4.1	Содержит данные для стандартизации адресов в США
adminpack	2.1	Административные функции для ADPG/PostgreSQL
amcheck	1.3	Функции для проверки логической целостности таблиц и индексов
autoinc	1.0	Функции для автоинкрементных полей
bloom	1.0	Индексный метод доступа, основанный на фильтрах Блума
btree_gin	1.3	Обеспечивает поддержку GIN-индексации основных типов данных
btree_gist	1.7	Обеспечивает поддержку GiST-индексации основных типов данных
citext	1.6	Тип данных для строк, нечувствительных к регистру – `citext`
cube	1.5	Тип данных `cube` для представления многомерных кубов
dblink	1.2	Выполняет запросы в удаленной базе данных
dict_int	1.0	Пример дополнительного шаблона словаря для полнотекстового поиска. Этот пример словаря управляет индексацией целых чисел, позволяя индексировать такие числа и предотвращая слишком большой рост количества уникальных слов
dict_xsyn	1.0	Пример шаблона словаря полнотекстового поиска для расширенной обработки синонимов. Этот тип словаря заменяет слова группами их синонимов и позволяет искать слово по любому из его синонимов
earthdistance	1.1	Рассчитывает расстояния между точками на поверхности Земли
file_fdw	1.0	Обертка внешних данных (foreign-data wrapper) для доступа к неструктурированным файлам
fuzzystrmatch	1.2	Определяет сходство и расстояние между строками
hstore	1.8	Тип данных для хранения пар ключ/значение в одном значении поля
insert_username	1.0	Функции для отслеживания пользователей, изменивших таблицу
intagg	1.1	Агрегатор и нумератор целых чисел (устаревший функционал)
intarray	1.5	Функции и операторы для работы с массивами целых чисел без `NULL`
isn	1.2	Типы данных для международных стандартов нумерации
lo	1.1	Поддерживает управление большими объектами (Large Objects, LO и Binary Large OBjects, BLOB)
ltree	1.2	Тип данных для иерархических древовидных структур
moddatetime	1.0	Функции для отслеживания времени последней модификации
old_snapshot	1.0	Позволяет проверить состояние сервера, которое используется для реализации old_snapshot_threshold
pageinspect	1.12	Позволяет исследовать содержимое страниц базы данных на низком уровне
pg_buffercache	1.4	Предоставляет информацию о состоянии буферного кеша
pg_freespacemap	1.2	Предоставляет средства для исследования карты свободного пространства (Free Space Map, FSM)
pg_orphaned	1.0	Обработка неиспользуемых файлов
pg_prewarm	1.2	Предварительная загрузка данных отношений в кеш буферов
pg_repack	1.5.2	Позволяет реорганизовывать таблицы
pg_stat_statements	1.10	Предоставляет средства для отслеживания статистики планирования и выполнения SQL-операторов
pg_surgery	1.0	Проведение операций низкого уровня с поврежденными отношениями
pg_trgm	1.6	Предоставляет функции и операторы для определения сходства алфавитно-цифровых строк на основе триграмм, а также классы операторов индексов, поддерживающие быстрый поиск схожих строк
pg_vector	0.8.0	Позволяет работать с векторными данными
pg_visibility	1.2	Предоставляет возможность исследовать для определенной таблицы карту видимости (Visibility Map, VM) и информацию о видимости на уровне страниц
pgaudit	16.1	Обеспечивает подробное ведение журнала аудита сессий и/или объектов с помощью стандартных средств логирования PostgreSQL
pgauditlogtofile	1.6	Дополнение к `pgaudit`, которое перенаправляет строки журнала аудита в независимый файл вместо использования стандартных средств логирования сервера PostgreSQL
pgcrypto	1.3	Криптографические функции
pg_walinspect	1.1	Предоставляет функции SQL, которые позволяют проверять содержимое WAL-лога работающего кластера на низком уровне
pgrowlocks	1.2	Предоставляет функцию, показывающую информацию о блокировке строк для заданной таблицы
pgstattuple	1.5	Предоставляет функции для получения статистики на уровне кортежей
plpgsql	1.0	Процедурный язык PL/pgSQL. См. Использование PL/pgSQL
plperl	1.0	Процедурный язык PL/Perl
plperlu	1.0	Недоверенный процедурный язык PL/PerlU
plpython3u	1.0	Недоверенный процедурный язык PL/Python3U. См. Использование PL/Python
pltcl	1.0	Процедурный язык PL/Tcl. См. Использование PL/Tcl
pltclu	1.0	Недоверенный процедурный язык PL/TclU. См. Использование PL/Tcl
postgis	3.4.1	Геометрические и географические пространственные типы и функции PostGIS
postgis_raster	3.4.1	Растровые типы и функции PostGIS
postgis_sfcgal	3.4.1	SFCGAL-функции PostGIS
postgis_tiger_geocoder	3.4.1	Tiger-геокодер и обратный геокодер PostGIS
postgis_topology	3.4.1	Топологические пространственные типы и функции PostGIS
postgres_fdw	1.1	Обертка внешних данных (foreign-data wrapper) для доступа к данным, хранящимся на внешних серверах PostgreSQL
refint	1.0	Функции для реализации ссылочной целостности (устаревший функционал)
seg	1.4	Тип данных `seg` для представления отрезков или интервалов чисел с плавающей точкой
sslinfo	1.2	Предоставляет информацию о сертификатах SSL
tablefunc	1.0	Функции, возвращающие таблицы, включая `crosstab`
tcn	1.0	Предоставляет триггерную функцию, уведомляющую прослушивателей событий (listeners) об изменениях в любой таблице, к которой она прикреплена
timescaledb	2.15.0	Обеспечивает масштабируемые вставки и сложные запросы для данных временных рядов. См. Использование timescaledb
timescaledb_toolkit	1.19.0	Предоставляет дополнительные гиперфункции
tsm_system_rows	1.0	Предоставляет метод извлечения выборки `SYSTEM_ROWS`, который можно использовать в выражении `TABLESAMPLE` команды `SELECT`
tsm_system_time	1.0	Предоставляет метод извлечения выборки `SYSTEM_TIME`, который можно использовать в выражении `TABLESAMPLE` команды `SELECT`
unaccent	1.1	Словарь текстового поиска, который убирает надстрочные (диакритические) знаки из лексем
uuid-ossp	1.1	Предоставляет функции для генерирования универсальных уникальных идентификаторов (UUID), используя один из нескольких стандартных алгоритмов
xml2	1.1	Предоставляет функции для выполнения запросов XPath и преобразований XSLT

Релизы ADPG

Содержание

- 14.3
- 14.2
- 14.1

Назад к содержанию

14.3

14.3.1

Дата: 07.11.2025

Новые функции

Добавлена поддержка Astra Linux 1.7 "Воронеж"

Добавлен новый скрипт ADPG Control Agent с двумя компонентами: ADPG Control Agent, ADPG Control ADMN Agent

Добавлена команда `upgrade`

Добавлена команда `upgradeStandalone`

Улучшения

Добавлен параметр `--between_groups_sessions` в команде `failover`

Исправленные баги

Issues 1, 2, исправлена ошибка инсталлера ADCM на работу для кластера PostgreSQL

Пользовательский тип кластера в продукте "Шлюз для сбора данных" **AccessRouterAdmin**

Обой при установке сайзинга ADPG, если отсутствовала опция `WC_DB_OPT_S`

Проблемы с переключением при промежуточном обновлении 13.1 через промежуточную версию 14.1 (**patroni**)

14.2.4

Дата: 24.07.2025

Новые функции

Добавлена поддержка ОС Альт Сервер 10 (**Server 10**) и Альт 8 СП (**Alt 8**)

Добавлена поддержка PostgreSQL 17 (**pg_probackup**)

Добавлена поддержка Patroni 4.0.5 API

Добавлена поддержка Python 3.12 (**Astra Linux 1.8**)

Добавлена поддержка Python 3.11 (**Astra Linux 1.7**)

Добавлена поддержка Python 3.10 (**Astra Linux 1.7**)

Добавлена поддержка Python 3.9 (**Astra Linux 1.7**)

Добавлена поддержка Python 3.8 (**CentOS 7**)

Добавлена поддержка HTTPS для Enterprise Tools

Расширение Timescale DB обновлено до версии 2.15.3

Улучшения

Важное: был удален из сборочных конструктов **import**

Добавлена проверка на наличие нужных компонентов на кластерах

Добавлена возможность создания кластера PostgreSQL 14 с поддержкой расширения Timescale DB

Добавлена возможность создания кластера PostgreSQL 14 с поддержкой расширения Timescale DB

Добавлена проверка доступности репозиториев, что сервер ADPG должен обязательно находиться на том же узле

Arcadia обновлена до версии 2.5

Добавлена проверка в установочном скрипте на конфликты, где также проверяется наличие пакетов **/opt/postgrespro** выбранной на установку версии PostgreSQL

Исправленные баги

Некоторые строки конфигурации со специальными символами могли быть закодированы обратным слэшем по отображению безопасности

Исправлена ошибка при неправильном вводе ключа в командах на shell 2.0

Падение продуктов PostgreSQL и Шлюз для сбора данных на ADPG 2.0

Добавлен параметр на перезагрузку после установки **Pgx** сервиса Monitoring и сервиса

Проверка общих для основных Postgres пакетов на наличие в версиях репозиториев

Не удалялись первые репозитории Debian на новых хост

14.2.3

Дата: 13.03.2025

Новые функции

Добавлена поддержка Python 3.9 и Python 3.10 для Astra Linux

Добавлена поддержка Python 3.8 для **postgresql.conf**

Переменная **pl/pgsql_check_pragmas_float** была перенесена для отключения этого временных проблем

Переменная **SSL_SUPPORTED** была добавлена для проверки, есть ли SSL, перед перезапуск, что ADCD не сможет запустит, сервер перезапускается и данные в **.info**

Добавлена переменная **SSLCERT** для указания пути к сертификату

Имена сетевых интерфейсов ADPG были изменены на имена сетевых Patroni

Исправленные баги

Переменная **TUNING_addresses** неправильно обрабатывается в случае использования нескольких IP адресов

Действие перед **Check** завершалось ошибкой, поскольку в файле patroni.log появлялись IP адреса и не FQDN

Действие **Abort upgrade** завершалось ошибкой, если не существовала директория **/upgrade**, которая была удалена

Действие **Complete upgrade** завершалось ошибкой, если кластер PostgreSQL был уже запущен перед обновлением

Использовался параметр Patroni, когда было достаточно параметра **Patroni Arguments** в Patroni

Пакеты, необходимые для сборки Patroni, не обновлялись на версии для совместимости

Действие **Update configuration** не работало, если был установлен **pg_stat_statements**

Значение после **PG_PORT** изменялось, но значение по умолчанию оставалось **Patroni Arguments** в Patroni

Для действия **Reinstall** завершалось ошибкой, если была указана **Citus** Patroni

Действие **Edit configuration** не работало, если был установлен **pg_stat_statements**

Пользователям с правами sudo нужно было вводить пароль для каждого действия

14.2.2

Дата: 12.01.2025

Новые функции

Добавлена поддержка Astra Linux 1.8 версии **orel** (**Astra Linux 1.8**)

Известные проблемы ADPG

Евгения Кузина

Назад к содержанию

Проблема	С версии	Fixed	Обходное решение
После истечения срока действия сертификатов SSL действия кластера ***Stop*** и ***Start*** завершаются ошибкой	16.3.4	—	Обновите сертификаты и сохраните их по тем же путям. Выполните действие кластера ***Stop***, а затем действие кластера ***Start***
Действие ***Expand*** сервиса ADPG завершается ошибкой, если SSL включен с помощью действия кластера ***Manage SSL***	16.3.4	—	Выполните следующие шаги: 1. Выполните действие ***Manage SSL***, чтобы отключить переключатель ***Enable SSL***. 2. Добавьте сервис ADPG на новый хост через действие ***Expand***. 3. Поместите ключи и сертификаты SSL на новый хост ADPG в соответствии с путями, указанными в окне действия ***Manage SSL***. 4. Выполните действие ***Manage SSL***, чтобы активировать переключатель ***Enable SSL***.
Действие ***Add/remove components*** сервиса Monitoring, которое выполняется для добавления компонента Grafana на новый хост, завершается ошибкой с сообщением `No package matching 'admprom-grafana-9.5.8' found available, installed or updated`	16.3.3	—	■ ADPG EE: сначала разверните сервисы ADPG, Etcd и Chrony на новом хосте с помощью действия ***Expand***, затем запустите действие ***Add/remove components***, чтобы добавить Grafana. ■ ADPG CE: сначала разверните сервис Chrony на новом хосте с помощью действия ***Expand***, затем запустите действие ***Add/remove components***, чтобы добавить Grafana.
После действия ***Abort upgrade*** в случае отмены мажорного обновления 14 → 16 сервис Monitoring отмечается красным флажком, отображается ошибка `Service has an issue with config`, и кнопка ***Save*** выделяется красным в конфигурационной секции сервиса Metrics storage	16.3.3	—	Пересохраните конфигурацию без изменения значений
Действия ***Check*** кластера и сервиса Monitoring не завершаются ошибкой, если какая-либо подзадача мониторинга заканчивается сбоем	16.3.3	—	На странице ***Jobs*** разверните все подзадачи сервиса Monitoring, чтобы выявить потенциальные проблемы
После мажорного обновления с версии ADPG 14 на ADPG 16 ***statuschecker*** кластера отмечен желтым цветом несмотря на то, что все компоненты кластера, хосты и сервисы отмечены зеленым. Переустановка ***statuschecker*** не помогает	16.3.3	—	Перезапустите контейнер ADCM (и контейнер Postgres, если используется внешняя база данных)
При подключении пользователя pgbouncer к `psql` появляется предупреждение `Some psql features might not work` из-за версии pgbouncer 1.23.0	16.3.1	—	Можно игнорировать это предупреждение
Удаление (с использованием 🗑) сервиса Monitoring, ошибочно распределенного на хост с неподдерживаемой ОС, завершается ошибкой	14.3.6	—	Перераспределите компоненты мониторинга на другой хост с поддерживаемой ОС (CE: CentOS, Ubuntu; EE: CentOS, Alt Linux, Astra Linux, Ubuntu, RED OS). После этого можно успешно удалить сервис Monitoring из кластера
Установка кластера ADPG завершается ошибкой, если одно из имен хостов кластера совпадает с именем, зарезервированным для сервисов: ■ `adpg` ■ `balancer` ■ `chrony` ■ `etcd` ■ `prometheus_monitoring`	14.3.6	—	Пересоздайте кластер с именем хоста, которое не совпадает с именем сервиса. Например, в случае `adpg` можно использовать `adpg1`, `adpg2` или любое другое незарезервированное слово
Порт, который слушает PgBouncer, не может быть изменен для работающего кластера (параметр PgBouncer `listen_port` сервиса ADPG)	14.3.6	—	После изменения порта выполните действие Reconfigure & Restart сервиса ADPG
Если добавляются ноды для расширения сервиса Etcd, новые ноды не включаются в конфигурацию кластера ADPG	14.3.3	—	Для решения проблемы выполните действие ***Reconfigure & Restart*** сервисов ADPG и Balancer
Кластерное действие Upgrade, используемое для обновления версии ADPG 14.3.8 до ADPG 16, нестабильно работает при перемещении реплик. С помощью этого действия можно выполнить мажорное обновление кластеров ADPG с сервисом ADPG, установленным только на одном хосте (без реплик)	16.3.1	16.3.2	Используйте логическое резервное копирование/восстановление с помощью утилит pg_dump и `pg_restore` для перемещения данных из ADPG 14 в ADPG 16
Если используется порт сервиса ADPG, отличный от порта по умолчанию (5432), при настройке резервного хранилища (backup repo) не удается создать станзу	14.3.7	16.3.1	Переключитесь на порт сервиса ADPG по умолчанию — 5432
Во время установки ADPG в операционной системе Astra Linux "Орел" 1.7.x может возникнуть следующая ошибка: `invalid locale name "en_US.UTF-8"`	14.3.6	16.3.4	Добавьте локаль `en_US.UTF-8`
С того момента, как была реализована новая встроенная система мониторинга в версии 14.3.4, ADPG больше не поддерживает интеграцию с импортируемым кластером мониторинга, и поэтому любое минорное обновление 14.x → 14.y сбрасывает настройку ***Enable monitoring*** на значение `false`	14.2.1	16.3.1	Вам необходимо настроить систему мониторинга и установить ***Enable monitoring*** в значение `true` вручную
Обновление кластера ADPG с предыдущей версии до версии 14.3.2 не сохраняет настройки, указанные в поле ***postgresql.conf customsection*** конфигурации сервиса ADPG	14.2.1	14.3.2	Если необходимо обновить кластер ADPG с версий 14.2.1 - 14.3.1, требуется восстановить содержимое ***postgresql.conf custom section*** вручную