СУБД FTData

АО «Современные Системы»

Версия 17.5.3, 2025-09-04

Содержание

Сервер СУБД FTData . . . 1
- Версии СУБД FTData . . . 1
- Пакеты СУБД FTData . . . 1
- Установка сервера СУБД FTData . . . 3
- Удаление сервера СУБД FTData . . . 5
- Обновление сервера СУБД FTData . . . 6

Обновление БД на требуемую версию СУБД FTData . . . 6
- Обновление с БД PostgreSQL . . . 6
- Обновление БД к актуальной минорной версии СУБД FTData . . . 7
- Миграция кластера БД FTData 16 на СУБД FTData 17 . . . 8

Сбор диагностической информации СУБД FTData . . . 13
- Настройка сбора дампов памяти процессов . . . 13
- Сбор данных о сбое СУБД FTData . . . 17
- Сбор диагностической информации о критических падениях . . . 18

Управление сервером СУБД FTData . . . 19
- Работа с БД . . . 20
- Проверка версий ПО . . . 21

Конфигурирование сервера СУБД FTData . . . 22

Работа с лицензией СУБД FTData . . . 25

Дополнительные возможности мониторинга в СУБД FTData . . . 27
- Мониторинг активности процессов сервера . . . 27
- Мониторинг использования кэша субтранзакций . . . 30
- Логирование неудачных блокировок отношений . . . 32
- Статистика планирования . . . 34
- Выдача планов и идентификаторов запросов . . . 39

Хинты . . . 40
- Общие сведения . . . 40
- Cardinality . . . 41
- Index . . . 42
- Materialize . . . 43
- Parallel . . . 45

Интервальное секционирование . . . 48
- Поддержка интервального секционирования . . . 48
- Расширение синтаксиса операторов CREATE TABLE/ALTER TABLE для поддержки интервального секционирования . . . 49
- Расширение таблиц системного каталога . . . 51

Выключение механизма миграции строк между разделами секционированной таблицы . . . 52

Оптимизация запросов с оператором ИЛИ . . . 55

Общая информация . . . 55
Ограничения . . . 55
Конфигурация . . . 56
Пример работы оптимизации . . . 56
Оптимизация кэша субтранзакций . . . 58
Кэш субтранзакций в PGPROC . . . 58
Статусы субтранзакций в pg_subtrans . . . 58
Сжатие данных таблиц и индексов . . . 61
Управление сжатием таблиц и индексов . . . 61
Поиск свободного места для записи сжатой страницы данных . . . 63
Восстановление после сбоя . . . 64
Утилита ftdbackrest . . . 66
Общие сведения . . . 66
Установка . . . 68
Начало работы . . . 69
Резервное копирование . . . 77
Хранение . . . 78
Восстановление . . . 80
Множественные репозитории . . . 84
Удаление станзы . . . 84
Резервное копирование на выделенный хост репозитория . . . 85
Параллельное резервное копирование и восстановление . . . 88
Запуск и остановка . . . 88
Репликация . . . 89
Асинхронное архивирование . . . 93
Резервное копирование из резервного сервера . . . 95
Расширение ftd_profile . . . 96
Общие сведения . . . 96
Быстрый старт . . . 96
Пример скриптов для автоматизации установки `ftd_profile` . . . 98
Требования . . . 99
Установка и настройка расширения ftd_profile . . . 101
Использование расширения ftd_profile . . . 104
Описание метрик ОС . . . 110
Статистика по внутренним метрикам экземпляра FTData . . . 111
Расширение ftd_trace . . . 113
Общие сведения . . . 113
Расширение oracle_fdw . . . 116
Общие сведения . . . 116
Установка расширения . . . 116
Удаление расширения . . . 117

Обновление расширения . . . 117
Пример использования . . . 118
Описание объектов и функций расширения . . . 120
Опции . . . 121
Использование . . . 125
Внутреннее устройство . . . 131
Проблемы . . . 131
Расширение pbl . . . 134
Общие сведения . . . 134
Повторное использование планов . . . 134
Неподдерживаемые типы запросов . . . 135
Ручная фиксация планов . . . 135
Хранение планов и запросов в кэше . . . 136
Идентификаторы запроса . . . 137
Идентификатор структуры плана . . . 138
Отслеживание зависимостей планов . . . 139
Обмен статистикой запросов между сессиями . . . 140
Сохранение содержания кэша при перезапуске сервера . . . 140
Управление памятью глобального кэша . . . 140
Взаимодействие pbl с кэшем подготовленных выражений . . . 141
Параметры конфигурации сервера . . . 143
Установка расширения . . . 145
Удаление расширения . . . 145
SQL-интерфейс расширения . . . 145
Ограничение количества используемых процессорных ядер . . . 156
Изменения в выпусках . . . 157

Сервер СУБД FTData

СУБД выпускается в виде набора пакетов для ОС:

- Astra Linux 1.7;
- Astra Linux 1.8;
- Debian 11;
- Альт СП Сервер релиз 10;
- Oracle Linux 8.6;
- РЕД ОС 8.0;
- РЕД ОС 7.3.

Версии СУБД FTData

Полная версия СУБД FTData имеет следующий формат:

`[M.I].[A.B-C]`

Где:

- `[M]` - мажорная версия базового PostgreSQL (например 17);
- `[M.I]` - полная версия базового PostgreSQL (например 17.5):
  - `M` - мажорная версия базового PostgreSQL;
  - `I` - минорная версия базового PostgreSQL.
- `[A.B-C]` - версия FTData (например `1.1-1`):
  - `A` - мажорная версия FTData;
  - `B` - минорная версия FTData (необязательный параметр);
  - `C` - номер сборки FTData.

Например: `17.5.3` - версия СУБД FTData основанная на PostgreSQL 17.5 (мажорная версия PostgreSQL 17).

Пакеты СУБД FTData

Схема именования пакетов СУБД FTData

Схема именования пакетов СУБД FTData:

- rpm: `ftdata-[M]-[NAME]-[M.I].[A.B-C].[OS].x86_64.rpm`;
- deb: `ftdata-[M]-[NAME]_[M.I].[A.B-C].[OS]_amd64.deb`.

Схема именования пакетов расширений для СУБД FTData аналогична:

- rpm: `ftdata-[M]-[NAME]-[M.I].[A.B-C].[OS].x86_64.rpm`;
- deb: `ftdata-[M]-[NAME]_[M.I].[A.B-C].[OS]_amd64.deb`.

Обозначения в схемах именования пакетов:

- `[M]` - мажорная версия базового PostgreSQL (например `17`);
- `[M.I]` - полная версия базового PostgreSQL (например `17.5`) см. Версии СУБД FTData;
- `[A.B-C]` - версия FTData (например `1.1-1`):
  - `A` - мажорная версия FTData;
  - `B` - минорная версия FTData (необязательный параметр);
  - `C` - номер сборки FTData. Версии СУБД FTData;
- `[NAME]` - имя компонента (см. Компоненты СУБД FTData, поставляемые в виде пакетов), для сервера СУБД отсутствует;
- `[OS]` - код дистрибутива:
  - `ast1.7` - Astra Linux 1.7;
  - `ast1.8` - Astra Linux 1.8;
  - `deb` - Debian 11;
  - `alt10` - Альт СП Сервер релиз 10;
  - `el8` - Oracle Linux 8;
  - `red80` - РЕД ОС 8.0;
  - `el7` - РЕД ОС 7.3;

Пример: наименование клиента и сервера СУБД, пакета расширения oracle_fdw:

- rpm:
  - Сервер (без [NAME]): `ftdata-17-17.5.3-1.red80.x86_64.rpm`;
  - Клиент: `ftdata-17-client-17.5.3-1.red80.x86_64.rpm`;
  - Расширение: `ftdata-17-oracle-fdw-17.5.3-1.red80.x86_64.rpm`.
- deb:
  - Сервер (без [NAME]): `ftdata-17_17.5.3-1.ast1.7_amd64.deb`;
  - Клиент: `ftdata-17-client_17.5.3-1.ast1.7_amd64.deb`;
  - Расширение: `ftdata-17-oracle-fdw_17.5.3-1.ast1.7_amd64.deb`.

Компоненты СУБД FTData, поставляемые в виде пакетов

Имя компонента `[NAME]`	Назначение
Нет.	Набор серверных программ, расширения СУБД FTData.
`client`	Набор клиентских программ СУБД FTData.
`ftdbackrest`	Утилита управления резервным копированием и восстановлением кластеров баз данных СУБД FTData.
`libpq`	Библиотека C API для работы с сервером СУБД FTData.
`ftd-profile`	Расширение `ftd`_profile.
`llvmjit`	Пакет поддержки JIT-компиляции.
`plpython3`	Пакет загружаемого процедурного языка PL/Python (позволяет реализовывать функции и процедуры PostgreSQL на языке программирования Python 3).
`plperl`	Пакет загружаемого процедурного языка PL/Perl (позволяет реализовывать функции и процедуры PostgreSQL на языке программирования Perl).
`pltcl`	Пакет загружаемого процедурного языка PL/Tcl (позволяет реализовывать функции и процедуры PostgreSQL на языке программирования Tcl).
`dev`	Пакет для разработчиков библиотеки libecpg, libpgtypes, libecpg-compat и заголовочные файлы.
`oracle-fdw`	Расширение oracle_fdw.

Установка сервера СУБД FTData

В процессе установки СУБД FTData может потребоваться установка дополнительных пакетов, относящихся к ОС, поэтому прежде необходимо убедиться в доступности системных репозиториев и возможности установки из них. Перед установкой deb-пакетов рекомендуется выполнить команду: `sudo apt update`.

Пакеты СУБД FTData на основе PostgreSQL 16 и PostgreSQL 17 не конфликтуют между собой и любыми версиями PostgreSQL. То есть указанные СУБД могут быть установлены и запущены на одном сервере.

После установки СУБД FTData следует настроить сбор диагностической информации (см. раздел Сбор диагностической информации СУБД FTData).

Установка на дистрибутивах Debian и Astra Linux

Установка сервера FTData (минимальный набор пакетов):

```
sudo apt install -y ./ftdata-17-libpq*.deb
sudo apt install -y ./ftdata-17-client*.deb
sudo apt install -y ./ftdata-17_17.5*.deb
```

Установка дополнительных пакетов FTData:

> Если пакет `llvmjit` установлен, то JIT компиляция будет включена по умолчанию (это может привести к замедлению работы СУБД). Рекомендуется устанавливать пакет только по необходимости.

```
sudo apt install -y ./ftdata-17-ftdbackrest*.deb
sudo apt install -y ./ftdata-17-llvmjit*.deb
sudo apt install -y ./ftdata-17-plperl*.deb
sudo apt install -y ./ftdata-17-plpython3*.deb
sudo apt install -y ./ftdata-17-pltcl*.deb
sudo apt install -y ./ftdata-17-dev*.deb
```

Установка на дистрибутивах Oracle Linux и РЕД ОС

Установка сервера FTData (минимальный набор пакетов):

```
sudo yum install -y ./ftdata-17-libpq*.rpm
sudo yum install -y ./ftdata-17-client*.rpm
sudo yum install -y ./ftdata-17-17.5*.rpm
```

Установка дополнительных пакетов FTData:

> Если пакет `llvmjit` установлен, то JIT компиляция будет включена по умолчанию (это может привести к замедлению работы СУБД). Рекомендуется устанавливать пакет только по необходимости.

```
sudo yum install -y ./ftdata-17-ftdbackrest*.rpm
sudo yum install -y ./ftdata-17-llvmjit*.rpm
sudo yum install -y ./ftdata-17-plperl*.rpm
sudo yum install -y ./ftdata-17-plpython3*.rpm
sudo yum install -y ./ftdata-17-pltcl*.rpm
sudo yum install -y ./ftdata-17-dev*.rpm
```

Установка на дистрибутивах Альт

Установка сервера FTData (минимальный набор пакетов):

```
sudo apt-get install -y ./ftdata-17-libpq*.rpm
sudo apt-get install -y ./ftdata-17-client*.rpm
sudo apt-get install -y ./ftdata-17-17.5*.rpm
```

Установка дополнительных пакетов FTData:

Если пакет `llvmjit` установлен, то JIT компиляция будет включена по умолчанию (это может привести к замедлению работы СУБД). Рекомендуется устанавливать пакет только по необходимости.

```
sudo apt-get install -y ./ftdata-17-ftdbackrest*.rpm
sudo apt-get install -y ./ftdata-17-llvmjit*.rpm
sudo apt-get install -y ./ftdata-17-plperl*.rpm
sudo apt-get install -y ./ftdata-17-plpython3*.rpm
sudo apt-get install -y ./ftdata-17-pltcl*.rpm
sudo apt-get install -y ./ftdata-17-dev*.rpm
```

Удаление сервера СУБД FTData

При удалении сервера СУБД FTData не удаляются данные БД и настройки пользователя postgres.

Все пакеты сервера СУБД FTData зависят от пакета `ftdata-17-libpq`. При удалении `ftdata-17-libpq` автоматически удалятся все установленные пакеты СУБД FTData.

Удаление сервера FTData на дистрибутивах Debian и Astra Linux:

```
sudo apt remove -y ftdata-17-libpq
```

Удаление сервера FTData на дистрибутивах Oracle Linux и РЕД ОС:

```
sudo yum remove -y ftdata-17-libpq
```

Удаление сервера FTData на дистрибутивах Альт:

```
sudo apt-get remove -y ftdata-17-libpq
```

Обновление сервера СУБД FTData

Для обновления сервера СУБД FTData следует удалить существующие пакеты и установить новую версию. Если для построения индексов и представлений в БД использовались функции и операторы расширения dbi_funcs, следует выполнить обновление БД по инструкции для обновления расширения dbi_funcs.

Также следует выполнить процедуру преобразования БД, как описано в разделе Обновление БД на требуемую версию СУБД FTData.

Обновление БД на требуемую версию СУБД FTData

Для полноценного использования всех возможностей, предоставляемых СУБД FTData, требуется выполнить процедуры обновления и миграции БД, в ходе которой будут выполнены обновление метаданных и перенос данных БД. Ниже приводится таблица, поясняющая процесс обновления БД на СУБД FTData 17.5.3. Для понимания терминологии и нумерации версий см. раздел Версии СУБД FTData.

Версия СУБД	Изменение	Выполняемые действия
PostgreSQL 17	Переход с ванильной версии на FTData в рамках единой мажорной ванильной версии	Выполнить скрипт обновления `db_convert.sql` (см. Обновление с БД PostgreSQL).
PostgreSQL 16	Переход с ванильной версии на FTData с поднятием мажорной ванильной версии	Выполнить скрипт обновления `db_convert.sql` на FTData 16.9 (см. Обновление с БД PostgreSQL), а затем выполнить миграцию БД (см. Миграция кластера БД FTData 16 на СУБД FTData 17) на FTData 17.5.3.
FTData 17	Изменение минорной или мажорной версии FTData	Выполнить скрипт обновления `db_convert.sql` на FTData 17.5 (см. Обновление БД к актуальной минорной версии СУБД FTData).
FTData 16	Изменение ванильной мажорной версии FTData	Выполнить скрипт обновления `db_convert.sql` на FTData 16.9 (см. Обновление БД к актуальной минорной версии СУБД FTData), а затем выполнить миграцию БД (см. Миграция кластера БД FTData 16 на СУБД FTData 17) на FTData 17.5.3.

Обновление с БД PostgreSQL

В данном разделе описывается процедура обновления метаданных БД, созданной СУБД PostgreSQL, для добавления расширенных функций СУБД FTData. Преобразование может

быть выполнено только для одноимённых мажорных версий (от PostgreSQL 16 к FTData 16 или от PostgreSQL 17 к FTData 17). После преобразования файлы базы данных могут использоваться с FTData.

Запускается экземпляр FTData с указанием пути к кластеру:

```
/opt/ftdata/M/bin/pg_ctl start -D {путь к кластеру БД, созданному PostgreSQL версии M}
```

где `M` - мажорная версия PostgreSQL и FTData (`16` или `17`).

Для БД выполнится скрипт `db_convert.sql`, входящий в серверный пакет FTData. Скрипт выполняется для БД `postgres` и каждой из пользовательских преобразуемых баз данных один раз.

```
/opt/ftdata/M/bin/psql -d postgres -f /opt/ftdata/M/share/postgresql/db_convert.sql
/opt/ftdata/M/bin/psql -d {имя БД, созданной PostgreSQL версии M} -f /opt/ftdata/M/share/postgresql/db_convert.sql
...
/opt/ftdata/M/bin/psql -d {имя БД, созданной PostgreSQL версии M} -f /opt/ftdata/M/share/postgresql/db_convert.sql
```

где `M` - мажорная версия PostgreSQL и FTData (`16` или `17`).

В ходе выполнения скрипта будет напечатан список внесённых изменений.

```
`pg_catalog` transformation summary:

function pg_stat_get_activity():   updated
view     pg_stat_activity:         updated
function ftd_stat_get_subxids():   created
function ftd_stat_clean_subxids(): created
view     ftd_stat_subxids:         created
column   partintervalexpr:         added to pg_partitioned_table
```

Если при внесении изменений в `pg_catalog` возникнут конфликты, ни одно из изменений не будет применено (все манипуляции выполняются в одной транзакции).
Скрипт `db_convert.sql` вносит только необходимые изменения, - повторное выполнение скрипта безопасно.

Обновление БД к актуальной минорной версии СУБД FTData

В данном разделе описывается процедура обновления метаданных БД, созданной СУБД FTData более ранней версии, для добавления расширенных функций СУБД FTData. Преобразование может быть выполнено только в рамках одной мажорной версии FTData.

Для БД выполняется скрипт `db_convert.sql`, входящий в серверный пакет FTData. Скрипт

выполняется для БД postgres и каждой из пользовательских преобразуемых баз данных один раз.

```
/opt/ftdata/M/bin/psql -d postgres -f /opt/ftdata/M/share/postgresql/db_convert.sql
/opt/ftdata/M/bin/psql -d {имя БД, созданной предыдущей версией FTData M} -f
/opt/ftdata/M/share/postgresql/db_convert.sql
...
/opt/ftdata/M/bin/psql -d {имя БД, созданной предыдущей версией FTData M} -f
/opt/ftdata/M/share/postgresql/db_convert.sql
```

где M - мажорная версия FTData (16 или 17).

В ходе выполнения скрипта будет напечатан список внесённых изменений.

```
`pg_catalog` transformation summary:

function pg_stat_get_activity():   no update required
view     pg_stat_activity:         no update required
function ftd_stat_get_subxids():   already exists
function ftd_stat_clean_subxids(): already exists
view     ftd_stat_subxids:         already exists
column   partintervalexpr:         added to pg_partitioned_table
```

Если при внесении изменений в pg_catalog возникнут конфликты, ни одно из изменений не будет применено (все манипуляции выполняются в одной транзакции).
Скрипт db_convert.sql вносит только необходимые изменения, - повторное выполнение скрипта безопасно.

Миграция кластера БД FTData 16 на СУБД FTData 17

Описание определяет выполнение последовательности действий для обновления отдельного кластера базы данных. Процесс обновления комплекса, состоящего из набора реплик, необходимо определить самостоятельно.

Возможны следующие варианты миграции:

- Миграция кластера БД средствами pg_upgrade;
- Миграция кластера БД средствами pg_dumpall;
- Миграция кластера базы данных с применением логической репликации.

Перед началом работы следует установить СУБД FTData 17.5.3 (СУБД FTData 16 удалять не требуется).

Миграция кластера БД средствами pg_upgrade

Документация на утилиту pg_upgrade доступна в разделе pg_upgrade

> документации PostgreSQL 17.

Утилита `pg_upgrade` запускает серверы с портом 50432, чтобы не допустить нежелательных клиентских подключений. В процессе обновления оба кластера могут использовать один номер порта, так как они не будут работать одновременно. Однако для проверки старого работающего сервера новый порт должен отличаться от старого.

При использовании опции `--clone` миграция выполняется так же быстро и экономит место на диске, как и с опцией `--link`, но позволяет сохранить рабочее состояние старого кластера при запуске нового. Для этого варианта тоже требуется, чтобы старый и новый каталоги данных находились в одной файловой системе. Клонирование возможно только в некоторых операционных системах с определёнными файловыми системами.

При использовании опции `--link` миграция выполнится гораздо быстрее (так как файлы не копируются) и потребует меньше места на диске, но будет потеряна возможность обращаться к старому кластеру после запуска нового. Этот вариант также требует, чтобы каталоги данных старого и нового кластера располагались в одной файловой системе. Табличные пространства и `pg_wal` могут находиться в других файловых системах.

> Использование опции `--link` усложняет восстановление кластера (см. раздел Восстановление кластера БД FTData 16).

Создать новый кластер БД FTData 17

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl init -D /var/ftdata/17/data
```

Сделать резервную копию данных БД FTData 16

Копию можно сделать утилитой `ftdbackrest` (см. раздел Утилита ftdbackrest).

Копию можно создать утилитой `pg_dumpall`:

```
sudo -u postgres mkdir /var/ftdata/16/bkp
sudo -u postgres /opt/ftdata/16/bin/pg_dumpall > /var/ftdata/16/bkp/dumpall
sudo -u postgres cp /var/ftdata/16/data/*.conf /var/ftdata/16/bkp
sudo -u postgres tar -czf /var/ftdata/16/bkp.tar.gz -C /var/ftdata/16/bkp /var/ftdata/16/bkp
```

Архив `/var/ftdata/16/bkp.tar.gz` будет содержать все данные, необходимые для восстановления.

Остановить сервер FTData 16

```
sudo -u postgres /opt/ftdata/16/bin/pg_ctl -D /var/ftdata/16/data stop
```

Выполнить проверку возможности миграции

При запуске `pg_upgrade` с опцией `--check` будет проверена совместимость, и появится сообщение о том, какие коррективы вам нужно будет внести вручную после обновления. Проверка должна выполняться с теми же флагами, что и миграция (добавляется только флаг `--check`).

```
sudo -u postgres /opt/ftdata/17/bin/pg_upgrade \
  -b /opt/ftdata/16/bin -B /opt/ftdata/17/bin \
  -d /var/ftdata/16/data  -D /var/ftdata/17/data \
  --jobs $(nproc) \
  --check
```

Выполнить миграцию

```
sudo -u postgres /opt/ftdata/17/bin/pg_upgrade \
  -b /opt/ftdata/16/bin -B /opt/ftdata/17/bin \
  -d /var/ftdata/16/data  -D /var/ftdata/17/data \
  --jobs $(nproc) \
  2>&1 | tee /var/ftdata/17/data/upgrade.log
```

Если при миграции происходит ошибка, `pg_upgrade` завершает свою работу. Перед повторным запуском `pg_upgrade` следует восстановить кластер FTData 16 (см. Восстановление кластера БД FTData 16) и внести коррективы БД, чтобы `pg_upgrade` могла успешно восстановить схему. Если проблема возникла в модуле расширения, может потребоваться удалить его в исходном кластере, а затем установить его в кластере FTData 17 после миграции.

Скопировать конфигурацию кластера

Перенесите нестандартные настройки в файлах конфигурации `pg_hba.conf`,`postgresql.conf` и `postgresql.auto.conf` кластера БД FTData 16 в кластер БД FTData 17.

Запустить сервер СУБД FTData 17

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

Завершить миграцию вручную

Изучить лог `/var/ftdata/17/data/upgrade.log` утилиты `pg_upgrade`.

Там могут находиться инструкции, которые следует выполнить после запуска БД. Утилита может сгенерировать файлы скриптов, которые должны запускаться администратором. Эти скрипты будут подключаться к каждой базе данных, требующей дополнительных операций. Каждый такой скрипт следует выполнять командой:

```
/opt/ftdata/17/bin/psql {параметры подключения к БД} -f {имя файла скрипта для БД}
```

Поскольку статистика оптимизатора не передается `pg_upgrade`, утилита предложит запустить команду для восстановления этой информации на новой БД.

Удалить файлы кластера БД FTData 16

После того, как будет проверена работоспособность резервной копии данных БД FTData 16, можно удалить данные кластера, запустив скрипт, указанный в логе `/var/ftdata/17/data/upgrade.log`. Автоматическое удаление невозможно, если в старом каталоге данных находятся дополнительные табличные пространства.

Восстановление кластера БД FTData 16

Если после выполнения миграции `pg_upgrade` без опции `--link`, в кластере БД FTData 16 ничего не изменилось, кластер БД FTData 16 можно перезапустить:

```
sudo -u postgres /opt/ftdata/16/bin/pg_ctl -D /var/ftdata/16/data start
```

Если использовался ключ `--link`, у кластеров БД FTData 16 и 17 могут оказаться общие файлы данных:

- Если работа `pg_upgrade` была прервана до начала расстановки ссылок, в кластере FTData 16 ничего не изменилось; его можно просто перезапустить.
- Если вы не запускали кластер FTData 17, кластер FTData 16 не претерпел никаких изменений, за исключением того, что при создании ссылки на данные к имени `$PGDATA/global/pg_control` было добавлено окончание .old. Чтобы продолжить использование старого кластера, достаточно убрать окончание .old из имени файла `$PGDATA/global/pg_control`; после этого старый кластер можно будет перезапустить.
- Если вы запускали кластер FTData 17, он внёс изменения в общие файлы, и использовать кластер FTData 16 небезопасно. В этом случае старый кластер требуется восстановить из резервной копии.

Миграция кластера БД средствами pg_dumpall

Документация на утилиту `pg_upgrade` доступна в разделе pg_dumpall документации PostgreSQL 17.

Миграция средствами `pg_dumpall` - это создание дампа данных из версии FTData 16 и восстановление их в FTData 17.

Методы резервного копирования на уровне файловой системы не будут работать. (Существуют проверки, которые не позволяют вам использовать каталог данных с несовместимой версией `FTData`, поэтому не будет большого вреда, если вы попытаетесь запустить неправильную версию сервера в

каталоге данных).

В этом разделе предполагается, что ваша существующая установка находится в каталоге `/opt/ftdata/16/`, а файлы кластера БД будут перенесены из `/var/ftdata/16/data/` в `/var/ftdata/17/data/`.

Создать новый кластер БД FTData 17

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl init -D /var/ftdata/17/data/
```

Скопировать конфигурацию кластера

Перенесите нестандартные настройки в файлах конфигурации `pg_hba.conf`,`postgresql.conf` и `postgresql.auto.conf` кластера БД FTData 16 в кластер БД FTData 17.

Запустить сервер СУБД FTData 17

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

Переключить сервер СУБД FTData 16 на другой порт

```
sudo -u postgres sh -c 'echo "port = 5433" >> /var/ftdata/16/data/postgresql.conf'
sudo -u postgres /opt/ftdata/16/bin/pg_ctl -D /var/ftdata/16/data restart
```

Сделать резервную копию данных БД FTData 16

см. Сделать резервную копию данных БД FTData 16

Выполнить миграцию

Файл резервной копии `dumpall` и файлы конфигурации кластера будут помещены в директорию `/var/ftdata/16/bkp`.

При миграции убедитесь, что ваша база данных недоступна для записи. Это не влияет на целостность резервной копии, но измененные данные не будут включены. При необходимости отредактируйте разрешения в файле /var/ftdata/16/data/pg_hba.conf, чтобы запретить доступ всем, кроме вас.

```
sudo -u postgres sh -c '\
    /opt/ftdata/16/bin/pg_dumpall -p 5433 \
    | /opt/ftdata/17/bin/psql -d postgres'
```

Остановить сервер СУБД FTData 16

```
sudo -u postgres /opt/ftdata/16/bin/pg_ctl -D /var/ftdata/16/data start
```

Удалить файлы кластера БД FTData 16

После того, как будет проверена работоспособность резервной копии данных БД FTData 16, можно удалить данные кластера.

```
sudo -u postgres rm -rf /var/ftdata/16/data
```

Миграция кластера базы данных с применением логической репликации

Логическая репликация описана в разделе Chapter 31. Logical Replication документации PostgreSQL 17. Также рекомендуется ознакомиться с разделом 20.6. Replication документации.

Логическая репликация поддерживает репликацию между различными мажорными версиями FTData. При этом публикующий узел (publisher) и узел-подписчик (subscriber) могут находиться на одном сервере.

После синхронизации с публикующим узлом FTData 16 следует остановить его, сконфигурировать и запустить узел подписчик FTData 17 для работы вместо публикующего узла. Такое переключение приводит к простою всего в несколько секунд для миграции.

Этот метод миграции можно выполнить с использованием встроенных средств логической репликации, а также с использованием внешних систем логической репликации, таких как `pglogical`, `Slony`, `Londiste` и `Bucardo`.

Сбор диагностической информации СУБД FTData

Для оперативной поддержки СУБД FTData, в случае сбоев в работе сервера, требуется собирать диагностическую информацию. Необходимо сконфигурировать систему для сбора дампов памяти процессов (`core dump`), завершившихся нештатно.

Без дампа памяти процесса, завершившегося нештатно, работа поддержки будет замедлена, а в некоторых случаях невозможна до тех пор, пока не удастся собрать необходимую информацию, добившись аналогичного сбоя повторно.

Настройка сбора дампов памяти процессов

Дамп памяти (`core dump`) — это файл, в который записывается адресное пространство (память) процесса при его нештатном завершении. Дампы ядра могут создаваться по

требованию (например, отладчиком) или автоматически, при завершении процесса. Дампы памяти создаются ядром Linux при сбое программы и могут быть переданы утилите systemd-coredump для дальнейшей обработки.

Для сбора подробной информации о причинах сбоя СУБД FTData требуется разрешить сбор дампов памяти для процессов СУБД FTData и утилиты ftdbackrest.

> Для кластеров, развёрнутых в контейнерах Docker или аналогичных, следует настраивать сбор дампов памяти, в соответствии с документацией на систему контейнеризации. Приведённые ниже инструкции не учитывают особенности работы СУБД в контейнере. Следует также убедиться, что дамп памяти процесса, завершившегося нештатно, формируется.

Настройка ограничений на размер дампа памяти

Снять ограничения на размер дампов памяти для пользователя, которому принадлежит БД (по умолчанию `postgresql`), можно путем внесения изменения в конфигурационный файл:

```
sudo tee /etc/security/limits.d/10-cd.conf <<EOF
postgres hard core unlimited
postgres soft core unlimited
EOF
```

Изменения вступят в силу только после перезагрузки системы.

> Для уменьшения размера дампа памяти можно отключить сохранение данных из `shared_buffers` при помощи параметра `coredump_store_shared_buffers`.
>
> По умолчанию `coredump_store_shared_buffers=on`.

Настройка systemd-coredump

Для управления хранением дампов памяти рекомендуется использовать утилиту coredumpctl. Она может поставляться как в составе systemd, так и в виде отдельного пакета (например, `systemd-coredump` для Debian 11).

Информацию о coredumpctl можно получить в следующих источниках:

- Обзор coredumpctl на странице документации systemd.
- Документация для coredumpctl на ресурсе man7.
- Документация для coredump.conf на ресурсе man7.

Передача дампов памяти утилите coredumpctl

Необходимо внести изменения в конфигурационный файл:

```
sudo tee /etc/sysctl.d/10-cp.conf <<EOF
kernel.core_pattern = |/usr/lib/systemd/systemd-coredump %P %u %g %s %t %c %h %e
kernel.core_pipe_limit = 0
kernel.core_uses_pid = 0
EOF
```

Изменение директории дампов памяти

Файлы дампов памяти, собранные coredumpctl, по умолчанию будут сохраняться в директории `/var/lib/systemd/coredump`. Утилита coredumpctl не имеет параметра конфигурации для изменения директории, в котором будут храниться дампы памяти.

Для смены директории хранения дампов памяти необходимо использовать механизм bind-mount.

В файле `/etc/fstab` добавить строку для монтирования нужной директории для дампов памяти:

```
{абсолютный путь до директории} /var/lib/systemd/coredump none defaults,bind 0 0
```

Обновить конфигурацию системы:

```
mount /var/lib/systemd/coredump
systemctl daemon-reload
```

Настройка размеров хранилища и архивации дампов памяти

> Для сохранения дампа памяти в хранилище по пути `/var/lib/systemd/coredump` должно быть достаточно места для размещения несжатой копии данных из ОЗУ, зарезервированных процессом СУБД FTData - не менее 150 % от величины, заданной параметром `postgresql.conf shared_buffers`.

Ниже приведён пример для хранилища дампов памяти объемом 200 Гб:

```
sudo mkdir -p /etc/systemd/coredump.conf.d/
sudo tee /etc/systemd/coredump.conf.d/10-cdc.conf <<EOF
[Coredump]
Storage=external
Compress=yes
ProcessSizeMax=50G
ExternalSizeMax=50G
MaxUse=200G
KeepFree=1G
EOF
```

Рекомендуемые значения для `ProcessSizeMax` и `ExternalSizeMax` - не менее 150 % от величины, заданной параметром `postgresql.conf shared_buffers`.

Рекомендуемое значение `MaxUse` - не менее 320 % от `ProcessSizeMax`.

Более точно значения `ExternalSizeMax` и `MaxUse` можно подобрать при настройке кластера, проанализировав потребление ОЗУ процессами СУБД FTData. `ExternalSizeMax` должен превышать максимальное потребление памяти одним процессом СУБД FTData, как минимум, на 20 %.

Проверка генерации дампов памяти

Для полноты проверки, после внесения изменений в системные настройки дампов памяти, следует перезагрузить систему.

Имитация сбоя СУБД

Для имитации сбоя лучше использовать тестовый кластер. При этом postgresql.conf рекомендуется скопировать из основного кластера.

Имитировать сбой можно, выполнив следующие команды:

```
sudo mkdir /var/ftdata/17/test
sudo chmod 744 /var/ftdata/17/test
sudo chown postgres:postgres /var/ftdata/17/test
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/test initdb
sudo -u postgres bash -c "echo 'port = 1543' >> /var/ftdata/17/test/postgresql.conf"
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/test start
sudo kill -s SIGSEGV $(sudo cat /var/ftdata/17/test/postmaster.pid | sed -n 1p)
sudo rm -r /var/ftdata/17/test
```

Проверка наличия дампа памяти

Выполняется командой:

```
sudo coredumpctl info /opt/ftdata/17/bin/postgres
```

Значение `Storage` (в некоторых версиях - `Coredump`) должно указывать на файл:

```
Storage: /var/lib/systemd/coredump/core.postgres.26.265231304a0d4d7b8e68b2f0a554e1d2.67668.1695636212000000.lz4
```

Если указано значение `Storage`: `none`, то это значит, что дамп памяти не сохранился, и следует проверить настройки: `/etc/security/limits.d/10-cd.conf`

Сбор данных о сбое СУБД FTData

В некоторых случаях возможно возникновение сбоев СУБД FTData, большинство из которых выглядит, как неожиданное завершение серверного процесса, обрабатывающего запросы клиента (backend). В случае возникновения такого сбоя, при наличии вероятности повреждения общей памяти процессов, сервер перезагружается с немедленным завершением всех клиентских процессов и записью текущего состояния БД на диск (checkpoint). Причина сбоя фиксируется в файле журнала сервера.

Для устранения причин сбоя требуется подготовить и передать в техническую поддержку СУБД FTData следующие файлы:

- Информацию о версии FTData (см. Проверка версий ПО);
- Журналы сервера;
- Дампы памяти процессов.

Подготовка журнала сервера

Расположение файлов журналов настраивается параметром `log_directory` (в `postgresql.conf`), при этом относительные пути указаны для корневой директории БД. При перезапуске сервера может быть создан новый файл журнала, поэтому для полноты картины следует передать в техническую поддержку СУБД FTData два файла журнала (если файл только один, то - один) с наиболее поздней датой изменения.

Подготовка файла дампа памяти процесса

В случае нештатного завершения процесса генерируется дамп памяти процесса.

Наличие файла дампа памяти процесса можно проверить так:

```
sudo coredumpctl list /opt/ftdata/17/bin/postgres
```

В появившемся списке следует найти наиболее близкие по времени к сбою записи и экспортировать дампы памяти для них, указав PID в следующей команде (PID процесса, в котором возник сбой, можно также узнать из журнала сервера):

```
sudo coredumpctl -o core-dump-{PID} dump {PID} /opt/ftdata/17/bin/postgres
```

В результате, в текущей директории появится файл `core-dump-{PID процесса}`, который следует передать в техническую поддержку СУБД FTData.

> Файлы дампов памяти могут быть доступны для чтения не сразу после сбоя, так как некоторое время coredumpctl выполняет их обработку. В это время в директории `/usr/lib/systemd/systemd-coredump`, где хранится coredump, размещаются временные файлы; их имена начинаются с `.#core.postgres.`

Сбор диагностической информации о критических падениях

Данный функционал позволяет собирать информацию при падении соединений или сервера. Собранная информация записывается в файлы и хранится в соответствующей директории.

Данный функционал в настоящее время не поддерживается в Windows.

Содержание собранной диагностической информации

Диагностическая информация включает в себя следующие сведения:

- причину сбоя;
- время, когда произошел сбой (формат времени `YYYY-MM-DD HH:MM:SS`, локализованный под часовой пояс системы);
- id процесса, на котором произошел сбой;
- сообщение, если есть возможность получить его;
- текст запроса, на котором произошел сбой;
- список стека вызова функций;
- состояние регистров.

Причины сбоев для сбора диагностики

Данный функционал собирает информацию при следующих случаях:

- появление сигналов `SIGILL`, `SIGSEGV`, `SIGBUS`, `SIGFPE`, `SIGABRT`;
- при вызове утилит логирования `elog` и `ereport` с уровнем логирования `FATAL` и `PANIC`.

Конфигурационные параметры для сбора диагностики

Сбор диагностической информации регулируется конфигурационными параметрами, такими как:

- `crash_report_collect` - флаг, контролирующий включение и выключение сбора диагностики, по умолчанию имеет значение `off`;
- `crash_reports_dir` - строка, указывающая путь, где будет храниться вся собранная информация, по умолчанию имеет значение `"crash_reports"`.

Данные параметры нужно прописывать в файле `postgresql.conf`.

Местонахождение собранной диагностической информации

Так как все случаи, кроме `FATAL`, считаются крайне критичными (приводят к выключению сервера), то они записываются в заранее подготовленный файл `crash_reports.crp` в

директории, которая была указана через параметр crash_reports_dir. Для FATAL создаются файлы в той же директории в формате: crash_<time of crash>_<pid>.crp (формат времени YYYY-MM-DD-HH-MM-SS, локализованный под часовой пояс системы).

Ротация файлов диагностики

Есть функционал ротации собранной диагностической информации. Ротация происходит:

- в файле для сбора крайне критичных ситуаций (crash_reports.crp);
- в директории диагностики со всеми файлами.

Ротация в обоих случаях происходит при перезапуске сервера. Ротация поддерживает размер файла и суммарный размер файлов в директории, заданной через параметр.

Для управления ротацией в обоих случаях есть соответствующие параметры:

- crash_report_max_size - максимальный размер файла в мегабайтах, при достижении которого происходит ротация (значение по умолчанию: 16);
- crash_report_keep_size - максимальный размер всех файлов в директории для сбора диагностики (значение по умолчанию: 64). Данное значение должно быть больше crash_report_max_size, как минимум, в три раза.

Данные параметры нужно прописывать в файле postgresql.conf.

При ротации файла crash_reports.crp содержимое сохраняется в отдельный файл (формат файла: crash_report_<time of rotation>.crp) (формат времени YYYY-MM-DD-HH-MM-SS, локализованный под часовой пояс системы). При ротации директории удаляются старые файлы, пока суммарный размер файлов в директории больше значения из параметра.

Управление сервером СУБД FTData

Для контроля за состоянием процессов сервера предпочтительно использовать встроенные в ОС средства управления службами, такие как systemd или init.d. Операции, описанные ниже, приведут к однократному запуску сервера (сервер не будет запущен после перезагрузки ОС). Пример файла службы systemd расположен по пути: /lib/systemd/system/ftdata-17.service.

Создание БД

Здесь и далее предполагается, что работа ведётся с базой данных, расположенной по пути: /var/ftdata/17/data.

```
sudo mkdir -p /var/ftdata/17/data
sudo chown postgres:postgres -R /var/ftdata/17/data
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data init
```

Запуск сервера

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start \
 -l /var/ftdata/17/data/start.log
```

Если запуск будет неудачным, в файле `/var/ftdata/17/data/start.log` будут описаны причины сбоя.

```
sudo -u postgres tail /var/ftdata/17/data/start.log
```

Перезапуск сервера

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data restart
```

Загрузка файлов конфигурации без остановки сервера

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data reload
```

Проверка состояния сервера

```
/opt/ftdata/17/bin/pg_isready -U postgres
```

Остановка сервера

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
```

Работа с БД

Подключение к БД утилитой psql

Подключение будет успешным только в случае, если сервер СУБД запущен.

```
/opt/ftdata/17/bin/psql -d postgres -U postgres
```

Выполнение SQL-запроса утилитой psql

Подключение будет успешным только в случае, если сервер СУБД запущен. В примере будет выполнен запрос, будут напечатаны текст запроса и его результат, управление вернётся в терминал Linux.

```
/opt/ftdata/17/bin/psql -a -q -d postgres -U postgres -c \
 "/*текст запроса: */ SELECT version();"
```

Выполнение SQL из файла утилитой psql

Подключение будет успешным только в случае, если сервер СУБД запущен. В примере будет выполнены запрос, будут напечатаны текст запроса и его результат, управление вернётся в терминал Linux.

```
cat << EOF > ~/test.sql
-- текст запроса:
SELECT version();
EOF
/opt/ftdata/17/bin/psql -a -q -d postgres -U postgres -f ~/test.sql
```

Проверка версий ПО

Версия СУБД FTData

SQL запрос:

```
-- Версия СУБД FTData
SHOW ftd_server_version;
-- FTData | 17.5.3-1

-- Данные о сборке СУБД FTData
SHOW ftd_server_build;
-- 9b7f5ad6815 2025-09-04T12:00:00+07:00

-- Версия ядра PostgreSQL (формат не изменён)
SELECT version();
-- PostgreSQL 17.5 on x86_64-pc-linux-gnu, compiled by gcc (AstraLinuxSE 8.3.0-6) 8.3.0, 64-bit
```

Команда:

```
/opt/ftdata/17/bin/postgres --version
# postgres (PostgreSQL) 17.5
# FTData | 17.5.3-1 9b7f5ad6815 2025-09-04T12:00:00+07:00
```

Версия системного каталога СУБД FTData

SQL запрос:

```
SELECT catalog_version_no FROM pg_control_system();
```

Чтение файла:

```
cat /opt/ftdata/17/share/catalog_version
```

Версии доступных расширений

SQL запрос:

```
SELECT name,default_version FROM pg_available_extensions;
```

Конфигурирование сервера СУБД FTData

Настройка параметров

Инструкция по настройке и описание параметров сервера PostgreSQL приведены в разделе "Chapter 20. Server Configuration" документации PostgreSQL 17.

Сервер СУБД FTData 17 имеет те же параметры, что и PostgreSQL 17, а также дополнительные параметры, которые описаны ниже.

Список специальных параметров СУБД FTData

Потребление ресурсов

local_cache_subtrans_entries (integer)

Параметр задаёт количество последних результатов поиска в SLRU-кэше субтранзакций, хранимых в локальном кэше. Параметр описан в разделе "Локальный кэш субтранзакций".

transaction_buffers (integer)

Задаёт объём памяти, используемый для кэширования содержимого `pg_xact`. Если значение указано без единиц измерения, оно считается заданным в блоках (8192 байт). Значение по умолчанию — 0, которое означает, что будет использовано shared_buffers/512 блоков, но не более 1024 и не менее 16 блоков. Этот параметр можно задать только при запуске сервера. Параметр перенесён из PostgreSQL 17.0.

Статистика времени выполнения

ftd_xact_stat_collect (boolean)

Параметр включает сбор статистики для анализа производительности кэша субтранзакций. Данные размещаются в системном представлении `ftd_stat_subxids`. Параметр описан в разделе "Мониторинг использования кэша субтранзакций".

ftd_xact_stat_reset_on_overflow (boolean)

Параметр определяет поведение заполнения данными системного представления `ftd_stat_subxids` при переполнении счётчиков. Параметр описан в разделе "Мониторинг использования кэша субтранзакций".

Журнал предзаписи (WAL)

wal_compression (enum)

Это стандартный параметр PostgreSQL 17 (см. раздел "20.5. Write Ahead Log" документации).

Для СУБД FTData 17 установлено значение по умолчанию - `lz4` (для СУБД PostgreSQL значение `off`).

Параметр включает сжатие WAL указанным методом. Если этот параметр имеет значение `on`, сервер СУБД FTData сжимает образы полных страниц, записываемые в WAL, когда включён режим `full_page_writes` или при создании базовой копии. Сжатый образ страницы будет развёрнут при воспроизведении WAL. Параметр может принимать значения `pglz`, `lz4` и `zstd`, наряду со стандартными `on` и `off`, которые оставлены в целях обратной совместимости. Логические эквивалентные значения, такие как `on`, `true`, `yes` и `1`, эквивалентны `pglz`. Значение по умолчанию — `lz4`. Изменить этот параметр могут только суперпользователи и пользователи с соответствующим правом SET. Включив сжатие, вы можете без дополнительных рисков повреждения данных уменьшить объём WAL, ценой дополнительной нагрузки на процессор, связанной со сжатием данных при записи в WAL и разворачиванием их при воспроизведении WAL.

Планирование запросов

optimize_union_or (boolean)

Параметр включает оптимизацию запросов, содержащих оператор `OR`. Параметр описан в разделе "Оптимизация запросов с оператором ИЛИ".

Журналирование

log_lock_skip_locked (boolean)

Параметр включает функцию логирования неудачных блокировок отношений. Параметр описан в разделе "Логирование неудачных блокировок отношений".

log_lock_skip_locked_level (integer)

Параметр определяет уровень сообщений о неудачных блокировках отношений в логе сервера (и/или клиента). Параметр описан в разделе "Логирование неудачных блокировок отношений".

Параметры сжатия данных

zmd_search_free_space (enum)

Может принимать следующие значения: `none`, `current_buffer`, `recent_buffers`. Параметр описан в разделе "Поиск свободного места для записи сжатой страницы данных".

Предопределённые параметры

ftd_server_version (string)

Версия сервера СУБД FTData. Не изменяется.

Пример: `FTData | 17.5.3-1`.

ftd_server_build (string)

Версия исходного кода и дата сборки сервера СУБД FTData. Не изменяется.

Пример: `9b7f5ad6815 2025-09-04T12:00:00+07:00`.

ftd_client_id (string)

Строка-идентификатор клиента. Не изменяется.

Пример: `АО «СОВРЕМЕННЫЕ СИСТЕМЫ» ОГРН: 1065407112286`.

ftd_contract_id (string)

Строка номера договора. Не изменяется.

Пример: `0/0 от 2025-08-14`.

ftd_not_before (string)

Начало срока действия лицензии. Не изменяется.

Пример: `2025-08-21`.

ftd_not_after (string)

Конец срока действия лицензии. Не изменяется.

Пример: `2026-02-22`.

ftd_core_count (int)

Количество ядер. Не изменяется.

Пример: `64`.

ftd_edition (enum)

Вид сборки. Не изменяется.

Пример: `Enterprise`.

ftd_config (string)

Строка дополнительных параметров конфигурации сборки (может быть пустой). Не изменяется.

ftd_license_valid (enum)

Состояние лицензии. Равно `true`, если лицензия корректна, иначе - строка с описанием проблемы. Не изменяется.

Пример: `true`.

Работа с лицензией СУБД FTData

Получение лицензии

Для полноценной работы сервера СУБД FTData необходимо получить лицензию. Лицензия выдаётся на определённый срок, до наступления и после которого перестаёт быть действительной. Далее возможно продление срока лицензии.

Лицензия состоит из пары файлов: `license.yaml` и `license.bin`.

Лицензия действительна только для заранее определённой сборки FTData и типа (дистрибутива) операционной системы.

Активация лицензии

Полученные от поставщика СУБД файлы лицензии: `license.yaml` и `license.bin` необходимо поместить в директорию:

```
/opt/ftdata/M/share/postgresql
```

где `M` - мажорная версия PostgreSQL и FTData (`16` или `17`).

Лицензия будет загружена при следующем старте FTData.

Проверка состояния лицензии

Узнать состояние лицензии сервера FTData можно путём чтения следующих параметров конфигурации:

Имя параметра	Назначение
`ftd_client_id`	Строка-идентификатор клиента
`ftd_contract_id`	Строка номера договора
`ftd_not_before`	Начало срока действия лицензии
`ftd_not_after`	Конец срока действия лицензии
`ftd_core_count`	Количество ядер
`ftd_edition`	Вид сборки
`ftd_config`	Дополнительные параметры конфигурации сборки
`ftd_license_valid`	Состояние лицензии

Эти параметры невозможно изменить при помощи внесения их в `postgresql.conf` или путём выполнения SQL-запросов.

При отсутствии действующей лицензии параметр `ftd_edition` имеет значение `Trial`, и работа СУБД происходит с соответствующими ограничениями по количеству ядер и по сроку действия.

Дополнительные возможности мониторинга в СУБД FTData

Мониторинг активности процессов сервера

В СУБД FTData введены дополнительные параметры мониторинга состояния текущих активных процессов (клиентских и служебных), помимо уже имеющихся в СУБД PostgreSQL.

Представление pg_stat_activity

В представлении `pg_stat_activity` приведена информация о состоянии текущих активных процессов (клиентских и служебных) сервера СУБД FTData. Данное представление присутствует в СУБД PostgreSQL. В СУБД FTData добавлены поля:

- `backend_nsubxids` - полное количество субтранзакций в выбранной транзакции клиента, либо NULL - в случае отсутствия субтранзакций.
- `backend_subxids_cached` - количество субтранзакций, сохраненных в кэше субтранзакций в PGPROC (см. раздел Кэш субтранзакций в PGPROC), либо NULL - в случае отсутствия субтранзакций.

Поля модифицированного представления `pg_stat_activity`.

Атрибут	Тип	Описание
`datid`	oid	OID базы данных, к которой подключен данный серверный процесс.
`datname`	name	Имя базы данных, к которой подключен данный серверный процесс.
`pid`	integer	Идентификатор данного серверного процесса.
`leader_pid`	integer	Идентификатор процесса лидера параллельной группы, если этот процесс является обработчиком параллельного запроса. NULL, если этот процесс является лидером параллельной группы или не участвует в параллельных запросах.
`usesysid`	oid	OID пользователя, подключенного к данному серверному процессу.
`usename`	name	Имя пользователя, подключенного к данному серверному процессу.
`application_name`	text	Имя приложения, подключенного к данному серверному процессу.

Атрибут	Тип	Описание
`client_addr`	inet	IP-адрес клиента, подключенного к данному серверному процессу. Если это поле равно NULL, это означает, что клиент подключен через Unix-сокет на сервере, или что это внутренний процесс, такой как autovacuum.
`client_hostname`	text	Имя компьютера подключенного клиента, полученное в результате обратного DNS-поиска по адресу client_addr. NULL только для IP-соединений и только в том случае, если включен log_hostname.
`client_port`	integer	Номер порта TCP, который клиент использует для связи с данным серверным процессом, или -1, если используется Unix-сокет. Если это поле равно NULL, это означает, что это внутренний процесс сервера.
`backend_start`	timestamp with time zone	Время, когда этот процесс был запущен. Для клиентских серверных процессов это время подключения клиента к серверу.
`xact_start`	timestamp with time zone	Время, когда была начата текущая транзакция этого процесса, или NULL, если транзакция неактивна. Если текущий запрос является первой транзакцией, этот столбец равен столбцу query_start.
`query_start`	timestamp with time zone	Время, когда был запущен текущий активный запрос, или, если состояние не активно, когда был запущен последний запрос.
`state_change`	timestamp with time zone	Время, когда состояние было изменено в последний раз.
`wait_event_type`	text	Тип события, которого ожидает серверный процесс, если таковое имеется; иначе NULL.
`wait_event`	text	Имя события ожидания, если серверный процесс в настоящее время находится в состоянии ожидания, иначе NULL.

Атрибут	Тип	Описание
`state`	text	Текущее состояние данного серверного процесса: • active: выполняет запрос; • idle: ожидает новой команды клиента; • idle in transaction: находится в транзакции, но в данный момент не выполняет запрос; • idle in transaction (aborted): это состояние аналогично состоянию idle in transaction, за исключением того, что один из операторов в транзакции вызвал ошибку; • fastpath function call: выполняет функцию fast-path; • disabled: функция track_activities отключена.
`backend_xid`	xid	Идентификатор транзакции верхнего уровня данного серверного процесса, если таковой имеется.
`backend_xmin`	xid	Горизонт xmin текущего серверного процесса.
`query_id`	bigint	Идентификатор последнего запроса этого серверного процесса. Если состояние активно, в этом поле отображается идентификатор текущего выполняемого запроса. Во всех остальных состояниях отображается идентификатор последнего выполненного запроса. По умолчанию идентификаторы запросов не вычисляются, поэтому это поле будет равно нулю, если не включен параметр compute_query_id или не настроен сторонний модуль, вычисляющий идентификаторы запросов.
`query`	text	Текст последнего запроса этого серверного процесса. Если состояние активно, в этом поле отображается текущий выполняющийся запрос. Во всех остальных состояниях отображается последний выполненный запрос. По умолчанию текст запроса обрезается на 1024 байта; это значение можно изменить с помощью параметра track_activity_query_size.
`backend_type`	text	Тип текущего серверного процесса.
`backend_nsubxids`	smallint	Полное количество субтранзакций в выбранной транзакции клиента, либо NULL в случае отсутствия субтранзакций.
`backend_subxids_cached`	smallint	Количество субтранзакций, сохраненных в PGPROC структуре у клиента, либо NULL в случае отсутствия субтранзакций.

Функция pg_stat_get_activity

Синтаксис функции:

```
pg_stat_get_activity(pid)
```

Функция возвращает запись с информацией о состоянии активного процесса СУБД FTData с указанным идентификатором или по одной записи для каждого активного процесса если передан параметр `NULL`. Возвращаемые записи аналогичны по структуре записями представления `pg_stat_activity` (см. описание выше).

У функции один обязательный параметр: `pid` - идентификатор процесса или `NULL` для запроса информации по всем активным процессам.

Типы данных входных параметров	Возвращаемый тип данных
(integer)	SETOF record

Мониторинг использования кэша субтранзакций

Для ускорения работы кэша субтранзакций в СУБД FTData встроен локальный кэш субтранзакций (см. раздел Локальный кэш субтранзакций). Серверный процесс (backend) кэширует результаты поиска в SLRU-кэше субтранзакций в локальный кэш процесса.

Данные в представлении `ftd_stat_subxids` позволяют оценить долю использования локального кэша (как результат деления `cache_local` на `total`), а также отследить снижение нагрузки на SLRU-кэш субтранзакций (по `cache_slru` и `cache_disk`), вызванное использованием локального кэша.

Представление ftd_stat_subxids

Представление настраивается параметрами конфигурации `postgresql.conf` или SQL-командами. Для применения изменений не требуется перезагрузка сервера, достаточно перезагрузить конфигурацию.

Имя параметра	Назначение	Значение по умолчанию
`ftd_xact_stat_collect`	Флаг разрешения сбора статистики.	`off`
`ftd_xact_stat_reset_on_overflow`	Флаг сброса счётчиков при переполнении `total`: Если флаг выключен, значения счётчиков (кроме `overflow`) будут сброшены в 0. Если флаг включен, значения счётчиков (кроме `overflow`) будут поделены на 2.	`off`

Формат строк представления `ftd_stat_subxids`

Имя колонки	Тип данных	Описание
total	bigint	Счётчик обращений к кэшу на проверку активности субтранзакций (также включает обращения, в результате которых данные в кэше не найдены).
cache_pgproc	bigint	Счётчик обращений на проверку активности субтранзакций, обработанных кэшем субтранзакций в PGPROC (см. раздел Кэш субтранзакций в PGPROC).
cache_local	bigint	Счётчик обращений на проверку активности субтранзакций, обработанных локальным кэшем субтранзакций (см. раздел Локальный кэш субтранзакций).
cache_slru	bigint	Счётчик чтений страниц из разделяемой памяти SLRU-кэша субтранзакций для поиска родительской транзакции (см. раздел SLRU-кэш субтранзакций в pg_subtrans). Для одной операции поиска может потребоваться несколько обращений к SLRU (при этом счётчик total увеличится только на единицу).
cache_disk	bigint	Счётчик чтений страниц с диска SLRU-кэша субтранзакций для поиска родительской транзакции (см. раздел SLRU-кэш субтранзакций в pg_subtrans). Для одной операции поиска может потребоваться несколько обращений к SLRU (при этом счётчик total увеличится только на единицу).
overflow	bigint	Счётчик переполнений total, увеличивается на 1 при переполнении. Поведение остальных счётчиков регулируется флагом ftd_xact_stat_reset_on_overflow.

В представлении pg_stat_slru счётчики blks_hit и blks_read учитывают все чтения из SLRU-кэша субтранзакций, в ftd_stat_subxids аналогичные счётчики (cache_slru и cache_disk соответственно) учитывают только обращения к SLRU-кэшу субтранзакций, результат которых сохраняется в локальный кэш.

Функция ftd_stat_clean_subxids

Синтаксис функции:

```
ftd_stat_clean_subxids()
```

Функция используется для сброса счётчиков (всем счётчиками присваивается значение 0) из представления ftd_stat_subxids до завершения транзакции. Возвращает целое число (bigint) - значение total перед сбросом.

Логирование неудачных блокировок отношений

СУБД FTData и PostgreSQL поддерживают механизм явной блокировки отношений (для всего отношения полностью, постранично и построчно). Данные блокировки можно захватывать в неблокирующих режимах:

- `NOWAIT` - если захват блокировки не удался, будет выдано сообщение об ошибке.
- `SKIP LOCKED` - если захват блокировки не удался, результат запроса не будет учитывать данные из отношения, для которых не удалось захватить блокировку.

Такие блокировки поддерживаются командами `SELECT`, `LOCK`, `VACUUM`, `ANALYZE`.

Для выявления потенциальных проблем, связанных с такими блокировками, добавлен дополнительный режим логирования. При провале попытки блокировки в режиме `NOWAIT` или `SKIP LOCKED` в логе сервера (и/или логе клиента) появляется сообщение, формат которого описан ниже.

Включение логирования неудачных блокировок отношений

Функция включается параметрами конфигурации `postgresql.conf` или SQL-командами. Для применения изменений не требуется перезагрузка сервера, достаточно перезагрузить конфигурацию.

Имя параметра	Назначение	Значение по умолчанию
`log_lock_skip_locked`	Флаг разрешения логирования неудачных блокировок отношений.	`off`
`log_lock_skip_locked_level`	Уровень сообщений о неудачных блокировках отношений в логе сервера (и/или клиента). Допустимые значения: `DEBUG5`, `DEBUG4`, `DEBUG3`, `DEBUG2`, `DEBUG1`, `INFO`, `NOTICE`, `WARNING`, `ERROR`, `LOG`, `FATAL`.	`WARNING`

Если уровень сообщений, установленный параметром `log_lock_skip_locked_level`, ниже уровня `log_min_messages`, то логирование не выполняется, - не производится сбор данных.

Формат сообщений в логе

Сообщение состоит из нескольких частей:

- Фиксированный заголовок (`LockAcquire: conditional lock failed`);
- Информация о блокировке, которую не удалось получить (начинается с `Lock:`);
- Информация о процессе, выполнявшем блокировку (начинается с `Acquirer:`);
- Информация о процессах, удерживавших блокировку (начинается с `Holders:`).

```
LockAcquire: conditional lock failed
 Lock: mode: {mode} {object}
 Acquirer: pid: {pid} usesysid: {role id} xid: {xid}
 Holders:
  type: {conflict} pid: {pid} usesysid: {role id} xid: {xid} mode: {mode}
  type: {conflict} pid: {pid} usesysid: {role id} xid: {xid} mode: {mode}
```

Где:

- `{mode}` - обозначение режима (или нескольких режимов) блокировки (см. документацию PostgreSQL 17 13.3.1. Table-Level Locks и 13.3.2. Row-Level Locks);
- `{object}` - описание блокированного объекта:
  - Для всего отношения `database: {DB id} relation: {rel id} ({rel name})`;
  - Для страницы отношения `database: {DB id} relation: {rel id} ({rel name}) page: {page}`;
  - Для кортежа отношения `database: {DB id} relation: {rel id} ({rel name}) page: {page} tuple: {tuple}`;
  - Для транзакции (блокировка кортежа) `xid: {xid}`.
    - `{DB id}` - идентификатор (OID) базы данных в pg_database или строка `shared` для общих отношений кластера (например, `pg_database`);
    - `{rel oid}` и `{rel name}` - идентификатор (OID) и имя представления в pg_class;
    - `{page}` - номер страницы отношения;
    - `{tuple}` - номер кортежа отношения.
- `{pid}` - pid бэкенд процесса;
- `{role id}` - идентификатор (OID) роли в pg_authid;
- `{xid}` - номер транзакции выполняемой процессом;
- `{conflict}` - состояние конфликта: `no conflict` - взятые данным процессом блокировки не конфликтуют с запрашиваемой блокировкой, `conflict` - конфликтуют.

Пример сообщения

```
-- Запрос блокировки
LOCK TABLE test IN ROW SHARE MODE NOWAIT;
-- Сообщение в логе сервера
WARNING: LockAcquire: conditional lock failed (regression)
 Lock: mode: RowExclusiveLock database: 24398 relation: 26387 (test)
 Acquirer: pid: 9059 usesysid: 223 xid: 851
 Holders:
  type: conflict pid: 9001 usesysid: 220 xid: 750 mode: AccessShareLock ShareRowExclusiveLock
  type: no conflict pid: 9010 usesysid: 222 xid: 765 mode: AccessShareLock RowShareLock
```

Процессу с pid 9059 не удалось заблокировать отношение `test` в режиме `RowExclusiveLock`, т.к.

процесс с pid 9001 заблокировал его в режимах AccessShareLock и ShareRowExclusiveLock (конфликтует с RowExclusiveLock). Также блокировка отношения удерживается процессом с pid 9010 в режимах AccessShareLock и RowShareLock, но эти режимы не конфликтуют с RowExclusiveLock.

Статистика планирования

В состав серверного пакета СУБД FTData входит доработанное расширение pg_stat_statements. Помимо представлений pg_stat_statements, pg_stat_statements_info и функции pg_stat_statements_reset, добавлены представления ftd_stat_plans и ftd_stat_mem_usage для сбора статистики планирования.

Представление ftd_stat_plans хранит выдачу SQL-команды EXPLAIN для планов запросов из представления pg_stat_statements. В представлении статистика планирования и выполнения запросов разделена для общих и частных планов (см. Общие и частные планы) с одинаковой структурой (см. Идентификатор структуры плана). Такое разделение позволяет оценить эффективность работы кэша подготовленных выражений.

Представление ftd_stat_mem_usage используется для настройки размеров хранилища статистики планирования в разделяемой памяти.

Параметры конфигурации сервера для сбора статистики планирования

Сбор статистики планирования настраивается параметрами конфигурации postgresql.conf или SQL-командами. Для применения изменений требуется перезагрузка сервера.

Имя параметра	Назначение	Значение по умолчанию	Диапазон
ftd_stat_plans.enable	Флаг разрешения сбора статистики планирования.	off	on / off
ftd_stat_plans.shmem_storage_size	Размер хранилища статистики планирования в разделяемой памяти, в мегабайтах.	10	1..200
ftd_stat_plans.split_by_constid	Режим сбора планов запросов. on - планы для запросов с одинаковыми структурами (planid), но разными значениями констант, хранятся в отдельных строках ftd_stat_plans.	on	on / off

Подключение расширения pg_stat_statements в БД для сбора статистики планирования

Настроить параметры postgresql.conf

- Включить загрузку библиотеки `pg_stat_statements.so` при старте сервера СУБД:

```
shared_preload_libraries = 'pg_stat_statements'
```

- Разрешить работу `pg_stat_statements` и включить отслеживание операций планирования:
  - `pg_stat_statements.max` - должен быть больше 100 (значение по умолчанию - `5000`);
  - `pg_stat_statements.track` - должен быть установлен как `all` или `top` (значение по умолчанию - `top`);
  - `pg_stat_statements.track_planning` - должен быть установлен как `on` (значение по умолчанию - `off`);
  - `compute_query_id` - должен быть установлен как `on` (значение по умолчанию - `off`).
- Включить сбор статистики планирования:
  - `ftd_stat_plans.enable` - должен быть установлен как `on` (значение по умолчанию - `off`);
  - `ftd_stat_plans.shmem_storage_size` - можно не изменять (значение по умолчанию - `10`);

Пример содержания `postgresql.conf`:

```
# Отслеживание планирования запросов pg_stat_statements
shared_preload_libraries = 'pg_stat_statements'
pg_stat_statements.max = 5000 # можно выбрать любое значение больше 100
pg_stat_statements.track = top # можно также выбрать all
pg_stat_statements.track_planning = on
compute_query_id = on
# Сбор статистики планирования запросов ftd_stat_plans
ftd_stat_plans.enable = on
ftd_stat_plans.shmem_storage_size = 10 # в МБ, от 1 до 200 МБ
```

Перезагрузить сервер СУБД

```
/opt/ftdata/17/bin/pg_ctl reload -D {путь до БД}
```

Установить расширение

```
CREATE EXTENSION pg_stat_statements;
```

Представление ftd_stat_plans

Представление содержит данные статистики планирования для запросов, представленных в `pg_stat_statements`. Это представление содержит отдельные строки для каждой

комбинации идентификатора пользователя, идентификатора базы данных, признака запроса верхнего уровня, идентификатора запроса, идентификатора структуры плана (один запрос может иметь несколько вариантов плана) и идентификатора набора констант.

Если параметр `pg_stat_statements.save` установлен как `on` (значение по умолчанию), при завершении работы сервера СУБД данные из представления `ftd_stat_plans` сохраняются на диск (данные `pg_stat_statements` также сохранятся), и при следующем запуске сервера данные восстановятся. Во время работы СУБД данные `ftd_stat_plans` не копируются на диск.

При вызове SQL-функции `pg_stat_statements_reset()` данные будут удалены из представления `ftd_stat_plans` (данные `pg_stat_statements` также удалятся).

Для просмотра текстов планов, собранных для запросов, выполненных другими пользователями, у текущего пользователя должна быть роль `pg_read_all_stats`. Если такой роли у пользователя нет, вместо текста `EXPLAN` в колонке `plan` будет передана строка `<insufficient privilege>`. Получить роль `pg_read_all_stats` можно, выполнив SQL-команду:

```
GRANT pg_read_all_stats TO current_user;
```

Формат строк представления `ftd_stat_plans`:

Имя колонки	Тип данных	Описание
`userid`	`oid`	OID пользователя, выполнявшего запрос.
`dbid`	`oid`	OID базы данных, в которой выполнялся запрос.
`queryid`	`bigint`	Идентификатор запроса - хеш-код, вычисленный по дереву запроса (Query).
`constid`	`int`	Идентификатор констант - хеш-код значений констант в запросе. `0` - если план захвачен в режиме `ftd_stat_plans.split_by_constid = off`.
`planid`	`bigint`	Идентификатор запроса структуры плана - хеш-код, вычисленный по дереву плана запроса (см. Идентификатор структуры плана).
`plan`	`text`	План запроса в формате, аналогичном выдаче команды `EXPLAIN`, и список значений параметров, при которых был захвачен план (параметры длиной более 256 символов выводятся не полностью). В режиме `ftd_stat_plans.split_by_constid = off` также добавляется значение `constid` запроса, при котором выполнен захват плана.
`query`	`text`	План запроса с константами.
`gen_plans`	`bigint`	Число операций планирования общего (generic) плана для этой структуры плана (см. Общие и частные планы).

Имя колонки	Тип данных	Описание
gen_min_plan_time	float8	Минимальное время, затраченное на планирование общего (generic) плана для этой структуры плана.
gen_max_plan_time	float8	Максимальное время, затраченное на планирование общего (generic) плана для этой структуры плана.
gen_mean_plan_time	float8	Среднее время, затраченное на планирование общего (generic) плана для этой структуры плана.
gen_min_start_cost	float8	Минимальная стоимость запуска плана (startup cost), рассчитанная при планировании общего (generic) плана для этой структуры плана.
gen_max_start_cost	float8	Максимальная стоимость запуска плана (startup cost), рассчитанная при планировании общего (generic) плана для этой структуры плана.
gen_mean_start_cost	float8	Средняя стоимость запуска плана (startup cost), рассчитанная при планировании общего (generic) плана для этой структуры плана.
gen_min_total_cost	float8	Минимальная стоимость выполнения плана (total cost), рассчитанная при планировании общего (generic) плана для этой структуры плана.
gen_max_total_cost	float8	Максимальная стоимость выполнения плана (total cost), рассчитанная при планировании общего (generic) плана для этой структуры плана.
gen_mean_total_cost	float8	Средняя стоимость выполнения плана (total cost), рассчитанная при планировании общего (generic) плана для этой структуры плана.
gen_calls	bigint	Число операций выполнения общего (generic) плана для этой структуры плана.
gen_min_exec_time	float8	Минимальное время, затраченное на выполнение общего (generic) плана для этой структуры плана.
gen_max_exec_time	float8	Максимальное время, затраченное на выполнение общего (generic) плана для этой структуры плана.
gen_mean_exec_time	float8	Среднее время, затраченное на выполнение общего (generic) плана для этой структуры плана.
cst_plans	bigint	Число операций планирования частного (custom) плана для этой структуры плана (см. Общие и частные планы).
cst_min_plan_time	float8	Минимальное время, затраченное на планирование частного (custom) плана для этой структуры плана.
cst_max_plan_time	float8	Максимальное время, затраченное на планирование частного (custom) плана для этой структуры плана.

Имя колонки	Тип данных	Описание
cst_mean_plan_time	float8	Среднее время, затраченное на планирование частного (custom) плана для этой структуры плана.
cst_min_start_cost	float8	Минимальная стоимость запуска плана (startup cost), рассчитанная при планировании частного (custom) плана для этой структуры плана.
cst_max_start_cost	float8	Максимальная стоимость запуска плана (startup cost), рассчитанная при планировании частного (custom) плана для этой структуры плана.
cst_mean_start_cost	float8	Средняя стоимость запуска плана (startup cost), рассчитанная при планировании частного (custom) плана для этой структуры плана.
cst_min_total_cost	float8	Минимальная стоимость выполнения плана (total cost), рассчитанная при планировании частного (custom) плана для этой структуры плана.
cst_max_total_cost	float8	Максимальная стоимость выполнения плана (total cost), рассчитанная при планировании частного (custom) плана для этой структуры плана.
cst_mean_total_cost	float8	Средняя стоимость выполнения плана (total cost), рассчитанная при планировании частного (custom) плана для этой структуры плана.
cst_calls	bigint	Число операций выполнения частного (custom) плана для этой структуры плана.
cst_min_exec_time	float8	Минимальное время, затраченное на выполнение частного (custom) плана для этой структуры плана.
cst_max_exec_time	float8	Максимальное время, затраченное на выполнение частного (custom) плана для этой структуры плана.
cst_mean_exec_time	float8	Среднее время, затраченное на выполнение частного (custom) плана для этой структуры плана.
mem_usage	int	Объем памяти, занимаемый информацией в хранилище статистики (в разделяемой памяти), в байтах.

Представление ftd_stat_mem_usage

Представление содержит данные об использовании хранилища статистики в разделяемой памяти.

Имя колонки	Тип данных	Описание
usage	bigint	Размер памяти, занимаемый использованными страницами хранилища (кратен размеру страницы - 8192 байта), в байтах.

Имя колонки	Тип данных	Описание
usage_max	bigint	Максимальный размер памяти, занимаемый использованными страницами хранилища (кратен размеру страницы - 8192 байта), в байтах.
total	bigint	Полный размер хранилища (кратен размеру страницы - 8192 байта), в байтах.
allocs	bigint	Счётчик количества выделенных для хранения статистики (для одной структуры плана одного запроса) участков памяти.
envicted	bigint	Счётчик количества удалённых для повторного использования участков памяти (сбрасывается функцией pg_stat_statements_reset()).

По соотношению usage_max и total можно оценить, насколько точно подобран параметр ftd_stat_plans.shmem_storage_size. Чем ближе usage_max к total, тем рациональнее используется разделяемая память.

Если usage_max составляет 97% и более от total, следует рассмотреть возможность увеличения ftd_stat_plans.shmem_storage_size. Если envicted растёт в течение нескольких часов с относительной скоростью (разы в единицу времени), сопоставимой с allocs, следует увеличить ftd_stat_plans.shmem_storage_size (не все варианты планов помещаются в хранилище).

Выдача планов и идентификаторов запросов

СУБД FTData и PostgreSQL поддерживают выдачу планов запросов (query plans) при помощи SQL-команды EXPLAIN. В FTData этот механизм расширен: помимо самого плана запроса, статистики и queryid (см. Идентификаторы запроса), возможно вывести:

- constid (см. Идентификаторы запроса);
- planid (см. Идентификатор структуры плана).

Вывод constid и planid

Параметры выводятся в результате команды EXPLAIN, если:

- в EXPLAIN передана опция VERBOSE;
- параметр конфигурации compute_query_id равен on;
- для constid: constid вычислен (не равен 0).

Хинты

Общие сведения

Расширение `ftd_hints` предназначено для того, чтобы пользователь мог повлиять на выбор планировщиком того или иного плана выполнения запроса.

Для того, чтобы воспользоваться хинтами, нужно загрузить библиотеку `ftd_hints`. С правами суперпользователя это можно сделать командой:

```
load 'ftd_hints';
```

Чтобы не суперпользователь мог загрузить библиотеку, ее необходимо скопировать в каталог `/opt/ftdata/17/lib/plugins` и загружать командой:

```
load '/opt/ftdata/17/lib/plugins/ftd_hints';
```

Префикс `/opt/ftdata/17/lib/plugins` - обязателен.

Альтернативный вариант загрузки библиотеки - добавить `ftd_hints` в список `shared_preload_libraries` в конфигурационном файле `postgresql.conf`:

```
shared_preload_libraries = 'ftd_hints'
```

Пока расширение не будет загружено, все хинты будут считаться только комментариями и не будут обработаны.

Все хинты должны располагаться внутри комментариев:

```
/*+ hint1(params) hint2(params) */
```

Знак `'+'` обязателен, внутри одного комментария может быть больше одного хинта:

```
select /*+ cardinality(employee 100) index(employee employee_cool_idx) */ id, name from employee where dep_id = 100;
```

Комментарий, содержащий хинты, должен располагаться сразу после ключевого слова `SELECT`. Все остальные комментарии, содержащие хинты, будут проигнорированы.

С целью отладки предусмотрена переменная `ftd_hints.log_level`. Если установить ее в значение `info`, то в консоли будет отображен список распознанных хинтов с параметрами и информация, был ли использован данный хинт:

```
alex=# load 'ftd_hints'; set ftd_hints.log_level = info;
LOAD
SET
alex=# select /*+ cardinality(pg_database 13) */ oid, datname from pg_database;
INFO:  Cardinality(alias = pg_database, nrows = 13.000000) - used
  oid  |  datname
-------+-----------
 13599 | postgres
 16384 | alex
     1 | template1
 13598 | template0
(4 rows)
```

Cardinality

Хинт `+cardinality` принудительно устанавливает оценочное количество строк в таблице, в результате чего стоимости разных типов соединений с этой таблицей будут отличаться от той, которая была бы рассчитана встроенными алгоритмами оценки на основе статистики. Что в свою очередь влияет на выбор планировщиком того или иного типа соединения.

Хинт `+cardinality` стоит применять, когда планировщик некорректно оценивает число строк в результате выборки. Такая ситуация может возникнуть, например, при работе с подзапросами.

Рассмотрим следующий пример: планировщик определил ожидаемое количество строк как 50, тогда как в действительности их всего 3, а тип соединения — Hash Join

```
create table t1 as select id as id, 'alex'||id as name from generate_series(1, 10000) as id;
create index on t1(id);
explain analyze select * from ( select * from t1 where id%3000=0) ttt join t1 on ttt.id=t1.id;
                                                    QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Hash Join  (cost=205.62..398.62 rows=50 width=24) (actual time=1.008..1.840 rows=3 loops=1)
   Hash Cond: (t1.id = t1_1.id)
   ->  Seq Scan on t1  (cost=0.00..155.00 rows=10000 width=12) (actual time=0.008..0.541 rows=10000 loops=1)
   ->  Hash  (cost=205.00..205.00 rows=50 width=12) (actual time=0.612..0.612 rows=3 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 9kB
         ->  Seq Scan on t1 t1_1  (cost=0.00..205.00 rows=50 width=12) (actual time=0.196..0.608 rows=3 loops=1)
               Filter: ((id % 3000) = 0)
               Rows Removed by Filter: 9997
 Planning Time: 0.219 ms
 Execution Time: 1.862 ms
(10 rows)
```

Если же подсказать планировщику точное количество строк, план будет другим, время выполнения сократится, а тип соединения будет Nested Loop:

```
alex=# explain analyze select /*+cardinality(ttt 3)*/ * from ( select * from t1 where id%3000=0) ttt join t1 on
ttt.id=t1.id;
INFO:  Cardinality(alias = ttt, nrows = 3.000000) - used
                                                    QUERY PLAN
```

```
-----------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.29..230.44 rows=3 width=24) (actual time=0.206..0.622 rows=3 loops=1)
   ->  Seq Scan on t1 t1_1  (cost=0.00..205.00 rows=50 width=12) (actual time=0.196..0.604 rows=3 loops=1)
         Filter: ((id % 3000) = 0)
         Rows Removed by Filter: 9997
   ->  Index Scan using t1_id_idx on t1  (cost=0.29..8.30 rows=1 width=12) (actual time=0.004..0.004 rows=1 loops=3)
         Index Cond: (id = t1_1.id)
 Planning Time: 0.237 ms
 Execution Time: 0.657 ms

(8 rows)
```

Index

Хинт `+index` заставляет PostgreSQL использовать указанный индекс при построении плана запроса, если указанный индекс существует и применим.

Пример (таблица с 2 индексами):

```
alex=# \d t1
                  Table "public.t1"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           |          |
 name   | text    |           |          |
Indexes:
    "t1_id_idx" btree (id)
    "t1_name_idx" btree (name)
```

По умолчанию для выполнения запроса:

```
explain select * from t1 where id < 100 and name = 'alex11'
```

PostgreSQL будет использовать индекс t1_name_idx:

```
alex=# explain select * from t1 where id < 100 and name = 'alex11';
                              QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using t1_name_idx on t1  (cost=0.29..8.30 rows=1 width=12)
   Index Cond: (name = 'alex11'::text)
   Filter: (id < 100)
(3 rows)
```

с помощью хинта `index` можно заставить PostgreSQL использовать индекс t1_id_idx:

```
alex=# explain select /*+index(t1 t1_id_idx)*/ * from t1 where id < 100 and name = 'alex11';
INFO:  Index(alias = t1, indexes = [t1_id_idx]) - used
                              QUERY PLAN
```

```
-----------------------------------------------------------------------
 Index Scan using t1_id_idx on t1  (cost=0.29..10.27 rows=1 width=12)
   Index Cond: (id < 100)
   Filter: (name = 'alex11'::text)
(3 rows)
```

Materialize

Хинт `+materialize` заставляет PostgreSQL материализовать результат подзапроса, в котором он указан. Подзапрос будет материализован в том случае, если выбранный планировщиком план его выполнения не предполагает материализации:

```
explain analyze select * from (select /*+materialize*/ * from t1 where id%2=0 ) ttt where name similar to 'alex12%';
INFO:  Materialize - used
                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Subquery Scan on ttt  (cost=0.00..205.88 rows=1 width=12) (actual time=0.020..3.598 rows=56 loops=1)
   Filter: (ttt.name ~ '^(?:alex12.*)$'::text)
   Rows Removed by Filter: 4944
   ->  Materialize  (cost=0.00..205.25 rows=50 width=12) (actual time=0.014..1.158 rows=5000 loops=1)
         ->  Seq Scan on t1  (cost=0.00..205.00 rows=50 width=12) (actual time=0.012..0.767 rows=5000 loops=1)
               Filter: ((id % 2) = 0)
               Rows Removed by Filter: 5000
 Planning Time: 0.207 ms
 Execution Time: 3.619 ms
(9 rows)
```

Подзапрос не будет материализован в случае, если результат его выполнения сгенерирован функцией:

```
alex=# explain select * from (select /*+materialize*/ id from unnest(ARRAY[1,2,3]) as id) tt join t1 on t1.id=tt.id;
INFO:  Materialize - not used
QUERY PLAN
-----------------------------------------------------------------------------
 Nested Loop  (cost=0.29..25.00 rows=3 width=16)
   ->  Function Scan on unnest id  (cost=0.00..0.03 rows=3 width=4)
   ->  Index Scan using t1_id_idx on t1  (cost=0.29..8.30 rows=1 width=12)
         Index Cond: (id = id.id)
(4 rows)
```

Материализация может быть полезна, если имеется несколько тяжелых подзапросов, результат выполнения которых мы собираемся соединить:

```
create table t1 as select id as id, 'alex'||id as name from generate_series(1, 10000) as id;
alex=# explain analyze select count(*)from
    ( select * from t1 where name similar to 'alex12%') tt1,
    ( select * from t1 where name similar to 'alex123%') tt2,
    ( select * from t1 where name similar to 'alex1234%') tt3
where (tt1.id=tt2.id and tt1.name=tt2.name) or (tt1.id=tt3.id and tt3.name=tt1.name) or (tt2.id=tt3.id and
tt3.name=tt2.name);
                                                                        QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------
```

```
---------------
 Aggregate  (cost=542.54..542.55 rows=1 width=8) (actual time=144.774..144.777 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..542.53 rows=1 width=0) (actual time=1.113..144.761 rows=131 loops=1)
         Join Filter: (((t1.id = t1_1.id) AND (t1.name = t1_1.name)) OR ((t1.id = t1_2.id) AND (t1_2.name = t1.name)) OR
((t1_1.id = t1_2.id) AND (t1_2.name
= t1_1.name)))
         Rows Removed by Join Filter: 1090
         ->  Nested Loop  (cost=0.00..360.01 rows=1 width=24) (actual time=1.046..91.142 rows=11 loops=1)
               ->  Seq Scan on t1 t1_1  (cost=0.00..180.00 rows=1 width=12) (actual time=0.077..5.176 rows=11 loops=1)
                     Filter: (name ~ '^(?:alex123.*)$'::text)
                     Rows Removed by Filter: 9989
               ->  Seq Scan on t1 t1_2  (cost=0.00..180.00 rows=1 width=12) (actual time=0.986..7.812 rows=1 loops=11)
                     Filter: (name ~ '^(?:alex1234.*)$'::text)
                     Rows Removed by Filter: 9999
         ->  Seq Scan on t1  (cost=0.00..180.00 rows=101 width=12) (actual time=0.008..4.864 rows=111 loops=11)
               Filter: (name ~ '^(?:alex12.*)$'::text)
               Rows Removed by Filter: 9889
 Planning Time: 0.339 ms
 Execution Time: 144.811 ms
(16 rows)
```

Если же материализовать подзапросы, время выполнения существенно сократится:

```
alex=# explain analyze select count(*)from
alex-#     ( select /*+materialize*/ * from t1 where name similar to 'alex12%') tt1,
alex-#     ( select /*+materialize*/ * from t1 where name similar to 'alex123%') tt2,
alex-#     ( select /*+materialize*/ * from t1 where name similar to 'alex1234%') tt3
alex-# where (tt1.id=tt2.id and tt1.name=tt2.name) or (tt1.id=tt3.id and tt3.name=tt1.name) or (tt2.id=tt3.id and
tt3.name=tt2.name);
INFO:  Materialize - used
INFO:  Materialize - used
INFO:  Materialize - used
                                                                                    QUERY PLAN

-----------------------------------------------------------------------------------------------------------------------
-----------------------------------
---------------
 Aggregate  (cost=544.33..544.34 rows=1 width=8) (actual time=17.976..17.978 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..544.08 rows=101 width=0) (actual time=1.132..17.967 rows=131 loops=1)
         Join Filter: (((t1_2.id = t1.id) AND (t1_2.name = t1.name)) OR ((t1_2.id = t1_1.id) AND (t1_1.name = t1_2.name))
OR ((t1.id = t1_1.id) AND (t1_1.nam
e = t1.name)))
         Rows Removed by Join Filter: 1090
         ->  Nested Loop  (cost=0.00..360.04 rows=1 width=24) (actual time=1.063..12.907 rows=11 loops=1)
               ->  Materialize  (cost=0.00..180.00 rows=1 width=12) (actual time=0.079..5.054 rows=11 loops=1)
                     ->  Seq Scan on t1  (cost=0.00..180.00 rows=1 width=12) (actual time=0.078..5.051 rows=11 loops=1)
                           Filter: (name ~ '^(?:alex123.*)$'::text)
                           Rows Removed by Filter: 9989
               ->  Materialize  (cost=0.00..180.00 rows=1 width=12) (actual time=0.089..0.713 rows=1 loops=11)
                     ->  Seq Scan on t1 t1_1  (cost=0.00..180.00 rows=1 width=12) (actual time=0.981..7.845 rows=1
loops=1)
                           Filter: (name ~ '^(?:alex1234.*)$'::text)
                           Rows Removed by Filter: 9999
         ->  Materialize  (cost=0.00..180.50 rows=101 width=12) (actual time=0.001..0.450 rows=111 loops=11)
               ->  Seq Scan on t1 t1_2  (cost=0.00..180.00 rows=101 width=12) (actual time=0.011..4.895 rows=111 loops=1)
                     Filter: (name ~ '^(?:alex12.*)$'::text)
                     Rows Removed by Filter: 9889
 Planning Time: 0.386 ms
 Execution Time: 18.015 ms
(19 rows)
```

Parallel

Хинт `+parallel` позволяет выполнить параллельное сканирование или соединение таблиц. Если в хинте указано имя таблицы, хинт будет указывать планировщику построить план параллельного сканирования этой таблицы. Если в хинте не указано имя таблицы, хинт будет указывать планировщику построить план параллельного соединения таблиц в том (под)запросе, в котором присутствует хинт.

По умолчанию, если в хинте не указано количество параллельных процессов, планировщик использует значение, установленное в `ftd_hints.max_parallel_workers`. Если `ftd_hints.max_parallel_workers` не установлено, оно принимается равным `max_parallel_workers`.

```
alex=# set ftd_hints.max_parallel_workers = 13;
SET
```

Следует иметь в виду, что запланированное количество процессов не означает, что при выполнении запроса все они будут запущены, так как на реальное количество процессов, выделенное на выполнение запроса, влияет параметр `max_worker_processes`, а также количество процессов, выполняющихся в текущий момент.

Пример (параллельное сканирование таблицы):

```
alex=# explain (costs off) select /*+parallel(8)*/ * from t1;
INFO:  Parallel(parallel_workers = 8.000000) - used
          QUERY PLAN
-------------------------------
 Gather
   Workers Planned: 4
   ->  Parallel Seq Scan on t1
(3 rows)
```

Пример (параллельное сканирование одной из таблиц в запросе):

```
alex=# explain (costs off) select /*+parallel(t2, 8)*/ * from t1 join t2 on t1.id=t2.t1_id;
INFO:  Parallel(alias = t2, parallel_workers = 8.000000) - used
             QUERY PLAN
------------------------------------
 Hash Join
   Hash Cond: (t2.t1_id = t1.id)
   ->  Gather
         Workers Planned: 8
         ->  Parallel Seq Scan on t2
   ->  Hash
         ->  Seq Scan on t1
(7 rows)
```

Пример (параллельное соединение таблиц):

```
alex=# explain (costs off) select /*+parallel(8)*/ * from t1 join t2 on t1.id=t2.t1_id;
INFO:  Parallel(parallel_workers = 8.000000) - used
                QUERY PLAN
-------------------------------------------
 Gather
   Workers Planned: 8
   ->  Parallel Hash Join
         Hash Cond: (t1.id = t2.t1_id)
         ->  Parallel Seq Scan on t1
         ->  Parallel Hash
               ->  Parallel Seq Scan on t2
(7 rows)
```

Хинт может быть использован внутри подзапроса для параллельного выполнения только этого подзапроса:

```
alex=# explain select * from (select /*+parallel(3)*/ t2.* from t1 join t2 on t1.id=t2.t1_id) ttt join t3 on
ttt.id=t3.t2_id;
INFO:  Parallel(parallel_workers = 3.000000) - used
                                      QUERY PLAN
-----------------------------------------------------------------------------------------
 Hash Join  (cost=6045.50..21466.03 rows=100000 width=34)
   Hash Cond: (t2.id = t3.t2_id)
   ->  Gather  (cost=2573.50..13861.03 rows=100000 width=17)
         Workers Planned: 3
         ->  Parallel Hash Join  (cost=1573.50..2861.03 rows=32258 width=17)
               Hash Cond: (t1.id = t2.t1_id)
               ->  Parallel Seq Scan on t1  (cost=0.00..957.67 rows=41667 width=4)
               ->  Parallel Hash  (cost=1052.67..1052.67 rows=41667 width=17)
                     ->  Parallel Seq Scan on t2  (cost=0.00..1052.67 rows=41667 width=17)
   ->  Hash  (cost=1636.00..1636.00 rows=100000 width=17)
         ->  Seq Scan on t3  (cost=0.00..1636.00 rows=100000 width=17)
(11 rows)
```

При распараллеливании запроса может быть получен выигрыш в скорости выполнения: Например (выполнение сканирования без распараллеливания):

```
alex=# explain analyze select * from t1 where name similar to 'alex123456789%';
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on t1  (cost=0.00..1791.00 rows=10 width=13) (actual time=92.709..92.710 rows=0 loops=1)
   Filter: (name ~ '^(?:alex123456789.*)$'::text)
   Rows Removed by Filter: 100000
 Planning Time: 0.174 ms
 Execution Time: 92.729 ms
(5 rows)
```

Распараллеливание сканирования уменьшает время выполнения запроса:

```
alex=# explain analyze select /*+parallel(t1, 4)*/ * from t1 where name similar to 'alex123456789%';
INFO:  Parallel(alias = t1, parallel_workers = 4.000000) - used
                                          QUERY PLAN
```

```
----------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..2062.83 rows=10 width=13) (actual time=32.960..33.191 rows=0 loops=1)
   Workers Planned: 4
   Workers Launched: 4
   ->  Parallel Seq Scan on t1  (cost=0.00..1061.83 rows=4 width=13) (actual time=26.738..26.738 rows=0 loops=5)
         Filter: (name ~ '^(?:alex123456789.*)$'::text)
         Rows Removed by Filter: 20000
 Planning Time: 0.221 ms
 Execution Time: 33.215 ms
(8 rows)
```

Интервальное секционирование

Поддержка интервального секционирования

Интервальное секционирование позволяет автоматически расширять секционированную таблицу новым разделом заданного диапазона значений при вставке в эту таблицу строки, для которой значение секционированного ключа выходит за рамки диапазонов существующих разделов.

В качестве примера рассмотрим секционированную таблицу t1, определенную следующим образом:

```
CREATE TABLE t1 (a INT) PARTITION BY RANGE (a) INTERVAL (500)
```

для которой определен единственный раздел t1_p0

```
CREATE TABLE t1_p0 PARTITION OF t1 FOR VALUES FROM (100) TO (200);
```

В такой конфигурации вставка строк в таблицу t1, для которых значение колонки 'a' находится в диапазоне [100, 200), приводит к добавлению строк в таблицу t1_p0. Попытка вставить строку со значением колонки 'a' из диапазонов: [200, 700), [700, 1200) и так далее приводит к созданию новой секции таблицы и вставке в нее строки, при условии, что такой секции еще не существует на момент вставки.

Имена таблиц автоматически создаваемых разделов формируются по следующей схеме:

```
ИмяБазовойТаблицы_{хэш сумма от значения нижней границы раздела}
```

Пример:

```
t1_e62684a2 — для автоматически созданной секции по ключу с типом INT
в диапазоне значений: [70, 80)
```

```
t2_d4db3cdd — для автоматически созданной секции по ключу
с типом DATE в диапазоне значений: [10 мая 2025, 20 мая 2025)
```

Для имен таблиц автоматически создаваемых разделов справедливо ограничение длины имени, устанавливаемое константой `NAMEDATALEN`. Эта константа задается при сборке сервера `PostgreSQL` и по умолчанию равна 64 байтам. Если имя таблицы автоматически создаваемого раздела превышает это ограничение, то оператор `INSERT` завершится с ошибкой. В сообщении об ошибке будет указано, что имя нового раздела превышает максимально

заданную длину.

Механизм интервального секционирования поддерживается с учетом следующих ограничений:

- интервальное секционирование (`interval partitioning`) поддерживается только для метода секционирования по диапазону значений (`range partitioning`);
- интервальное секционирование поддерживается только для несоставного ключа;
- для успешной работы механизма интервального секционирования необходимо, чтобы явно был определен хотя бы один раздел секционированной таблицы;
- для выполнения операции INSERT или UPDATE, приводящей к созданию нового раздела пользователем, который не является владельцем секционированной таблицы, необходимо, чтобы этот пользователь имел права на (`SELECT`, `INSERT`,`UPDATE`);
- автоматически созданные новые разделы секционированной таблицы принадлежат владельцу секционированной таблицы;
- тип ключа, для которого определяется интервальное секционирование, может быть целочисленным (`SMALLINT`, `INT`, `BIGINT`, `NUMERIC`) либо временным (`DATE`,`TIMESTAMP`);
- определение интервала, задаваемого синтаксической конструкцией `INTERVAL`(...), должно описываться константой или константным выражением;
- значения `NULL`, `DEFAULT`, `MINVALUE`, `MAXVALUE` и выражения с их участием не могут быть указаны в определении интервала;
- новые разделы для таблицы с интервальным секционированием создаются строго в порядке возрастания значений ключа секционирования; автоматическое создание разделов в порядке убывания значений ключа не поддерживается;
- при создании автоматической секции верхняя граница секции может быть сдвинута так, чтобы избежать перекрытия с существующей секцией. При этом размер вновь созданной автоматической секции не будет превышать заданный интервал;
- если из нескольких сессий выполняются операторы `INSERT`, которые должны привести к созданию нового раздела таблицы, то первая захватившая блокировку сессия создаст требуемый раздел, а остальные сессии будут ожидать завершения создания нового раздела и затем произведут вставку.

```
ERROR: duplicate key value violates unique constraint "pg_type_typename_nsp_index"
```

Расширение синтаксиса операторов CREATE TABLE/ALTER TABLE для поддержки интервального секционирования

В синтаксис оператора `CREATE TABLE` добавляется следующая конструкция:

```
CREATE TABLE table_name ( ... )
```

```
PARTITION BY RANGE (имя столбца)
INTERVAL (expr_or_const) ...;
```

где, expr_or_const представляет собой целочисленное положительное значение или период времени, либо константное выражение, возвращающее положительное значение или период времени. Период времени может быть указан для интервала типа DATE и TIMESTAMP. Для интервального ключа типа DATE значение интервала может иметь точность год/месяц/день (YEAR/MONTH/DAY); для ключа типа TIMESTAMP значение интервала может иметь точность час/минута или полный диапазон (HOUR/MINUTE/FULL_RANGE)

Пример 1

Создать таблицу t1, состоящую из столбца 'a', по которому выполняется секционирование по диапазону значений; в таблице определить секцию 't1_p1', содержащую строки, для которых значение столбца 'a' больше или равно 1 и меньше 1000; для добавляемых значений больше 1000 автоматически создавать новые разделы с диапазоном, охватывающим следующие 100 значений:

```
CREATE TABLE t1 (a INT) PARTITION BY RANGE (a) INTERVAL(100);
CREATE TABLE t1_p1 PARTITION OF t1 FOR VALUES FROM (1) TO (1000);
```

Пример 2

Аналогично примеру 1, создать таблицу t1 с разбиением данных по разделам на основании даты. Строки с значениями столбца 'a' за сентябрь 2023 года добавляются в раздел 'p1', все остальные данные распределяются по разделам шириной в один месяц:

```
CREATE TABLE t1 (a DATE) PARTITION BY RANGE (a) INTERVAL ('1' MONTH);
CREATE TABLE t1_p1 PARTITION OF t1 FOR VALUES FROM ('01-09-2023') TO ('30-09-2023');
```

В синтаксис оператора ALTER TABLE добавляется следующая конструкция:

```
ALTER TABLE table_name ... SET INTERVAL (expr_or_const) ...;
```

Пример 3

Для таблицы t1 установить интервальное секционирование с разбиением интервалов шириной 1000 значений:

```
ALTER TABLE t1 SET INTERVAL (1000);
```

Пример 4

Для таблицы t1 установить интервальное секционирование с разбиением интервалов шириной 1 месяц:

```
ALTER TABLE t1 SET INTERVAL ('1' MONTH);
```

Расширение таблиц системного каталога

Для сохранения метаданных, описывающих интервал секционированной таблицы, была расширена таблица системного каталога `pg_partitioned_table` (добавлена колонка `partintervalexpr`, имеющая тип `pg_node_tree`).

Выключение механизма миграции строк между разделами секционированной таблицы

Выполнение SQL-оператора UPDATE для секционированной таблицы может привести к перемещению строки из одной секции в другую. Это может произойти в случае модификации значения колонки (колонок) таблицы, по которой задан ключ секционирования.

Например, таблица t1 может быть определена, как секционированная таблица, в которой записи, имеющие значение 100, помещаются в секцию t1_p1; записи, имеющие значение 200, помещаются в секцию t1_p2. Ниже приведено определение таблицы t1 с секциями t1_p1 и t1_p2:

```
CREATE TABLE t1 (a INT, b VARCHAR(10)) PARTITION BY LIST (b);
CREATE TABLE t1_p1 PARTITION OF t1 FOR VALUES IN (100);
CREATE TABLE t1_p2 PARTITION OF t1 FOR VALUES IN (200);
INSERT INTO t1 VALUES (100, 'str1');
INSERT INTO t1 VALUES (200, 'str2');
```

Для такой таблицы выполнение следующего оператора:

```
UPDATE t1 SET val = 200 WHERE b = 'str1';
```

приведет к перемещению строки таблицы t1, имеющей значение 'str1' в колонке b, из секции t1_p1 в секцию t1_p2.

По умолчанию реализация декларативного секционирования PostgreSQL позволяет выполнять перемещение строки из одной секции в другую в процессе выполнения оператора UPDATE. Для выключения механизма перемещения строк между разделами секционированной таблицы в процессе выполнения оператора UPDATE был расширен синтаксис DDL операторов:

```
CREATE TABLE/ALTER TABLE
```

синтаксическими конструкциями:

```
ENABLE ROW MOVEMENT/DISABLE ROW MOVEMENT
```

Например, для таблицы t1 из предыдущего примера можно запретить перенос строк между разделами секционированной таблицы, выполнив следующий оператор:

```
ALTER TABLE t1 DISABLE ROW MOVEMENT;
```

Синтаксические конструкции:

```
ENABLE ROW MOVEMENT/DISABLE ROW MOVEMENT
```

могут быть применены только ко всей секционированной таблице. Попытка применения этих конструкций к отдельным разделам будет приводить к ошибке.

Например:

```
CREATE TABLE t1 (a INT) PARTITION BY RANGE (a);
CREATE TABLE t1_p1 PARTITION OF t1 FOR VALUES FROM (1) TO (100);
ALTER TABLE t1_p1 DISABLE ROW MOVEMENT;
```

```
ERROR: ROW MOVEMENT feature can be applied only for partitioned tables
```

Если для секционированной таблицы отключен механизм миграции строк между разделами, и выполнение оператора `UPDATE` над секционированной таблицей должно привести к удалению записи из одной секции таблицы и добавление в другую секцию, то выполнение такого оператора `UPDATE` завершится с ошибкой.

Например, для таблицы t1, определенной следующим образом:

```
CREATE TABLE t1 (a INT) PARTITION BY RANGE (a) DISABLE ROW MOVEMENT;
CREATE TABLE t1_p1 PARTITION OF t1 FOR VALUES FROM (1) TO (10);
CREATE TABLE t1_p2 PARTITION OF t1 FOR VALUES FROM (10) TO (20);
INSERT INTO t1 VALUES (5);
```

выполнение оператора из примера ниже:

```
UPDATE t1 SET a = 15 WHERE a = 5;
```

приведет к ошибке:

```
ERROR:  could not move the row into the another partition
```

Для хранения информации о включении/отключении механизма переноса строк секционированной таблицы добавлена новая опция в колонку reloptions таблицы системного каталога **pg_class**. Название новой опции: `disable_row_movement`. Опция `disable_row_movement` отображается в выводе команды '\d+' для секционированной таблицы только в том случае, если значение опции было установлено явно.

Как следствие, секционированная таблица, созданная с явным разрешением переноса строк:

```
CREATE TABLE PARTITION BY ... ENABLE ROW MOVEMENT
```

и таблица, созданная оператором:

```
CREATE TABLE PARTITION BY ...
```

без указания конструкции:

```
ENABLE ROW MOVEMENT
```

с точки зрения семантики обработки переноса строк между разделами, ведут себя одинаково, но различаются, с точки зрения вывода команды определения таблицы **\d+**.

Оптимизация запросов с оператором ИЛИ

Общая информация

Оптимизация запросов, содержащих оператор ИЛИ (OR) в условии WHERE, осуществляется путём преобразования таких запросов в запросы с использованием UNION ALL.

Запрос вида:

```
SELECT ... FROM table1, table2 WHERE ... AND (условие1 OR условие2)
```

в котором `условие1` — логическое выражение, ссылающееся только на `table1`, а `условие2` — только на `table2`, преобразуется в запрос вида:

```
SELECT ... FROM table1 WHERE ... AND условие1
UNION ALL
SELECT ... FROM table2 WHERE ... AND условие2 AND (условие1 IS NOT TRUE)
```

Преобразованный запрос может, в некоторых случаях, выполняться быстрее за счёт устранения соединений таблиц.

Оптимизация также может применяться к запросам с оператором IN, таким как:

```
SELECT ... FROM t1, t2 WHERE ... 'string' IN (t1.column1, t2.column2) ...
```

так как подобные запросы имеют внутреннее представление, эквивалентное:

```
SELECT .... FROM t1, t2 WHERE ... 'string' = t1.column1 OR 'string' = t2.column2
```

Выбор между модифицированным и исходным запросом происходит на основе оценки стоимости плана. Если план модифицированного запроса имеет большую стоимость, то преобразование не происходит.

Ограничения

Не все запросы могут быть преобразованы таким образом. Каждое подусловие оператора ИЛИ должно ссылаться ровно на одну таблицу. Функции, используемые в условиях, не должны иметь тип изменчивости VOLATILE. Также запрос не должен содержать CTE и подзапросы.

Конфигурация

По умолчанию, данное преобразование отключено, для его включения необходимо установить переменную конфигурации `optimize_union_or` в значение `on`.

Для этого необходимо добавить в конфигурационный файл postgresql.conf: `optimize_union_or = on` либо для текущей сессии выполнить команду:

```
SET optimize_union_or = on;
```

Пример работы оптимизации

Пусть есть две таблицы следующего вида: first и second, для которых по текстовым колонкам построены индексы:

```
id  |first_varchar
----+-------------------------------------------------------
   1|285f673c35d55e9ee898e972538aeb648db4b46a5b1fc4374024b...
   2|e8f96aa9c18edc68ecb7ee2ddfa5a20ca4642c06de657f2f367db...
   3|bbbefa46e0d077543fe1a3e06a171334a60d4646bb09012c7492c...
   4|c51090cc16fb6cc5b69b0483865ef32a438e6bef4fd83d7e8728d...
   5|d99741b6e957c111dcb04e971d8758d1f2a52a536fae5009257b3...
   ...

id  |second_varchar
----+-------------------------------------------------------
   1|836ce8497ed551f7df05552d5ec53b855da1031037bd8965e474f...
   2|5f86b716da0832bd0ed6e5f8d926e3b68508dd55cfc0b8d391641...
   3|addaef57e160415fa20725b2a899a9bcdc073075652e879197bcf...
   4|cbbf2ad29079f6e457965b00cde6597e146f12d115ca6c9aa74f2...
   5|be29d2d093319defbf4fad3d2e2b6073c56def71bd817aff382fd...
   ...
```

Запрос:

```
SELECT * FROM first, second WHERE first.id = second.id AND (first_varchar LIKE '001%' OR second_varchar LIKE '0b2%');
```

с выключенной оптимизацией имеет план выполнения:

```
 Merge Join  (cost=0.58..5660.48 rows=10 width=272) (actual time=0.552..26.738 rows=31 loops=1)
   Merge Cond: (first.id = second.id)
   Join Filter: (((first.first_varchar)::text ~~ '001%'::text) OR ((second.second_varchar)::text ~~ '0b2%'::text))
   Rows Removed by Join Filter: 49969
   ->  Index Scan using fpk on first  (cost=0.29..2330.29 rows=50000 width=136) (actual time=0.006..8.602 rows=50000
loops=1)
   ->  Index Scan using spk on second  (cost=0.29..2330.29 rows=50000 width=136) (actual time=0.005..8.709 rows=50000
loops=1)
 Planning Time: 0.203 ms
```

```
 Execution Time: 26.773 ms
```

после включения оптимизации план меняется на следующий:

```
 Append  (cost=0.70..100.01 rows=10 width=272) (actual time=0.017..0.123 rows=31 loops=1)
   ->  Nested Loop  (cost=0.70..49.98 rows=5 width=272) (actual time=0.016..0.046 rows=10 loops=1)
         ->  Index Scan using index1 on first  (cost=0.41..8.44 rows=5 width=136) (actual time=0.010..0.019 rows=10
loops=1)
               Index Cond: (((first_varchar)::text ~>=~ '001'::text) AND ((first_varchar)::text ~<~ '002'::text))
               Filter: ((first_varchar)::text ~~ '001%'::text)
         ->  Index Scan using spk on second  (cost=0.29..8.31 rows=1 width=136) (actual time=0.002..0.002 rows=1
loops=10)
               Index Cond: (id = first.id)
   ->  Nested Loop  (cost=0.70..49.99 rows=5 width=272) (actual time=0.010..0.074 rows=21 loops=1)
         ->  Index Scan using index2 on second  (cost=0.41..8.44 rows=5 width=136) (actual time=0.005..0.023 rows=22
loops=1)
               Index Cond: (((second_varchar)::text ~>=~ '0b2'::text) AND ((second_varchar)::text ~<~ '0b3'::text))
               Filter: ((second_varchar)::text ~~ '0b2%'::text)
         ->  Index Scan using fpk on first  (cost=0.29..8.31 rows=1 width=136) (actual time=0.002..0.002 rows=1 loops=22)
               Index Cond: (id = second.id)
               Filter: (((first_varchar)::text ~~ '001%'::text) IS NOT TRUE)
               Rows Removed by Filter: 0
 Planning Time: 0.362 ms
 Execution Time: 0.152 ms
```

Видно, что время исполнения запроса значительно уменьшается.

Оптимизация кэша субтранзакций

Механизм многоверсионного управления конкурентным доступом (MVCC) СУБД FTData при доступе к кортежам, обновляемым в различных одновременно выполняемых субтранзакциях, выполняет проверку видимости кортежа с использованием кэша субтранзакций.

Кэш используется для хранения идентификаторов родительской транзакции для каждой субтранзакции. При этом родительской транзакцией также может быть субтранзакция.

Информация обо всех активных субтранзакциях хранится в файлах, расположенных в директории `pg_subtrans` и в структурах PGPROC активных процессов. Для ускорения работы с этими данными они кэшируются; при этом сервер СУБД может использовать два разных кэша: кэш субтранзакций в PGPROC или кэш для файлов из директории `pg_subtrans`.

Кэш субтранзакций в PGPROC

> `PGPROC` - это структура, хранящая текущее состояние серверного процесса. Структуры `PGPROC` вех процессов объединены в пул, выделяемый в разделяемой памяти при запуске сервера СУБД. При установлении соединения с клиентом СУБД из пула выделяет незанятую структуру для вновь созданного процесса. Размер пула регулируется параметром max_connections.

Кэш субтранзакций в PGPROC хранит список вложенных субтранзакций, выполняемых процессом. Любой процесс имеет доступ к таким кэшам всех активных процессов. По информации из кэша можно определить родительскую транзакцию для любой активной субтранзакции (пока кэш не переполнен ни в одном из процессов).

Если количество вложенных субтранзакций превышает 64 хотя бы для одного процесса, то этот кэш не может быть использован для поиска родительской транзакции (в нём находятся не все записи об активных субтранзакциях сервера).

Структура PGPROC оптимизирована для быстрого доступа к данным из других процессов, поэтому, в первую очередь, проверяется возможность поиска в этом кэше, и только если кэш переполнен, выполняется поиск в кэшированных данных из `pg_subtrans`.

Статусы субтранзакций в pg_subtrans

`pg_subtrans` содержит данные обо всех субтранзакциях на сервере. Неактуальные данные для субтранзакций, которые вышли за области видимости активных транзакций, удаляются при создании checkpoint.

SLRU-кэш субтранзакций в pg_subtrans

Содержимое файлов из директории `pg_subtrans` кэшируется SLRU-кэшем субтранзакций.

Кэш хранит наиболее часто используемую часть данных из файлов `pg_subtrans` в разделяемой памяти сервера (данные хранятся в виде копий страниц файлов).

При запросе данных из кэша может возникать ситуация, когда данные представлены только на диске, а кэш в разделяемой памяти заполнен. В этой ситуации из разделяемой памяти будет вытеснена страница, последнее обращение к которой выполнялось раньше всего, и на её место будет помещена копия требуемой страницы. Такой алгоритм вытеснения данных из кэша называется "упрощённое вытеснение давно неиспользуемых" (англ. "simple least recently used" или "SLRU").

Для синхронизации доступа к SLRU-кэшу субтранзакций используется легковесная блокировка (`LWLock`) `SubtransSLRU`, а для разделения доступа к файлам из `pg_subtrans` - `SubtransBuffer`.

Масштабирование SLRU-кэша субтранзакций

Размер разделяемой памяти, выделяемой для SLRU-кэша субтранзакций СУБД FTData можно настроить параметром конфигурации `subtransaction_buffers` в `postgresql.conf` или SQL командами. Для применения изменений требуется перезагрузка сервера.

Выделяемая для SLRU-кэша субтранзакций разделяемая память ограничивается также параметром `shared_buffers`. Следует учитывать, что, если установить `shared_buffers` так, что у сервера не хватит разделяемой памяти для размещения кэша, сервер не запустится.

Локальный кэш субтранзакций

Для ускорения доступа к информации, полученной через SLRU-кэш субтранзакций из файлов из каталога pg_subtrans, серверные процессы СУБД FTData кэшируют результаты обращений к SLRU в памяти процесса. Доступ к локальному кэшу осуществляется без глобальных блокировок; это существенно ускоряет работу СУБД, обрабатывающей продолжительные транзакции с большим количеством вложенных субтранзакций.

Размер локального кэша СУБД FTData можно настроить параметром конфигурации `local_cache_subtrans_entries` в `postgresql.conf` или SQL-командами. Для применения изменений требуется перезагрузка сервера.

`local_cache_subtrans_entries` - задаёт количество последних результатов поиска в SLRU-кэше субтранзакций, хранимых в локальном кэше.

- Минимальное значение: 0 - кэш не работает.
- Максимальное значение: 2 147 483 647 (2^{32}-1).
- Значение по умолчанию: 200 000.

Одна запись кэша занимает 16 байт оперативной памяти серверного процесса. То есть, для кэша размером в 200 000 записей на сервере со 100 активными процессами будет зарезервировано
16 байт в записи * 200 000 записей в кэше * 100 экземпляров кэша = 320 МБ оперативной памяти.

Точно подобрать размер локального кэша, исходя из нагрузок приложения, можно, используя представление `ftd_stat_subxids` (см. раздел Мониторинг использования кэша субтранзакций).

Сжатие данных таблиц и индексов

ZMD — подсистема хранения данных (storage manager) таблиц, TOAST-таблиц и индексов на диске в сжатом виде. Подсистема включена в состав СУБД FTData, начиная с версии 17.2.1. ZMD сжимает страницы данных перед сохранением на диск, что, в зависимости от структуры таблицы или индекса, позволяет сэкономить до 80% места на диске.

> Подсистема хранения данных md в СУБД FTData (и PostgreSQL) хранит страницы данных в файлах как дамп этой страницы из памяти (в несжатом виде). Все страницы данных имеют одинаковый размер (8 кБ), даже в случае, если страница не заполнена полностью или пустая. Подробнее об устройстве страницы можно прочитать в документации PostgreSQL.

Управление сжатием таблиц и индексов

Чтобы создать сжатую таблицу или индекс, нужно указать параметр хранения `smgr` в выражении `WITH`. Доступны следующие значения:

- `lz4,{acceleration}` — сжатие кодеком lz4. `{acceleration}` — параметр степени сжатия, число от 0 (максимальное сжатие, минимальная скорость) до 65537 (минимальное сжатие, максимальная скорость);
- `lz4` — сжатие кодеком lz4. Аналогично `lz4,0` (максимальное сжатие, минимальная скорость);
- `pglz` - сжатие кодеком pglz;
- `no` — сжатие не выполняется (но доступ осуществляется через ZMD). Значение может быть использовано для тестирования производительности.

> В случае секционированных таблиц сжатие может быть использовано только для каждой секции по отдельности, но не для все таблицы целиком.

Создание сжатой таблицы

```
-- Максимальная степень сжатия lz4
CREATE TABLE t1(id INT, name VARCHAR) WITH (smgr=lz4);
-- Средняя степень сжатия lz4
CREATE TABLE t2(id INT, name VARCHAR) WITH (smgr="lz4,32769");
-- Минимальная степень сжатия lz4
CREATE TABLE t2(id INT, name VARCHAR) WITH (smgr="lz4,65537");
-- Сжатие pglz
CREATE TABLE t3(id INT, name VARCHAR) WITH (smgr=pglz);
```

Создание сжатого индекса

```
-- Сжатие lz4
CREATE INDEX on t1(id) WITH (smgr=lz4);
-- Сжатие pglz
CREATE INDEX on t3(id) WITH (smgr=pglz);
```

Создание сжатого индекса для ограничения (constraint)

```
CREATE TABLE t4(
    id INT PRIMARY KEY WITH (smgr=lz4), -- Сжатый индекс
    name VARCHAR,
    UNIQUE(name) WITH (smgr="lz4,3") -- Сжатый индекс
) WITH (smgr=lz4); -- Параметр сжатия данных таблицы
```

Сжатие существующей таблицы или индекса

```
-- Преобразование таблицы к сжатому виду
ALTER TABLE t1 SET (smgr=lz4);
-- Преобразование индекса к сжатому виду
ALTER INDEX t1_id_idx SET (smgr=lz4);
```

Выполнение операции может занять значительное время в случае большого размера таблицы или индекса. Операция получает эксклюзивный доступ к таблице или индексу на время выполнения. Сжатие страниц таблицы происходит "как есть", т.е. содержимое страниц не изменится при последующей распаковке, что даёт возможность использовать, например, те же индексы, что были построены для первоначальной таблицы.

Сжатие существующей TOAST-таблицы

```
-- Преобразование TOAST-таблицы к сжатому виду
SET allow_system_table_mods=on;
-- Найти TOAST таблицу t1 можно так:
-- SELECT oid, reltoastrelid::regclass FROM pg_class WHERE relname='t1';
ALTER TABLE pg_toast.pg_toast_16385 SET (smgr=lz4);
SET allow_system_table_mods=off;
```

Чтобы изменить параметры TOAST-таблицы, нужно разрешить изменение системных таблиц (см. `allow_system_table_mods`).

Можно изменить параметр хранения для индекса TOAST-таблиц.

```
-- Преобразование индекса TOAST-таблицы к сжатому виду
SET allow_system_table_mods=on;
ALTER INDEX pg_toast_16385_id_idx SET (smgr=lz4);
SET allow_system_table_mods=off;
```

Допустимо изменение алгоритма сжатия уже сжатой таблицы

```
CREATE TABLE t1(id INT, name VARCHAR) WITH (smgr=lz4);
-- Работа с таблицей t1;
ALTER TABLE t1 SET (smgr=pglz);
```

Возврат таблицы или индекса к несжатому состоянию

Чтобы вернуть таблицу под управление стандартной системе хранения md, нужно сбросить параметр `smgr`:

```
-- Преобразование таблицы к несжатому виду
ALTER TABLE t1 RESET (smgr);
-- Преобразование индекса к несжатому виду
ALTER INDEX t1_id_idx RESET (smgr);
-- Преобразование TOAST-таблицы к несжатому виду
SET allow_system_table_mods=on;
ALTER TABLE pg_toast.pg_toast_16385 RESET (smgr);
SET allow_system_table_mods=off;
```

Поиск свободного места для записи сжатой страницы данных

В СУБД FTData (и PostgreSQL) представления (таблицы и индексы) хранятся на диске в виде набора файлов. Этот набор файлов логически разделён на слои (fork) и сегменты (segments):

- **Сегмент (segment)** — файл с данными представления. Размер сегмента не превышает 1 Гб. Если требуется сохранить более 1 Гб данных, эти данные будут разбиты на сегменты размером не более 1 Гб.
- **Слой (fork)** — один или несколько сегментов данных, выполняющих определённую функцию:
  - **основной слой (main fork)** хранит страницы данных таблицы,
  - **слой метаданных ZMD** хранит метаинформацию о расположении сжатых страниц данных в сегментах основного слоя.

При вставке или обновлении данных изменяется содержимое страницы данных, а значит, её размер в сжатом виде. Это приводит к тому, что не во всех случаях удаётся поместить обновлённую сжатую страницу на то же место в основном слое. Поэтому требуется найти

свободное место достаточного размера для помещения новой версии сжатой страницы. Для поиска свободного места в основном слое реализовано три стратегии, которые выбираются параметром конфигурации zmd_search_free_space:

- `none` — ZMD не будет искать свободное место, а будет всегда записывать данные в конец слоя данных. Это наиболее быстрый способ записи, однако при частом сбросе страниц на диск файл слоя данных может быстро расти. Единственный способ избавиться от фрагментации файла данных в таком режиме — выполнение команды VACUUM FULL для таблицы.
- `current_buffer` — ZMD будет искать свободное место в файле данных, просматривая только ту страницу слоя метаданных, в которой расположены метаданные этой страницы. Если свободное место не найдено, страница будет записана в конец файла данных. В большинстве случаев это наиболее оптимальный выбор.
- `recent_buffers` — ZMD будет искать свободное место во всех страницах метаданных, загруженных в оперативную память. Если свободное место не найдено, страница будет записана в конец файла данных. Это наиболее медленный способ, его не рекомендуется включать при выполнении команд преобразования несжатых таблиц и индексов в сжатую форму.

Восстановление после сбоя

В случае, если какой-либо сбой произойдёт во время работы со сжатыми таблицами, существует вероятность того, что данные в основном файле (слое) данных и данные в слое метаданных окажутся рассинхронизированы. Например, сжатая страница оказалась записана в файл данных, а буфер с метаданными, в котором был обновлён указатель на страницу, — нет. В этом случае у системы восстановления СУБД FTData не будет возможности считать даже устаревший вариант страницы из файла данных, чтобы применить на нём изменения, записанные в WAL-журнале.

> Чтобы данные были успешно восстановлены, необходимо воспользоваться параметром full_page_writes = on (включён по умолчанию), при включении которого СУБД FTData, после каждого первого изменения страницы после выполнения CHECKPOINT, будет сохранять в WAL-журнал изменяемую страницу целиком.

В процессе восстановления поиск свободного места будет осуществляться только среди вновь записанных (в процессе восстановления) страниц. После завершения процесса восстановления по WAL-журналам слой метаданных ZMD будет исправлен. Изменения в слое метаданных ZMD в WAL-журнал не сохраняются.

Расширение zmd_inspect

Расширение предоставляет средства, предназначенные для мониторинга состояния сжатых таблиц и индексов.

> Для работы ZMD установка расширения не требуется.

Расширение определяет тип данных `zmd_item`, который содержит в себе информацию о расположении страницы, которую он описывает. Этот тип данных может быть представлен в текстовом формате, его также принимают функции, анализирующие состояние страницы.

Функция	Описание

`ZMD_BLOCK_MAP(name TEXT)`
`→ TABLE(blockno INT,`
`offset_number INT2,`
`item ZMD_ITEM)`
	Функция возвращает представление слоя метаданных ZMD.

`ZMD_ITEM_CHECK_INDEX(item ZMD_ITEM)` `→ CSTRING`	Функция проверяет валидность индексации сжатого блока. В случае обнаружения ошибки возвращает ее описание, иначе `NULL`.
`ZMD_ITEM_CHECK_CONTENT(item ZMD_ITEM)` `→ CSTRING`	Функция проверяет содержимое сжатого блока. В случае обнаружения ошибки возвращает ее описание, иначе - `NULL`.
`ZMD_ITEM_MAINBLOCK(item ZMD_ITEM)` `→ INT`	Функция возвращает основной номер блока в слое данных, соответствующий `item`.
`ZMD_ITEM_BEGIN(item ZMD_ITEM)` `→ BIGINT`	Функция возвращает смещение начала сжатого блока в слое данных.
`ZMD_ITEM_END(item ZMD_ITEM)` `→ BIGINT`	Функция возвращает смещение конца сжатого блока в слое данных.
`ZMD_ITEM_SIZE(item ZMD_ITEM)` `→ BIGINT`	Функция возвращает длину сжатого блока в слое данных.
`ZMD_CHECK(relname TEXT)` `→ VOID`	Функция выполняет проверку целостности сжатой таблицы.

Перед вызовом функций `ZMD_ITEM_CHECK_INDEX` и `ZMD_ITEM_CHECK_CONTENT` необходимо получить блокировку на уровне таблицы не ниже `ShareLock`

```
BEGIN;
LOCK TABLE zmd IN SHARE MODE;
SELECT *, zmd_item_check_index(item) FROM zmd_block_map('zmd');
SELECT *, zmd_item_check_content(item) FROM zmd_block_map('zmd');
END;
```

Утилита ftdbackrest

Общие сведения

`ftdbackrest` - утилита управления резервным копированием и восстановлением кластеров баз данных СУБД FTData. `ftdbackrest` разработан на основе `pgBackRest`. В данном разделе представлен перевод документации `pgBackRest` с дополненным описанием функционала форка `ftdbackrest`. Далее по тексту, если присутствует описание применительно `pgBackRest`, это в равной степени относится и к функционалу `ftdbackrest`.

Цель `pgBackRest` - быть надежным, простым в использовании решением для создания и восстановления резервных копий, которое может плавно масштабироваться до самых больших баз данных и нагрузок путем использования алгоритмов, оптимизированных под требования баз данных.

Основные понятия

В данном разделе определены понятия, которые применимы к pgBackRest и к FTData.

Резервное копирование

Резервная копия (backup) — это консистентная копия кластера базы данных, которую можно использовать для восстановления базы данных после сбоя оборудования, для восстановления на момент времени (Point-in-time recovery, PITR) или в качестве резервной базы данных (standby).

Полное резервное копирование (full backup) - pgBackRest копирует все содержимое кластера базы данных в резервную копию. Первая резервная копия кластера базы данных всегда является полной резервной копией. pgBackRest всегда может напрямую восстановить полную резервную копию. Полная резервная копия не зависит от каких-либо файлов за пределами полной резервной копии.

Дифференциальное резервное копирование (differential backup): pgBackRest копирует только те файлы кластера базы данных, которые изменились с момента последнего полного резервного копирования. pgBackRest восстанавливает данные, копируя все файлы из выбранной дифференциальной резервной копии и соответствующие неизмененные файлы из предыдущей полной резервной копии. Преимущество дифференциальной резервной копии заключается в том, что для ее хранения требуется меньше места на диске, чем для хранения полной резервной копии, однако и дифференциальная резервная копия, и полная резервная копия должны быть валидными для восстановления дифференциальной резервной копии.

Инкрементное резервное копирование (incremental backup): pgBackRest копирует только те файлы кластера базы данных, которые были изменены с момента последнего резервного копирования (инкрементного, дифференциального или полного). Поскольку инкрементное резервное копирование включает только те файлы, которые были изменены с момента

предыдущего резервного копирования, они, как правило, намного меньше, чем полные или дифференциальные резервные копии. Как и в случае с дифференциальным резервным копированием, для восстановления из инкрементной резервной копии необходимо, чтобы все другие резервные копии (начиная с последнего полного копирования) также были валидными.

Блочное инкрементное резервное копирование (incremental block backup): pgBackRest копирует только те части файла, которые были изменены с момента последнего резервного копирования, а не весь файл целиком. Для включения блочного инкрементного резервного копирования необходимо включить опцию `repo-block` в конфигурационном файле `pgbackrest.conf`, рекомендуется включить эту опцию для всех типов резервного копирования, так как в этом случае блочное инкрементное резервное копирование будет работать лучше всего. Также необходимо включить объединение файлов. Как и в случае с дифференциальным и инкрементным резервным копированием, для восстановления из блочной инкрементной резервной копии необходимо, чтобы все другие резервные копии (начиная с последнего полного копирования) также были валидными.

Блочное инкрементное резервное копирование (block backup): pgBackRest копирует только те блоки (страницы) файлов данных, которые были изменены с момента последнего резервного копирования (инкрементного, дифференциального или полного). Блок считается измененным, если его LSN больше, чем начальный LSN предыдущего бэкапа. Поскольку инкрементное резервное копирование на уровне блоков включает только те блоки, которые были изменены с момента предыдущего резервного копирования, они, как правило, намного меньше, чем полные, дифференциальные или инкрементные резервные копии. Как и в случае с дифференциальным и инкрементным резервным копированием, для восстановления из блочной инкрементной резервной копии необходимо, чтобы все другие резервные копии (начиная с последнего полного копирования) также были валидными.

Из-за особенностей алгоритма восстановления блочной инкрементальной резервной копии контрольные суммы (опция `--data_checksums` утилиты `initdb`) должны быть включены, чтобы избежать дальнейшей работы с поврежденной базой, в случае повреждения данных резервной копии.

Восстановление

Восстановление (restore) - процесс копирования резервной копии в систему, где она будет запущена, как работающий кластер базы данных. Для правильной работы процесса восстановления требуются файлы резервных копий и один или несколько WAL-сегментов.

В случаях создания нескольких блочных инкрементных резервных копий и дальнейшего восстановления не из последней таковой, выполнение следующего резервного копирования, при выборе инкрементной блочной резервной копии, будет заменено на выполнение полного резервного копирования в силу особенностей алгоритмов.

Журнал предзаписи

Журнал предзаписи (WAL) — это механизм, который FTData использует для обеспечения сохранности зафиксированных изменений. Транзакции последовательно записываются в WAL и считаются зафиксированными, только когда эти записи сбрасываются на диск, после этого фоновый процесс записывает изменения в основные файлы кластера базы данных (также известные как куча (heap)). Если произошел сбой, то журнал предзаписи воспроизводится повторно для обеспечения согласованности базы данных. Журнал предзаписи разбит на отдельные 16-мегабайтные файлы, называемые сегментами. Сегменты следуют соглашению об именах 0000000100000A1E000000FE, где первые 8 шестнадцатеричных цифр представляют временную шкалу, а следующие 16 цифр — логический порядковый номер (LSN).

Шифрование

Шифрование (encryption) — это процесс преобразования данных в формат, который невозможно распознать, если не будет предоставлен соответствующий пароль. pgBackRest зашифровывает репозиторий на основе пароля, предоставленного пользователем, тем самым предотвращая несанкционированный доступ к данным, хранящимся в репозитории.

Станза

Станза (stanza) - это конфигурация кластера базы данных FTData, в которой определено, где расположен кластер, как будет выполняться его резервное копирование, параметры архивирования и так далее.

Установка

Утилита устанавливается в составе СУБД FTData (см. раздел Установка сервера СУБД FTData). ftdbackrest может быть установлен отдельно для работы в составе выделенного хоста репозитория или хоста репликации.

Установка ftdbackrest отдельно от СУБД FTData

Установка на дистрибутивы Astra Linux

```
$ sudo apt install -y ./ftdata-17-libpq*.deb
$ sudo apt install -y ./ftdata-17-ftdbackrest*.deb
```

Установка на дистрибутивы РЕД ОС

```
$ sudo yum install -y ./ftdata-17-libpq*.rpm
$ sudo yum install -y ./ftdata-17-ftdbackrest*.rpm
```

Установка на дистрибутив Альт

```
$ sudo apt-get install -y ./ftdata-17-libpq*.rpm
$ sudo apt-get install -y ./ftdata-17-ftdbackrest*.rpm
```

Начало работы

В данном разделе описана базовая настройка pgBackRest и FTData, а также представлены команды для резервного копирования и восстановления.

Установка

Требуется установить все обязательные пакеты дистрибутива FTData (см. раздел Установка сервера СУБД FTData).

Создание конфигурационного файла и необходимых директорий:

```
sudo mkdir -p -m 770 /var/log/pgbackrest
sudo chown postgres:postgres /var/log/pgbackrest
sudo mkdir -p /etc/pgbackrest
sudo mkdir -p /etc/pgbackrest/conf.d
sudo touch /etc/pgbackrest/pgbackrest.conf
sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
sudo chown postgres:postgres /etc/pgbackrest/pgbackrest.conf
sudo mkdir -p -m 770 /var/lib/ftdbackrest/17
sudo chown postgres:postgres /var/lib/ftdbackrest/17
```

С помощью команды `/opt/ftdata/17/bin/pgbackrest` можно проверить правильность установки `ftdbackrest`. Если какие-то зависимости были пропущены, появится соответствующее сообщение об ошибке.

Подготовка демонстрационного кластера

Подготовьте демонстрационный кластер БД (далее, кластер). В примерах ниже используется кластер, расположенный в `/var/ftdata/17/data`.

Настройка станзы

Большинство серверов баз данных будут иметь только один кластер базы данных FTData и, следовательно, одну станзу, тогда как серверы резервного копирования будут иметь отдельную станзу для каждого кластера базы данных, для которого требуется резервное копирование. Поскольку название станзы будет использоваться для основного кластера и всех реплик, лучше выбрать название, описывающее фактическую функцию кластера, например app или dw, а не название локального кластера, например main или prod.

Название «demo» точно описывает цель этого кластера, поэтому оно также является подходящим названием для станзы.

pgBackRest нужно указать, где находится базовый каталог данных для кластера, хотя путь может быть запрошен у FTData и напрямую, но в процессе восстановления FTData будет недоступен. Во время резервного копирования значение, предоставленное pgBackRest, будет сравниваться с путем, по которому работает FTData, и они должны совпасть, иначе процесс резервного копирования вернет ошибку. По умолчанию FTData хранит кластеры в /var/ftdata/17/[кластер].

Убедитесь, что `pg1-path` точно совпадает с `data_directory` в postgresql.conf.

При создании конфигурационного файла /etc/pgbackrest/pgbackrest.conf владельцу базы данных должны быть предоставлены права на чтение.

Конфигурационный файл pgBackRest должен соответствовать соглашению Windows INI:

- разделы обозначаются текстом в квадратных скобках;
- в каждом разделе содержатся пары ключ/значение;
- строки, начинающиеся с '#', игнорируются и могут использоваться, как комментарии.

Существует несколько способов загрузки конфигурационных файлов pgBackRest:

- параметры `config` и `config-include-path` заданы по умолчанию: файл конфигурации и *.conf файлы будут загружаться из пути по умолчанию (при условии, что они существуют);
- указан параметр `config`: будет загружен только указанный файл конфигурации, и предполагается, что он существует;
- указан параметр `config-include-path`: будут загружены *.conf файлы из указанного пути, и путь должен существовать. Конфигурационный файл будет загружаться из пути по умолчанию, при условии, что он существует. Если нужно загружать файлы только по указанному пути, то можно также указать параметр `no-config`;
- указаны значения для параметров `config` и `config-include-path`: файл конфигурации и *.conf файлы будут загружены из указанных путей. Предполагается, что файлы существуют;
- указано значение для параметра `config-path`: этот параметр переопределяет путь по умолчанию для конфигурационного файла и/или путь до *.conf файлов, если параметры `config` и/или `config-include-path` явно не указаны.

Файлы объединяются, как если бы они были одним большим файлом; порядок не имеет значения, но есть приоритет, основанный на разделах. Приоритет (от высшего к низшему):

- [станза: *команда*]
- [станза]
- [global: *команда*]
- [global]

`--config, --config-include-path` и `--config-path` - параметры командной строки

pgBackRest также может быть настроен с помощью переменных окружения.

Создание репозитория

В репозитории pgBackRest хранит резервные копии и архивирует сегменты WAL.

Заранее оценить, сколько места понадобится, может быть достаточно сложно. Лучше всего сделать несколько резервных копий, затем записать размер различных типов резервных копий (полные/инкрементные/дифференциальные/блочные) и посчитать количество WAL, сгенерированных за день. Это даст общее представление о том, сколько места потребуется.

Для примера, репозиторий будет храниться на том же хосте, что и сервер FTData. Это самая простая конфигурация, которая полезна в тех случаях, когда для резервного копирования хоста базы данных используется традиционное программное обеспечение резервного копирования.

Необходимо указать путь до репозитория в конфигурационном файле:

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-path=/var/lib/ftdbackrest/17
```

Настройка архивации

Для резервного копирования работающего кластера FTData необходимо включить архивирование WAL. Как минимум, один сегмент WAL будет создан в процессе резервного копирования, даже если в кластер не будет производиться явная запись.

```
archive_command = '/opt/ftdata/17/bin/pgbackrest --stanza=demo archive-push %p'
archive_mode = on
max_wal_senders = 3
wal_level = replica
```

FTData указывает расположение архивируемого сегмента WAL с помощью `%p`. Установка `wal_level=replica` и увеличение `max_wal_senders` позволит добавить реплики позже, без перезапуска основного кластера, даже если в данный момент нет ни одной реплики.

Кластер FTData необходимо перезапустить после внесения этих изменений и перед выполнением резервного копирования.

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data restart
```

Если ожидается, что архивирование сегмента WAL займет более 60 секунд (время, установленное по умолчанию), следует увеличить параметр времени ожидания архивирования `archive-timeout`. Обратите внимание, что этот параметр отличается от параметра `archive_timeout` в FTData, который используется для принудительного переключения сегмента WAL.

Команда `archive-push` имеет собственные опции конфигурации, например, можно установить более низкий уровень сжатия, чтобы ускорить архивирование, не влияя на сжатие, используемое для резервных копий.

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-path=/var/lib/ftdbackrest/17

[global:archive-push]
compress-level=3
```

Этот метод конфигурации можно использовать для любой команды и даже для определенной станзы, например demo:archive-push.

Настройка хранения

Срок действия резервных копий в pgBackRest зависит от параметров хранения, в примере ниже будут сохранены два полных бэкапа:

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-path=/var/lib/ftdbackrest/17
repo1-retention-full=2

[global:archive-push]
compress-level=3
```

Настройка шифрования репозитория

Репозиторий будет настроен с использованием шифрования с ключом. Шифрование всегда выполняется на стороне клиента, даже если на репозитории поддерживается шифрование.

Важно использовать длинную парольную фразу для ключа шифрования, для этого можно использовать следующую команду:

```
openssl rand -base64 48
```

Пример конфигурационного файла pgbackrest.conf:

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/ftdbackrest/17
repo1-retention-full=2

[global:archive-push]
compress-level=3
```

После того, как репозиторий сконфигурирован, а станза создана и проверена, параметры шифрования репозитория изменить нельзя.

Создание станзы

Для инициализации станзы необходимо запустить команду создания станзы `stanza-create`, рекомендуется запустить команду проверки после создания станзы, чтобы обеспечить правильную настройку архивирования и резервного копирования.

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --log-level-console=info stanza-create
```

Проверка конфигурации

Команда `check` проверяет правильность настройки pgBackRest и параметра `archive_command` для архивирования и резервного копирования указанной станзы. `check` проверяет все репозитории и базы данных, настроенные для хоста, на котором выполняется команда. Команду можно запустить на хосте репозитория или FTData. Также `check` можно запустить на резервном хосте, но команда только проверит конфигурацию репозитория, поскольку pg_switch_xlog()/pg_switch_wal() нельзя выполнить на резервном хосте.

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --log-level-console=info check
```

pg_create_restore_point('pgBackRest Archive Check') и pg_switch_xlog()/pg_switch_wal() вызываются, чтобы FTData заархивировал сегмент WAL.

Выполнение резервного копирования

По умолчанию pgBackRest будет ждать следующей запланированной контрольной точки перед запуском резервного копирования. В зависимости от настроек `checkpoint_timeout` и `checkpoint_segments`, может пройти довольно много времени, прежде чем контрольная точка

завершится, и можно будет начать резервное копирование. Как правило, лучше всего установить `start-fast=y`, чтобы резервное копирование началось немедленно, поскольку резервное копирование обычно запускается один раз в день, дополнительная контрольная точка не должна оказывать заметного влияния на производительность. Однако на очень загруженных кластерах может быть лучше использовать параметр `--start-fast` в интерфейсе командной строки и делать это при необходимости.

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/ftdbackrest/17
repo1-retention-full=2
start-fast=y

[global:archive-push]
compress-level=3
```

Чтобы выполнить резервное копирование кластера FTData, запустите pgBackRest с командой `backup`:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo \
       --log-level-console=info backup
```

По умолчанию pgBackRest попытается выполнить инкрементное резервное копирование, однако инкрементное резервное копирование должно быть основано на полной резервной копии, и, поскольку ни одной полной резервной копии еще не было создано, pgBackRest вместо этого запустит полное резервное копирование. Для указания типа резервного копирования может использоваться параметр `--type`:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --type=diff \
       --log-level-console=info backup
```

Во время оперативного резервного копирования pgBackRest ожидает архивирования сегментов WAL, необходимых для согласованности резервного копирования. Это время ожидания определяется параметром `archive-timeout`, который по умолчанию равен 60 секундам, если известно, что архивирование отдельного сегмента занимает больше времени, следует увеличить этот параметр.

Информация о резервном копировании

Используйте команду `info` для получения информации о резервных копиях.

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest info
stanza: demo
    status: ok
    cipher: aes-256-cbc

    db (current)
        wal archive min/max (12): 000000010000000000000001/000000010000000000000005
        full backup: 20230130-013750F
            timestamp start/stop: 2023-01-30 01:37:50 / 2023-01-30 01:37:53
            wal start/stop: 000000010000000000000002 / 000000010000000000000003
            database size: 23.4MB, database backup size: 23.4MB
            repo1: backup set size: 2.8MB, backup size: 2.8MB
        diff backup: 20230130-013750F_20230130-013754D
            timestamp start/stop: 2023-01-30 01:37:54 / 2023-01-30 01:37:56
            wal start/stop: 000000010000000000000004 / 000000010000000000000005
            database size: 23.4MB, database backup size: 8.3KB
            repo1: backup set size: 2.8MB, backup size: 496B
            backup reference list: 20230130-013750F
```

Команда `info` может быть применена к одной или всем станзам, вывод информации довольно стандартный и дает краткую сводку о резервных копиях для запрошенных станз в удобочитаемом формате (этот формат может меняться при обновлении pgbackrest).

Для вывода более полной информации используйте команду `--output=json`. Вывод JSON содержит гораздо больше информации, чем вывод обычным текстом, и остается практически неизменным.

Информация о каждой станзе находится в отдельном разделе, информацию только об одной станзе можно вывести с помощью параметра `--stanza`. Команда `status` выводит краткую информацию о состоянии станзы, если статус "ok", то pgBackRest работает в обычном режиме. При наличии нескольких репозиториев статус "mixed" указывает на то, что станза не находится в работоспособном состоянии в одном или нескольких репозиториях; в этом случае состояние станзы будет детализировано для каждого репозитория. В случаях, когда произошла неизвестная ошибка, будет использоваться код ошибки "other", и будут предоставлены полные сведения об ошибке. Параметры `wal archive min/max` показывает минимальное и максимальное значение WAL, хранящееся в данный момент в архиве, и, в случае нескольких репозиториев, будет сообщаться обо всех репозиториях, если не установлен параметр `--repo`. Обратите внимание, что могут быть пробелы в тексте из-за политик хранения архивов или по другим причинам.

Сообщение "backup/expire running" печатается в случае, если одна из этих команд в настоящее время выполняется на хосте.

Резервные копии отображаются в порядке от самой старой к самой новой. Самая старая резервная копия всегда будет полной резервной копией (обозначается буквой F в конце названия), но самая новая резервная копия может быть полной, дифференциальной (заканчивается буквой D), инкрементной (заканчивается буквой I) или блочной (заканчивается буквой B).

'timestamp start/stop' определяет период времени, когда выполнялось резервное копирование. 'timestamp stop' можно использовать для определения резервной копии, которую следует использовать при выполнении восстановления на определенный момент времени.

'wal start/stop' определяет диапазон WAL, необходимый для обеспечения согласованности базы данных при восстановлении. Команда резервного копирования перед завершением гарантирует, что этот диапазон WAL находится в архиве.

'database size' — это полный несжатый размер базы данных, а 'database backup size' — это объем данных в базе данных для фактического резервного копирования (они будут одинаковыми для полных резервных копий).

'repo' указывает, в каком репозитории находится эта резервная копия. 'backup set size' включает в себя все файлы из этой резервной копии и любых резервных копий, на которые есть ссылки в репозитории, которые необходимы для восстановления базы данных из этой резервной копии, в то время как 'backup size' включает только файлы в этой резервной копии (они также будут одинаковыми для полных резервных копий). Размеры репозитория отражают размеры сжатых файлов, если сжатие включено в pgBackRest.

'backup reference list' содержит дополнительные резервные копии, необходимые для восстановления этой резервной копии.

Восстановление резервной копии

Резервные копии могут защитить от ряда негативных сценариев, наиболее распространенными из которых являются сбой оборудования и повреждение данных. Самый простой способ имитировать повреждение данных — удалить важный файл кластера FTData.

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
sudo -u postgres rm /var/ftdata/17/data/global/pg_control
```

Запуск кластера без этого важного файла приведет к ошибке. Чтобы восстановить резервную копию кластера FTData, нужно запустить pgBackRest с командой `restore`. Кластер необходимо остановить (в данном случае он уже остановлен) и удалить все файлы из каталога данных FTData.

```
sudo find /var/ftdata/17/data -mindepth 1 -delete
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo restore
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

В этом случае кластер успешно запустится, так как команда `restore` заменила отсутствующий файл pg_control.

Резервное копирование

Если настроено несколько репозиториев, pgBackRest будет выполнять резервное копирование в репозиторий с наивысшим приоритетом (например, repo1), если не указан параметр `--repo`.

pgBackRest не имеет встроенного планировщика, поэтому лучше запускать его из cron или другого механизма планирования.

Объединение файлов

Объединение файлов (File Bundling) экономит время, необходимое для резервного копирования, и место в репозитории. Это особенно заметно, когда репозиторий хранится в хранилище объектов, таком как S3.

Функция объединения файлов включается с помощью опции `repo-bundle` в конфигурационном файле.

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/ftdbackrest/17
repo1-retention-full=2
start-fast=y

[global:archive-push]
compress-level=3
```

Полная резервная копия без объединения файлов будет иметь более 1000 файлов, но с помощью объединения общее количество файлов значительно уменьшается. Дополнительным преимуществом является то, что файлы нулевой длины не сохраняются (кроме файла манифеста), тогда как при обычном резервном копировании каждый файл нулевой длины хранится отдельно.

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --type=full backup
```

Для настройки объединения файлов можно использовать параметры `repo-bundle-size` и `repo-bundle-limit`, хотя в большинстве случаев значения по умолчанию являются оптимальными.

Одним из недостатков объединения файлов является сложность извлечения файлов из репозитория вручную. Также в объединенные пакеты файлов нельзя дописать другие файлы при перезапуске, поэтому рекомендуется не устанавливать слишком большое

значение для параметра repo-bundle-size.

Аннотации резервного копирования

Пользователи могут прикреплять к резервной копии информативные пары ключ/значение (Backup Annotations), эту опцию можно использовать несколько раз, чтобы прикрепить несколько аннотаций.

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --annotation=source="demo backup" \
       --annotation=key=value --type=full backup
```

Аннотации всегда отображаются в JSON-выводе. Для того, чтобы вывести аннотацию в текстовом выводе, необходимо добавить параметр --set.

Аннотации, включенные в команду резервного копирования, могут быть добавлены, изменены или удалены впоследствии с помощью команды annotate.

Хранение

Как правило, лучше хранить как можно больше резервных копий, чтобы обеспечить большее окно для восстановления на определенный момент времени, но также необходимо учитывать практические аспекты, такие как дисковое пространство. Параметры хранения удаляют старые резервные копии, когда они больше не нужны.

pgBackRest выполняет ротацию полных резервных копий на основе типа хранения: количества резервных копий или периода времени. Когда указано количество, срок действия зависит не от того, когда были созданы резервные копии, а от того, сколько резервных копий необходимо сохранить. Срок действия дифференциальных, инкрементных и блочных резервных копий истекает, когда истекает срок действия резервной копии, от которой они зависят. Архивный WAL сохраняется по умолчанию для резервных копий, срок действия которых еще не истек, однако, хотя это и не рекомендуется, это расписание можно изменить для каждого репозитория с помощью параметров хранения-архивирования.

Команда expire запускается автоматически после каждого успешного резервного копирования, также она может быть запущена пользователем. При запуске пользователем срок действия истечет в соответствии с настройками хранения для каждого настроенного репозитория. Если указан параметр --repo, истечение срока действия произойдет только в указанном репозитории. Срок действия также может быть ограничен пользователем для определенного набора резервных копий с помощью параметра --set, и, если параметр --repo не указан, будут выполняться поиски во всех репозиториях, и срок действия любого, соответствующего установленным критериям, будет аннулирован. Следует отметить, что расписание хранения архива будет проверяться и выполняться каждый раз при запуске команды expire.

Хранение полной резервной копии

Параметр `repo1-retention-full-type` определяет, как интерпретируется опция `repo1-retention-full`: либо как количество полных резервных копий (Full Backup Retention), которые необходимо хранить, либо как количество дней хранения полных резервных копий. Новые резервные копии должны быть созданы до истечения срока действия. Это означает, что, если `repo1-retention-full-type=count` и `repo1-retention-full=2`, то до истечения срока действия самой старой из них будут храниться три полных резервных копии, или если `repo1-retention-full-type=time` и `repo1-retention-full=20`, тогда должна быть одна полная резервная копия, у которой есть не менее 20 дней, прежде чем истечет срок действия.

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
repo1-bundle=y
repo1-cipher-pass=zWaf6XtpjIVZC5444yXB+cgFDFl7MxGlgkZSaoPvTGirhPygu4jOKOXf9LO4vjfO
repo1-cipher-type=aes-256-cbc
repo1-path=/var/lib/ftdbackrest/17
repo1-retention-full=2
start-fast=y

[global:archive-push]
compress-level=3
```

В примере выше указано: `repo1-retention-full=2`, и есть только одна полная резервная копия, поэтому при создании еще одной полной резервной копии срок действия не истечет ни для одной полной резервной копии.

Хранение архива

Хотя pgBackRest автоматически удаляет заархивированные сегменты WAL при истечении срока действия резервных копий (по умолчанию срок действия WAL истекает для полных резервных копий на основе параметра `repo1-retention-full`), иногда может быть полезно удалять архивы с истекшим сроком действия, как можно раньше, - для экономии места на диске (Archive Retention). Обратите внимание, что полные резервные копии рассматриваются, как дифференциальные резервные копии с целью хранения дифференциальных архивов.

Архив с истекающим сроком действия никогда не удалит сегменты WAL, необходимые для согласованности резервной копии. Однако, поскольку восстановление на момент времени (PITR) работает только с непрерывным потоком WAL, следует соблюдать осторожность при удалении архива. Для того чтобы получить информацию об истечении срока действия архивов, можно воспользоваться командой `expire` с параметром `dry-run` (параметр `dry-run` может быть использован только в командной строке).

Восстановление

Команда восстановления по умолчанию выбирает последнюю резервную копию из первого репозитория, в котором существуют резервные копии. Порядок, в котором проверяются репозитории, определяется файлом pgbackrest.conf. Если для восстановления нужна резервная копия из определенного репозитория, то можно использовать параметр `--repo` (например, `--repo=2`). Если для восстановления нужна определенная резервная копия, то можно использовать параметр `--set`.

Если при восстановлении на определенный момент времени (PITR), указаны параметры `--type=time` или `--type=lsn`, то необходимое время или LSN нужно указать с помощью параметра `--target`. Если параметр `--set` был не указан, то настроенные репозитории будут проверены по порядку на наличие резервной копии, содержащей запрошенное время или номер LSN. Если подходящая резервная копия не найдена, последняя резервная копия из первого репозитория, содержащего резервные копии, будет использоваться для `--type=time`, а для `--type=lsn` резервная копия не будет выбрана.

Владение восстановленными файлами

Если восстановление выполняется от имени пользователя без полномочий суперпользователя, то все восстановленные файлы будут принадлежать пользователю/группе, запустившей pgBackRest. Если существующие файлы не принадлежат исполняющему пользователю/группе, то возникнет ошибка, если владельцем файлов нельзя установить исполняющего пользователя/группы. В этом случае владелец файлов должен быть установлен привилегированным пользователем перед повторной попыткой восстановления.

Если восстановление выполняется от имени суперпользователя, pgBackRest попытается воссоздать права собственности, записанные в манифесте при создании резервной копии. В манифесте хранятся только имена пользователей/групп, поэтому для того, чтобы это работало, такие же имена должны существовать на хосте восстановления. Если имя пользователя/группы не может быть найдено локально, будет использоваться пользователь/группа каталога данных FTData и, наконец, будет использован суперпользователь, если имя пользователя/группы каталога данных не может быть определено.

Параметр delta

Восстановление резервной копии в режиме быстрого запуска требует предварительной очистки каталога кластера базы данных. Параметр `delta` позволяет pgBackRest автоматически определять, какие файлы в каталоге кластера базы данных должны быть сохранены, а какие необходимо восстановить из резервной копии. Он также удаляет файлы, отсутствующие в манифесте резервной копии, чтобы избавиться от расходящихся изменений. Для определения различий вычисляется криптографический хэш SHA-1 для каждого файла в каталоге кластера базы данных. Если хэш SHA-1 не совпадает с хэшем, хранящимся в резервной копии, этот файл будет восстановлен. Для повышения эффективности можно также применить опцию `process-max`.

Восстановление выбранных баз данных

Для демонстрации этой возможности создадим две базы данных: `test1` и `test2`:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "create database test1;"
sudo -u postgres /opt/ftdata/17/bin/psql -c "create database test2;"
```

Заполним каждую базу данных таблицами и данными:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "create table test1_table (id int); \
       insert into test1_table (id) values (1);" test1
sudo -u postgres /opt/ftdata/17/bin/psql -c "create table test2_table (id int); \
       insert into test2_table (id) values (2);" test2
```

Сделаем резервную копию:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --type=incr backup
```

Остановим кластер и восстановим только базу данных `test2`. Встроенные базы данных (`template0`, `template1` и `postgres`) восстанавливаются всегда.

> Восстановление может привести к ошибке, если не указан параметр `--type=immediate`, это связано с тем, что после достижения согласованности FTData будет помечать обнуленные страницы, как ошибочные, даже при полной записи страницы. Параметр `ignore_invalid_pages` может использоваться для игнорирования недопустимых страниц. В этом случае важно проверить логи FTData после восстановления, чтобы убедиться, что в выбранных базах данных не было сообщений о недопустимых страницах.

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --delta \
       --db-include=test2 --type=immediate --target-action=promote restore
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

После завершения процесса восстановления база данных `test2` будет содержать все ранее созданные таблицы и данные:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "select * from test2_table;" test2
 id
-----
  2
(1 row)
```

База данных `test1`, несмотря на успешное восстановление, будет недоступна. Это связано с

тем, что вся база данных была восстановлена как несвязанные обнуленные файлы. FTData может успешно применить WAL к обнуленным файлам, но база данных в целом будет недействительной, поскольку ключевые файлы не содержат данных. Это необходимо для предотвращения случайного использования базы данных, если она может содержать частичные данные, примененные во время воспроизведения WAL.

Поскольку база данных `test1` восстанавливается с несвязанными, обнуленными файлами, для нее потребуется место, равное объему WAL, который записывается во время восстановления. Хотя объем WAL, сгенерированный во время резервного копирования и примененный во время восстановления, может быть значительным, обычно он составляет небольшую долю от общего размера базы данных.

Восстановление на момент времени

В случае повреждения данных, восстановление на момент времени (Point-in-Time Recovery) обычно является более подходящим вариантом восстановления.

Восстановление на момент времени позволяет воспроизводить WAL от последней резервной копии до указанного LSN, времени, идентификатора транзакции или точки восстановления. Самой распространенной причиной восстановления является восстановление случайно удаленной таблицы или случайно удаленных данных. В данном примере рассмотрим восстановление удаленной таблицы (восстановление удаленных данных будет проходить точно таким же образом).

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --type=diff backup
sudo -u postgres /opt/ftdata/17/bin/psql -c "begin; \
       create table important_table (message text); \
       insert into important_table values ('Important Data'); \
       commit;"
```

Важно указывать время в формате, который ожидает FTData, и принимать во внимание смещения часовых поясов. Это снижает вероятность непреднамеренного преобразования часового пояса и, как следствие, неожиданного результата восстановления.

```
sudo -u postgres /opt/ftdata/17/bin/psql -Atc "select current_timestamp"
2023-01-30 01:39:00.325554+00
```

Теперь, когда время записано, удаляем таблицу. На практике определить точное время удаления таблицы намного сложнее, чем в этом примере, а иногда даже бывает и невозможно (в этом случае, придется провести некоторое расследование для уточнения времени).

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "begin; \
       drop table important_table; \
       commit;"
```

После сохранения времени и удаления таблицы, проведем восстановление на момент времени:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --delta \
       --type=time --target="2023-01-30 01:39:00.325554+00" \
       --target-action=promote restore
sudo -u postgres tail -8 /var/ftdata/17/data/postgresql.auto.conf
```

Содержимое файла `/var/ftdata/17/data/postgresql.auto.conf`:

```
# Removed by pgBackRest restore on 2023-01-30 01:39:02 # recovery_target = 'immediate'
# Removed by pgBackRest restore on 2023-01-30 01:39:02 # recovery_target_action = 'promote'
# Removed by pgBackRest restore on 2023-01-30 01:39:02 # recovery_target_timeline = 'current'
# Recovery settings generated by pgBackRest restore on 2023-01-30 01:39:02
restore_command = 'pgbackrest --stanza=demo archive-get %f "%p"'
recovery_target_time = '2023-01-30 01:39:00.325554+00'
recovery_target_action = 'promote'
```

pgBackRest автоматически сгенерировал параметры восстановления в postgresql.auto.conf, поэтому FTData можно запустить сразу же после выполнения команды восстановления. Как только FTData завершит восстановление, таблица снова будет существовать, и к ней можно будет обращаться.

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
sudo -u postgres /opt/ftdata/17/bin/psql -c "select * from important_table"
    message
----------------
 Important Data
(1 row)
```

В логе FTData также содержится важная информация, такая, как время и транзакция, на которой восстановление было остановлено, а также указано время последней примененной транзакции.

В случае, если была выбрана резервная копия, созданная после нужного времени, то FTData не сможет восстановить потерянную таблицу (FTData может воспроизводить WAL файлы только вперед, но не назад).

В случае, если состояние на момент выбранной временной точки отсутствует в станзе, запуск кластера завершится ошибкой:

```
FATAL:  recovery ended before configured recovery target was reached
```

Надежный метод — позволить pgBackRest автоматически выбрать резервную копию, способную восстановиться до целевого времени, т. е. резервную копию, которая

завершилась до указанного времени.

Множественные репозитории

Можно настроить несколько репозиториев (Multiple Repositories). Потенциальным преимуществом такого решения является возможность иметь локальный репозиторий для быстрого восстановления и удаленный репозиторий для запасного хранения.

Некоторые команды, например, `stanza-create/stanza-upgrade` будут автоматически работать со всеми настроенными репозиториями, в то время как другим командам (например, `stanza-delete`) потребуется указать репозиторий с помощью параметра `repo`.

Команда `archive-push` всегда отправляет WAL в архив во все настроенные репозитории, но резервное копирование необходимо планировать отдельно для каждого репозитория, так как типы резервных копий и срок хранения зависят от репозитория. Аналогично, при восстановлении нужно также указывать репозиторий. Как правило, лучше указать репозиторий для восстановления с низкой задержкой/стоимостью, даже если это означает большее время восстановления. Только тестирование восстановления может определить, какой репозиторий будет наиболее эффективным.

Удаление станзы

Команда `stanza-delete` удаляет данные в репозитории, связанные с указанной станзой.

> Используйте эту команду с осторожностью, — она безвозвратно удалит все резервные копии и архивы из репозитория pgBackRest для указанной станзы.

Чтобы удалить станзу необходимо выполнить следующие шаги:

- завершить работу кластера FTData, связанного со станзой:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
```

- запустить команду `stop` на хосте, где будет выполняться команда удаления станзы:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --log-level-console=info stop
```

- выполнить команду `stanza-delete`:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --repo=1 \
       --log-level-console=info stanza-delete
```

После успешного выполнения команды пользователь должен удалить станзу из всех конфигурационных файлов pgBackRest и/или переменных среды.

Чтобы удалить станзу из нескольких репозиториев, нужно выполнить команду `stanza-delete` с указанием параметра `--repo` для каждого репозитория.

Резервное копирование на выделенный хост репозитория

Конфигурация, описанная в разделе "Начало работы", подходит для простых конфигураций, однако очень часто выделяют отдельный хост репозитория, где хранятся резервные копии и архивные файлы WAL. Такой подход разделяет резервные копии и архив WAL от сервера базы данных, поэтому сбои хоста базы данных оказывают меньшее влияние.

На хостах FTData путь `pg1-path` должен быть путем к локальному кластеру FTData, также `pg1-host` не должен быть настроен. При настройке хоста репозитория в конфигурационном файле pgbackrest должна быть настроена опция `pg1-host` для подключения к основному и резервному (если есть) хостам. У хоста репозитория должны быть настроена только одна конфигурация pgbackrest, которая должна учитывать наличие более одного хоста FTData, при этом порядок не имеет значения, то есть pg1-path/pg1-host, pg2-path/pg2-host могут быть первичными или резервными.

Установка

Для установки выделенного хоста репозитория нужно выполнить следующие шаги:

- создать новый хост с именем репозитория для хранения резервных копий кластера;

> Версия pgBackRest, установленная на хосте репозитория, должна точно совпадать с версией, установленной на хосте FTData.

- создать пользователя pgbackrest для владения репозиторием pgBackRest. Любой пользователь может владеть репозиторием, но лучше не использовать postgres (если он существует), чтобы избежать путаницы:

```
sudo adduser --disabled-password --gecos "" pgbackrest
```

- установить пакеты `libpq`, `ftdbackrest` из дистрибутива FTData (см. раздел Установка ftdbackrest отдельно от СУБД FTData)
- создать каталоги журнала и конфигурации, а также файл конфигурации для pgBackRest:

```
sudo mkdir -p -m 770 /var/log/pgbackrest
sudo chown pgbackrest:pgbackrest /var/log/pgbackrest
sudo mkdir -p /etc/pgbackrest
sudo touch /etc/pgbackrest/pgbackrest.conf
sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
sudo chown pgbackrest:pgbackrest /etc/pgbackrest/pgbackrest.conf
sudo mkdir -p /var/lib/ftdbackrest/17
sudo chmod 750 /var/lib/ftdbackrest/17
sudo chown pgbackrest:pgbackrest /var/lib/ftdbackrest/17
```

Настройка беспарольного SSH

pgBackRest может использовать SSH без пароля (passwordless SSH) для связи между хостами. Также можно использовать TLS.

Для настройки беспарольного SSH нужно:

- сгенерировать пару ключей на репозитории;
- сгенерировать пару ключей на основном хосте;
- обменять ключи между репозиторием и основным хостом.

Настройка

Хост репозитория должен быть настроен с указанием хоста/пользователя первичного хоста (pg-primary) и пути к базе данных. Первичный сервер будет настроен, как pg1, чтобы можно было добавить резервный сервер позже.

Содержимое файла `/etc/pgbackrest/pgbackrest.conf` для хоста репозитория:

```
[demo]
pg1-host={url или IP хоста с БД для бэкапа}
pg1-host-user=postgres
pg1-path=/var/ftdata/17/data
repo1-path=/var/lib/ftdbackrest/17

[global]
repo1-retention-full=2
start-fast=y
log-level-console=info
```

Хост базы данных должен быть настроен с хостом/пользователем репозитория. По умолчанию для параметра `repo1-host-user` используется pgbackrest. Если пользователь postgres выполняет восстановление на хосте репозитория, лучше не позволять пользователю postgres также выполнять резервное копирование. Однако пользователь postgres может напрямую читать репозиторий, если он находится в той же группе, что и пользователь pgbackrest.

Содержимое файла `/etc/pgbackrest/pgbackrest.conf` для хоста кластера БД:

```
[demo]
pg1-path=/var/ftdata/17/data
repo1-host={url или IP хоста с бэкапом}
repo1-host-user=pgbackrest

[global]
log-level-console=info
```

Команды выполняются так же, как и в конфигурации с одним хостом, за исключением того, что некоторые команды, такие как резервное копирование и истечение срока действия, выполняются с хоста репозитория, а не с хоста базы данных.

Создание и проверка станзы

Создайте станзу в новом репозитории (выполнить на хосте репозитория):

```
sudo -u pgbackrest /opt/ftdata/17/bin/pgbackrest --stanza=demo stanza-create
sudo -u pgbackrest /opt/ftdata/17/bin/pgbackrest --stanza=demo check
```

Убедитесь, что конфигурация верна как на хостах базы данных, так и на хостах репозитория (выполнить на хосте кластера БД):

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo check
```

Выполнение резервного копирования

Чтобы выполнить резервное копирование кластера FTData, запустите pgBackRest с командой резервного копирования на хосте репозитория:

```
sudo -u pgbackrest /opt/ftdata/17/bin/pgbackrest --stanza=demo backup
```

Восстановление из резервной копии

Чтобы выполнить восстановление кластера FTData, запустите pgBackRest с командой восстановления на хосте базы данных:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --delta restore
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

Параллельное резервное копирование и восстановление

Для повышения производительности может быть применена параллельная обработка (Parallel Backup / Restore), количество процессов, которые будут использоваться для этой функции, задается с помощью параметра `--process-max`.

Обычно лучше не использовать более 25% доступных ЦП для команды резервного копирования. Резервное копирование не должно выполняться максимально быстро на регулярной основе, и процесс резервного копирования не должен влиять на производительность базы данных, если это возможно. Для очень маленьких резервных копий разница может быть не очень очевидной, но по мере увеличения размера базы данных экономия времени будет увеличиваться.

Команда восстановления может и должна использовать все доступные процессоры, так как во время восстановления кластер FTData выключается, и на хосте, как правило, не выполняется никакой другой важной работы. Если хост содержит несколько кластеров, это следует учитывать при настройке параллелизма восстановления.

Запуск и остановка

Бывают случаи, когда необходимо предотвратить запуск pgBackRest в системе. Например, при переключении с основного сервера на резервный лучше запретить запуск pgBackRest на старом основном сервере на случай, если FTData перезапустится, или его процессы нельзя будет полностью завершить.

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest stop
```

Теперь новые процессы pgBackRest не смогут запуститься.

Укажите параметр `--force`, чтобы завершить любые запущенные в данный момент процессы pgBackRest. При попытке остановить уже остановленный pgBackRest появится предупреждающее сообщение.

Запустите процессы pgBackRest с помощью команды `start`:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest start
```

Также есть возможность остановить(или запустить) pgBackRest для одной станзы:

```
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo stop
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo start
```

Репликация

Репликация (replication) позволяет создавать несколько копий кластера FTData (называемых резервными) из одного первичного.

Установка

Для установки резервного сервера потребуется:

- Установить FTData (см. раздел Установка сервера СУБД FTData) на хост репликации.
- Создать каталоги журнала и конфигурации, а также файл конфигурации для pgBackRest:

```
sudo mkdir -p -m 770 /var/log/pgbackrest
sudo chown postgres:postgres /var/log/pgbackrest
sudo mkdir -p /etc/pgbackrest
sudo touch /etc/pgbackrest/pgbackrest.conf
sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
sudo chown postgres:postgres /etc/pgbackrest/pgbackrest.conf
```

Настройка беспарольного SSH

pgBackRest может использовать SSH без пароля (passwordless SSH) для связи между хостами. Также можно использовать TLS. Для настройки беспарольного SSH нужно:

- сгенерировать пару ключей на хосте;
- обменять ключи между репозиторием и pg-standby.

Горячее резервирование

Сервер базы данных, который выполняет репликацию с использованием архива WAL и разрешает доступ только в режиме чтения, называется Горячий резерв (Hot Standby).

Конфигурация pgBackRest очень похожа на pg-primary, за исключением того, что резервный тип восстановления будет использоваться для удержания кластера в режиме восстановления, когда будет достигнут конец потока WAL.

Настройка pgBackRest на резервном сервере:

```
[demo]
repo1-host={url или IP хоста с БД для репликации}
repo1-host-user=postgres
pg1-path=/var/ftdata/17/data

[global]
log-level-file=info
```

Подготовка демонстрационного кластера

Подготовьте демонстрационный кластер БД (далее - кластер). В примерах ниже используется кластер, расположенный в `/var/ftdata/17/data`.

Создание резервного кластера

Теперь резерв можно создать с помощью команды `restore`.

> Если кластер не предназначен для того, чтобы стать новым первичным кластером (а предназначен, например, для отчетов или тестирования), используйте `--archive-mode=off` или установите `archive_mode=off` в postgresql.conf, чтобы отключить архивирование. Если архивирование не отключено, репозиторий может быть перегружен WAL-записями, что может затруднить восстановление.

Восстановление демонстрационного резервного кластера:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --delta --type=standby restore
sudo -u postgres cat /var/ftdata/17/data/postgresql.auto.conf
```

Чтобы разрешить соединения только для чтения на pg-standby, нужно включить параметр `hot_standby` перед запуском FTData, иначе попытки подключения будут отклонены. Остальная часть конфигурации предназначена для случая, когда резервный сервер становится основным:

```
archive_command = '/opt/ftdata/17/bin/pgbackrest --stanza=demo archive-push %p'
archive_mode = on
hot_standby = on
max_wal_senders = 3
wal_level = replica
```

Запуск сервера:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

В логе FTData содержится много полезной информации о процессе восстановления. Обратите особое внимание на то, чтобы кластер перешел в режим ожидания и был готов принимать подключения только для чтения.

Простой способ проверить правильность настройки репликации — создать таблицу на pg-primary, а затем запросить ту же таблицу в pg-standby.

Создание новой таблицы в pg-primary:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c " \
begin; \
create table replicated_table (message text); \
insert into replicated_table values ('Important Data'); \
commit; \
select * from replicated_table";
message
----------------
 Important Data
(1 row)
```

Запрос из pg-standby:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "select * from replicated_table;"
ERROR:  relation "replicated_table" does not exist
LINE 1: select * from replicated_table;
```

Поскольку FTData извлекает сегменты WAL из архива для выполнения репликации, изменения не будут видны на резервном сервере до тех пор, пока сегмент WAL, содержащий эти изменения, не будет отправлен из pg-primary. Это можно сделать вручную, вызвав функцию pg_switch_wal(), которая помещает текущий сегмент WAL в архив (будет создан новый сегмент WAL, который будет содержать дальнейшие изменения):

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "select *, current_timestamp from pg_switch_wal()";
```

После некоторого времени таблица появится на pg-standby.

Потоковая репликация

Вместо того чтобы полагаться исключительно на архив WAL, потоковая репликация (Streaming Replication) устанавливает прямое подключение к основному серверу и применяет изменения, как только они вносятся в основной. Это приводит к гораздо меньшему отставанию между основным и резервным серверами.

Для потоковой репликации требуется пользователь с правами репликации:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c " \
       create user replicator password 'jw8s0F4' replication";
```

Файл pg_hba.conf необходимо обновить, чтобы резервный сервер мог подключаться в качестве пользователя с правами репликации. Также нужно заменить приведенный ниже IP-адрес фактическим IP-адресом вашего pg-standby. После изменения файла pg_hba.conf потребуется перезагрузка.

```
sudo -u postgres sh -c 'echo \
       "host    replication     replicator      172.17.0.7/32           md5" \
       >> /var/ftdata/17/data/pg_hba.conf'
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data reload
```

Для того чтобы резервный сервер мог связаться с основным, нужно настроить параметр `primary_conninfo` в конфигурационном файле pgBackRest:

```
[demo]
pg1-path=/var/ftdata/17/data
recovery-option=primary_conninfo=host=172.17.0.5 port=5432 user=replicator

[global]
log-level-file=detail
repo1-host=repository
```

Можно настроить пароль в параметре primary_conninfo, но использование файла .pgpass является более гибким и безопасным:

```
sudo -u postgres sh -c 'echo \
       "172.17.0.5:*:replication:replicator:jw8s0F4" \
       >> /var/lib/postgresql/.pgpass'
sudo -u postgres chmod 600 /var/lib/postgresql/.pgpass
```

Теперь резервный сервер может быть создан с помощью команды `restore`:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data stop
sudo -u postgres /opt/ftdata/17/bin/pgbackrest --stanza=demo --delta --type=standby restore
sudo -u postgres cat /var/ftdata/17/data/postgresql.auto.conf
```

Параметр `primary_conninfo` был записан в файл postgresql.auto.conf, поскольку он был настроен, как параметр восстановления в pgbackrest.conf. Параметр `--type=preserve` можно использовать при восстановлении, чтобы оставить существующий файл postgresql.auto.conf на месте, если такое поведение предпочтительнее.

Запускаем кластер:

```
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data start
```

Теперь, когда таблица создается на pg-primary, она появляется на pg-standby практически сразу же и без необходимости вызова pg_switch_wal().

Асинхронное архивирование

Асинхронное архивирование (Asynchronous Archiving) включается с помощью параметра `archive-async`. Этот параметр включает асинхронную операцию как для команд `archive-push`, так и для `archive-get`. Также необходимо настроить путь до буфера (spool path), - команды будут хранить здесь временные данные.

Создадим папку для буферизации на pg-primary:

```
sudo mkdir -p -m 750 /var/spool/pgbackrest
sudo chown postgres:postgres /var/spool/pgbackrest
```

Создадим папку для буферизации на pg-standby:

```
sudo mkdir -p -m 750 /var/spool/pgbackrest
sudo chown postgres:postgres /var/spool/pgbackrest
```

Кроме настройки пути до буфера, нужно включить асинхронное архивирование. Асинхронное архивирование дает некоторое преимущество за счет уменьшения количества подключений к удаленному хранилищу, но установка параметра `process-max` может значительно повысить производительность за счет распараллеливания операций. Убедитесь, что параметр `process-max` не установлен настолько большим, чтобы это повлияло на нормальную работу базы данных.

Настройка файла конфигурации на pg-primary:

```
[demo]
pg1-path=/var/ftdata/17/data

[global]
archive-async=y
log-level-file=detail
repo1-host=repository
spool-path=/var/spool/pgbackrest

[global:archive-get]
process-max=2

[global:archive-push]
process-max=2
```

Настройка файла конфигурации на pg-standby:

```
[demo]
pg1-path=/var/ftdata/17/data
recovery-option=primary_conninfo=host=172.17.0.5 port=5432 user=replicator

[global]
archive-async=y
log-level-file=detail
repo1-host=repository
spool-path=/var/spool/pgbackrest

[global:archive-get]
process-max=2

[global:archive-push]
process-max=2
```

Параметр `process-max` не используется при резервном копировании и восстановлении, это позволяет использовать разные значения для `archive-push` и `archive-get`.

В демонстрационных целях потоковая репликация будет прервана, чтобы заставить FTData получить WAL с помощью команды `restore_command`:

```
sudo -u postgres /opt/ftdata/17/bin/psql -c "alter user replicator password 'bogus'"
sudo -u postgres /opt/ftdata/17/bin/pg_ctl -D /var/ftdata/17/data restart
```

Команда archive-push

Асинхронная команда `archive-push` переносит архивирование WAL в отдельный процесс (или процессы) для повышения пропускной способности. Он работает, "заглядывая вперед", чтобы увидеть, какие сегменты WAL готовы к архивированию. Сегменты WAL передаются в архив непосредственно из каталога pg_xlog/pg_wal, и команда `archive_command` считается успешно выполненной только после того, как сегмент WAL был сохранен в архиве.

Путь до буфера (spool path) содержит текущий статус архивации WAL. Файлы состояния, записанные в папку буферизации, обычно имеют нулевую длину и должны занимать минимальное количество места (максимум несколько МБ) и очень мало операций ввода-вывода. Всю информацию в этом каталоге можно воссоздать, поэтому нет необходимости сохранять каталог буфера, если кластер перемещается на новое оборудование.

Файл `[stanza]-archive-push-async.log` можно использовать для мониторинга активности асинхронного процесса.

Команда archive-get

Асинхронная команда `archive-get` поддерживает локальную очередь WAL для повышения пропускной способности. Если сегмент WAL не найден в очереди, он извлекается из

репозитория вместе с достаточным количеством последовательных сегментов WAL для заполнения очереди. Максимальный размер очереди определяется параметром `archive-get-queue-max`. Всякий раз, когда очередь опустошается больше, чем наполовину, для ее заполнения будет загружаться больше WAL.

Асинхронная работа наиболее полезна в средах, которые генерируют много WAL или имеют соединение с хранилищем репозитория с высокой задержкой. В случае подключения с высокой задержкой может быть полезно увеличить значение `process-max`.

Файл `[stanza]-archive-get-async.log` можно использовать для мониторинга активности асинхронного процесса.

Резервное копирование из резервного сервера

pgBackRest может выполнять резервное копирование из резервного сервера вместо основного (Backup from a Standby), для этого необходимо настроить хост pg-standby и включить опцию резервного копирования. Если настроено более одного резервного сервера, то для резервного копирования будет использоваться первый найденный работающий резервный сервер.

```
[demo]
pg1-host=pg-primary
pg1-path=/var/ftdata/17/data
pg2-host=pg-standby
pg2-path=/var/ftdata/17/data

[global]
[yellow-background]*backup-standby=y*
process-max=3
repo1-path=/var/lib/ftdbackrest/17
repo1-retention-full=2
start-fast=y
```

Для выполнения резервного копирования требуются как первичная, так и резервная базы данных, хотя подавляющее большинство файлов будет скопировано из резервной базы данных, чтобы снизить нагрузку на первичную. Хосты базы данных можно настроить в любом порядке, pgBackRest автоматически определит, какой из них является основным, а какой — резервным.

pgBackRest создает резервную копию, идентичную резервной копии, созданной на основном сервере. Для этого он запускает/останавливает резервное копирование на первичном хосте pg-primary и копирует только те файлы, которые реплицированы с хоста pg-standby, а затем копирует несколько оставшихся файлов с pg-primary. Это означает, что логи и статистика из первичной базы данных будут включены в резервную копию.

Расширение ftd_profile

Общие сведения

Расширение `ftd_profile` предназначено для сбора исторических данных нагрузки на сервера СУБД FTData. За основу было взято расширение pg_profile, и добавлен функционал по сбору метрик операционной системы. Установленное расширение позволяет собирать данные нагрузки со всех доступных серверов FTData. При установке расширения создается репозиторий для хранения исторических данных и набор функций для сбора данных и формирования отчетов. СУБД FTData не имеет механизма формирования заданий (job), поэтому необходимо использовать cron-подобный инструмент для периодического запуска функции получения снимка. Более подробное описание расширения pg_profile доступно в базовой документации.

Быстрый старт

Расширение входит в дистрибутив FTData и не требует для себя дополнительных библиотек и дистрибутивных пакетов. Требуется предварительная установка расширений: `file_fdw`, `pg_stat_statements`, `dblink`, которые также включены в дистрибутив. Желательна, но не обязательна, установка расширения `pg_wait_sampling`, которое включено в дистрибутив, начиная с версии 17.2.1. Когда расширение `pg_wait_sampling` установлено, расширение `ftd_profile` будет дополнительно собирать статистику по событиям ожидания для процессов в СУБД.

Конфигурация

- Один сервер `local` с установленной СУБД FTData;
- База данных - "nbs";
- Логин суперпользователя - "postgres";
- Логин пользователя для установки `ftd_profile` - "profile_usr";
- Схема для установки расширения - "ftd_profile".
- Параметры `ftd_profile`:
  - количество дней хранения данных = 7 (значение по умолчанию);
  - количество запросов в отчете = 20 (значение по умолчанию);
  - максимальная длина запроса = 20000 (значение по умолчанию).
- Временное окно для получения размера объектов базы данных - 2 часа, начиная с 23 ч. (TMZ +3). (Так как сбор этой метрики нагружает СУБД, обязательно выполните эту настройку);
- Периодичность запуска получения метрик - 1 раз в 30 минут;
- Сбор метрик ОС - включить все (по умолчанию все метрики ОС выключены);

- Метод аутентификации описан в файле `hba.conf`, для host - "md5".

Настройка экземпляра Postgres

Для настройки экземпляра СУБД FTData необходимо отредактировать конфигурационный файл `postgresql.conf`, добавив в него следующие строки, и перезапустить экземпляр.

```
shared_preload_libraries = 'dbi_funcs,pg_stat_statements,pg_wait_sampling'
track_counts = on
```

Выполнить команды psql на сервере `local`:

```
\c nbs postgres
CREATE ROLE profile_usr login password 'profile_pwd';
CREATE SCHEMA dblink;
CREATE EXTENSION dblink SCHEMA dblink;
CREATE SCHEMA stat_statements;
CREATE EXTENSION pg_stat_statements SCHEMA stat_statements;
CREATE SCHEMA wait_sampling;
CREATE EXTENSION pg_wait_sampling SCHEMA wait_sampling;
CREATE SCHEMA file_fdw;
CREATE EXTENSION file_fdw SCHEMA file_fdw;
CREATE SCHEMA ftd_profile AUTHORIZATION profile_usr;
GRANT USAGE ON SCHEMA dblink to profile_usr;
GRANT USAGE ON SCHEMA file_fdw to profile_usr;
GRANT USAGE ON SCHEMA stat_statements to profile_usr;
GRANT USAGE ON SCHEMA wait_sampling to profile_usr;
GRANT USAGE ON FOREIGN DATA WRAPPER file_fdw TO profile_usr;
GRANT pg_execute_server_program TO profile_usr;
GRANT pg_read_server_files TO profile_usr;
GRANT pg_read_all_stats TO profile_usr;
GRANT EXECUTE ON FUNCTION stat_statements.pg_stat_statements_reset(oid,oid,int8) TO profile_usr;
GRANT EXECUTE ON FUNCTION wait_sampling.pg_wait_sampling_reset_profile TO profile_usr;
GRANT SELECT ON ftd_stat_subxids TO profile_usr;
\c nbs profile_usr
CREATE EXTENSION pg_wait_sampling SCHEMA wait_sampling;
CREATE EXTENSION ftd_profile SCHEMA ftd_profile;
SELECT ftd_profile.set_server_connstr('local','dbname=nbs port=5432 host=localhost user=profile_usr
password=profile_pwd');
SELECT ftd_profile.set_server_size_sampling('local','23:00+03',interval '2 hour',interval '2 hour');
SELECT ftd_profile.set_os_metric(NULL, true);
```

Необходимо настроить `crontab` для владельца СУБД в ОС - пользователя `postgres`:

```
0,30 * * * *   psql -U profile_usr -d nbs -c 'SELECT ftd_profile.take_sample()' > /dev/null 2>&1
```

Установка и настройка `ftd_profile` завершены.

Для выгрузки результатов мониторинга для последующего анализа нужно в утилите psql выполнить:

```
\copy (select * from export_data()) to 'export.csv'
```

Для загрузки результатов мониторинга нужно выполнить:

```
CREATE TABLE ftd_profile.import (section_id bigint, row_data json);
\copy ftd_profile.import from 'export.csv'
SELECT * FROM ftd_profile.import_data('ftd_profile.import','Любое имя для сервера ');
DROP TABLE ftd_profile.import;
```

Пример скриптов для автоматизации установки `ftd_profile`

Скрипт для подготовки окружения `ftd_profile` на сервере `local`:

```
#!/bin/bash
PG_PATH=/opt/ftdata/17/bin
USERNAME=postgres
DB=nbs

$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS ftd_profile;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS dblink;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS pg_stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS pg_wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS dblink;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS ftd_profile cascade;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP ROLE IF EXISTS profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE ROLE profile_usr login password 'profile_pwd';"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA dblink;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION dblink SCHEMA dblink;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION pg_stat_statements SCHEMA stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION pg_wait_sampling SCHEMA wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION file_fdw SCHEMA file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA ftd_profile AUTHORIZATION profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA dblink to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA file_fdw to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA stat_statements to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA wait_sampling to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT usage ON FOREIGN DATA WRAPPER file_fdw TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT pg_execute_server_program  TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT pg_read_server_files TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT pg_read_all_stats TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT EXECUTE ON FUNCTION stat_statements.pg_stat_statements_reset(oid,oid,int8)
TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT EXECUTE ON FUNCTION wait_sampling.pg_wait_sampling_reset_profile TO
profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT SELECT ON ftd_stat_subxids TO profile_usr;"
```

Скрипт для подготовки окружения `ftd_profile` на внешнем сервере `remote`:

```
#!/bin/bash
PG_PATH=/opt/ftdata/17/bin
USERNAME=postgres
DB=nbs

$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS ftd_profile;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS pg_stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS pg_wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP EXTENSION IF EXISTS file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP SCHEMA IF EXISTS ftd_profile cascade;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "DROP ROLE IF EXISTS profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE ROLE profile_usr login password 'profile_pwd';"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION pg_stat_statements SCHEMA stat_statements;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION pg_wait_sampling SCHEMA wait_sampling;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION file_fdw SCHEMA file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SCHEMA ftd_profile AUTHORIZATION profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE SERVER ftd_fileserver FOREIGN DATA WRAPPER file_fdw;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA file_fdw to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA stat_statements to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON SCHEMA wait_sampling to profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT USAGE ON FOREIGN SERVER ftd_fileserver TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT pg_execute_server_program TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT pg_read_server_files TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT pg_read_all_stats TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT EXECUTE ON FUNCTION stat_statements.pg_stat_statements_reset(oid,oid,int8)
TO profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT EXECUTE ON FUNCTION wait_sampling.pg_wait_sampling_reset_profile TO
profile_usr;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "GRANT SELECT ON ftd_stat_subxids TO profile_usr;"
```

Скрипт для установки и настройки расширения `ftd_profile` на сервере `local`:

```
#!/bin/bash
PG_PATH=/opt/ftdata/17/bin
USERNAME=profile_usr
DB=nbs

$PG_PATH/psql -U $USERNAME -d $DB  -c "CREATE EXTENSION ftd_profile SCHEMA ftd_profile;"
$PG_PATH/psql -U $USERNAME -d $DB  -c "SELECT ftd_profile.set_server_connstr('local','dbname=$DB port=5432 host=localhost
user=profile_usr password=profile_pwd');"
$PG_PATH/psql -U $USERNAME -d $DB  -c "SELECT ftd_profile.create_server('remote','dbname=$DB port=5432 host=remote.foo.ru
user=profile_usr password=profile_pwd');"
$PG_PATH/psql -U $USERNAME -d $DB  -c "SELECT ftd_profile.create_fdw_objects('remote');"
$PG_PATH/psql -U $USERNAME -d $DB  -c "SELECT ftd_profile.set_server_size_sampling('local','23:00+03',interval '2
hour',interval '2 hour');"
$PG_PATH/psql -U $USERNAME -d $DB  -c "SELECT ftd_profile.set_server_size_sampling('remote','23:00+03',interval '2
hour',interval '2 hour');"
$PG_PATH/psql -U $USERNAME -d $DB  -c "SELECT ftd_profile.set_os_metric(NULL, true);"
```

Требования

Расширения

Для работы `ftd_profile` необходимы следующие расширения:

- `plpgsql` - процедурный язык PL/pgSQL;
- `dblink` - модуль подключения к внешней базе данных;
- `pg_stat_statements` - статистика планирования и выполнения сервером SQL-запросов;
- `file_fdw` - реализует обертку сторонних данных, с помощью которой можно обращаться к результатам системных команд и файлам данных операционной системы сервера;
- `pg_wait_sampling` - сбор статистики по событиям ожидания (необязательное расширение).

Параметры конфигурации

- `shared_preload_libraries` - в этом параметре указываются библиотеки, загружаемые при запуске сервера. Для работы расширения `ftd_profile` в список библиотек необходимо добавить `pg_stat_statements` и, при необходимости, `pg_wait_sampling`;
- `compute_query_id` - вычисление идентификаторов запросов. Для работы расширения `ftd_profile` данный параметр необходимо установить в значение `on` или `auto`;
- `track_io_timing` - включение статистики операций чтения/записи. Внимание! Может замедлять работу сервера. При сборе статистики по нагрузке параметр необходимо установить в значение `on`;
- `track_activities` - включение мониторинга текущих команд, выполняемых любым серверным процессом. Данный параметр необходимо установить в значение `on`;
- `track_counts` - параметр определяет необходимость сбора статистики по обращениям к таблицам и индексам. Данный параметр необходимо установить в значение `on`;
- `track_wal_io_timing` - включение статистики операций чтения/записи WAL. Внимание! Может замедлять работу сервера. При сборе статистики по нагрузке параметр необходимо установить в значение `on`;
- `track_functions` - включение отслеживания использования пользовательских функций. Данный параметр необходимо установить в значение `all` или `pl`;
- `pg_stat_statements.max` - максимальное число операторов, отслеживаемых модулем. Параметр задается при запуске сервера. Рекомендуемое значение параметра - `10000`;
- `pg_stat_statements.track` - параметр определяет, какие операторы будут отслеживаться. Для сбора данных по вложенным операторам значение необходимо установить в `all`;
- `pg_stat_statements.save` - параметр определяет, должна ли статистика операторов сохраняться после перезагрузки сервера. Данный параметр необходимо установить в значение `on`;
- `ftd_profile.topn` - количество основных объектов (выражений, отношений и так далее), которые должны отображаться в каждой отсортированной таблице отчета. Значение по умолчанию - `20`;
- `ftd_profile.min_lock_time` - определяет минимальное время собираемых блокировок в секундах. Значение по умолчанию - `5`;

- `ftd_profile.max_sample_age` - время хранения собранных метрик в днях. При очередном запуске функции сбора метрик объекты старше указанного параметра удалятся. Значение по умолчанию - `7`;
- `ftd_profile.max_query_length` - ограничение длины SQL-запроса в символах для отчетов; при этом сам запрос будет храниться полностью. Значение по умолчанию - `20000`.
- `ftd_profile.track_sample_timings` - При установке значения параметра в `on` собирается информация о времени, затраченном при сборе каждой метрики. Это полезно для анализа и снятия избыточной нагрузки с сервера СУБД, создаваемой расширением `ftd_profile`. Результаты можно посмотреть в представлении `v_sample_timings`.

Установка и настройка расширения ftd_profile

Для установки расширения `ftd_profile` необходимо произвести базовые настройки:

- создать пользователя для расширения;
- установить вспомогательные расширения (`dblink`, `pg_stat_statements`, `file_fdw` ,`pg_wait_sampling`), если они не установлены;
- установить расширение `ftd_profile`;
- выдать необходимые привилегии;
- провести настройки для `ftd_profile` на внешних серверах, по которым собираются данные исторической нагрузки.

БД для установки расширения ftd_profile

Создание пользователя

Для соблюдения требований безопасности рекомендуется создать непривилегированного пользователя, так как в описании сервера строка соединения содержится в открытом виде. При экспорте собранных метрик эта информация будет доступна в файле данных.

```
CREATE ROLE profile_usr LOGIN PASSWORD profile_pwd;
```

Установка вспомогательных расширений

Для каждого расширения создается отдельная схема. Данный этап рекомендуется для выполнения, иначе все объекты будут созданы в схеме `public` или в схеме по умолчанию:

Создать схему для расширения `dblink`:

```
CREATE SCHEMA dblink;
```

Создать расширение `dblink`:

```
CREATE EXTENSION dblink SCHEMA dblink;
```

Создать схему для расширения `pg_stat_statements`:

```
CREATE SCHEMA stat_statements;
```

Создать расширение `pg_stat_statements`:

```
CREATE EXTENSION pg_stat_statements SCHEMA stat_statements;
```

Создать схему для расширения `pg_wait_sampling`:

```
CREATE SCHEMA wait_sampling;
```

Создать расширение `pg_wait_sampling`:

```
CREATE EXTENSION pg_wait_sampling SCHEMA wait_sampling;
```

Создать схему для расширения `file_fdw`:

```
CREATE SCHEMA file_fdw;
```

Создать расширение `file_fdw`:

```
CREATE EXTENSION file_fdw SCHEMA file_fdw;
```

Установка расширения

Создать схему для расширения `ftd_profile` под суперпользователем:

```
CREATE SCHEMA ftd_profile AUTHORIZATION profile_usr;
```

Установить расширение `ftd_profile` под пользователем **profile_usr**:

```
CREATE EXTENSION ftd_profile SCHEMA ftd_profile;
```

Предоставление прав

Предоставление пользователю **profile_usr** прав на объекты:

```
GRANT USAGE ON SCHEMA dblink TO profile_usr;
GRANT USAGE ON SCHEMA file_fdw TO profile_usr;
GRANT USAGE ON SCHEMA stat_statements TO profile_usr;
GRANT USAGE ON SCHEMA wait_sampling TO profile_usr;
GRANT USAGE ON FOREIGN DATA WRAPPER file_fdw TO profile_usr;
GRANT pg_execute_server_program TO profile_usr;
GRANT pg_read_server_files TO profile_usr;
GRANT pg_read_all_stats TO profile_usr;
GRANT EXECUTE ON FUNCTION stat_statements.pg_stat_statements_reset TO profile_usr;
GRANT EXECUTE ON FUNCTION wait_sampling.pg_wait_sampling_reset_profile TO profile_usr;
GRANT SELECT ON ftd_stat_subxids TO profile_usr;
```

Внешний сервер БД

Для сбора метрик со стороннего сервера СУБД FTData требуется:

- создать пользователя с возможностью подключения;
- установить необходимые расширения;
- выдать права на использование расширений;
- создать объекты для сбора метрик ОС.

Создание пользователя

Создание пользователя для расширения `ftd_profile`:

```
CREATE ROLE profile_mon LOGIN PASSWORD strong_pwd;
```

Необходимо убедиться, что у пользователя есть возможность подключиться к любой БД кластера, и в pg_hba.conf разрешено подключение от БД, где установлено расширение `ftd_profile`.

Создание необходимых объектов

Выполняется суперпользователем:

```
CREATE SCHEMA file_fdw;
CREATE EXTENSION file_fdw SCHEMA file_fdw;
CREATE SERVER ftd_fileserver FOREIGN DATA WRAPPER file_fdw;
CREATE SCHEMA stat_statements;
CREATE EXTENSION pg_stat_statements SCHEMA stat_statements;
CREATE SCHEMA ftd_profile AUTHORIZATION profile_mon;
```

Имя схемы **ftd_profile** должно совпадать с именем схемы, где установлен `ftd_profile`.

Привилегии для получения статистических данных

```
GRANT USAGE ON FOREIGN  SERVER ftd_fileserver TO profile_mon;
GRANT USAGE ON SCHEMA file_fdw TO profile_mon;
GRANT EXECUTE ON function stat_statements.pg_stat_statements_reset TO profile_mon;
GRANT EXECUTE ON function wait_sampling.pg_wait_sampling_reset_profile TO profile_mon;
GRANT pg_read_all_stats TO profile_mon;
GRANT pg_execute_server_program TO profile_mon;
GRANT pg_read_server_files TO profile_mon;
```

Создание инфраструктуры для сбора метрик ОС на внешнем сервере БД

Для сбора метрик ОС на внешнем сервере требуется создать на нем необходимые таблицы при помощи функции `CREATE_FDW_OBJECTS`.

CREATE_FDW_OBJECTS

Синтаксис функции:

```
create_fdw_objects(server name)
```

Функция создает необходимые объекты на указанном сервере `server`, используя реквизиты подключения, определенные при создании записи. Создание сервера описано в разделе "Управление серверами".

Использование расширения ftd_profile

Управление серверами

При установке расширения `ftd_profile` создается запись локального сервера (`local`). Для сбора статистических данных на внешних серверах необходимо создать их при помощи функции `create_server`.

Функции управления серверами

CREATE_SERVER

Синтаксис функции:

```
create_server(server name, server_connstr text, server_enabled boolean = TRUE, max_sample_age integer = NULL, description
text = NULL)
```

Функция создает запись для внешнего сервера с именем `server` (должно быть уникальным; функции выгрузки данных и получения снимков будут идентифицировать сервер по этому имени), со строкой соединения `server_connstr`. Флаг активности `server_enabled` определяет, будут ли обрабатываться данные на этом сервере; по умолчанию он равен `TRUE`. Параметр `max_sample_age` определяет максимальное время хранения собранных метрик в днях. `description` - описание сервера, будет отображаться в заголовке отчета.

ENABLE_SERVER

Синтаксис функции:

```
enable_server(server name)
```

Функция добавляет сервер с именем `server` в обработку.

DISABLE_SERVER

Синтаксис функции:

```
disable_server(server name)
```

Функция исключает сервер с именем `server` из обработки.

RENAME_SERVER

Синтаксис функции:

```
rename_server(server name, new_name name)
```

Функция переименовывает сервер с именем `server` в `new_name`.

DROP_SERVER

Синтаксис функции:

```
drop_server(server name)
```

Функция удаляет сервер с именем `server` и всю накопленную по нему статистику.

SET_SERVER_MAX_SAMPLE_AGE

Синтаксис функции:

```
set_server_max_sample_age(server name, max_sample_age integer)
```

Функция устанавливает период хранения данных для сервера `server` в `max_sample_age` дней.

SET_SERVER_SIZE_SAMPLING

Синтаксис функции:

```
set_server_size_sampling(server name, window_start time with time zone = NULL, window_duration interval hour to second =
NULL, sample_interval interval day to minute = NULL)
```

Функция `set_server_size_sampling` определяет политику сбора размеров объектов базы данных. Для сбора размеров всех объектов в базе данных может потребоваться

значительное время. `ftd_profile` может пропускать сбор этой метрики. Политика определяется, как ежедневное окно, когда разрешен сбор размеров объектов базы данных и минимальный период между двумя вызовами функции `take_sample`.

Параметры:

- server - имя сервера;
- window_start - время начала окна. Указывается в формате "HH:MM+timezone";
- window_duration - продолжительность окна. Указывается в формате "interval '2 hour'".
- sample_interval - продолжительность перерыва между вызовами take_sample. Например, если указано значение "interval '2 hour'", это означает, что сбор метрик размера объектов базы данных будет не чаще, чем один раз в два часа.

Пример:

```
SELECT set_server_size_sampling('local','3:00+07',interval '2 hour',interval '2 hour');
```

SET_SERVER_DB_EXCLUDE

Синтаксис функции:

```
set_server_db_exclude(server name, exclude_db name[])
```

Функция определяет список `exclude_db[]` необрабатываемых баз для сервера `server`.

SET_SERVER_CONNSTR

Синтаксис функции:

```
set_server_connstr(server name, new_connstr text)
```

Функция меняет строку соединения `new_connstr` для сервера `server`.

SET_SERVER_DESCRIPTION

Синтаксис функции:

```
set_server_description(server name, description text)
```

Функция изменяет описание `description` для сервера `server`.

SHOW_SERVERS

Синтаксис функции:

```
show_servers()
```

Функция отображает список всех серверов.

Пример создания сервера:

```
SELECT ftd_profile.create_server('omega','host=name_or_ip dbname=postgres port=5432 user=profile_mon
password=strong_pwd');
```

В результате выполнения создастся сервер с именем omega и строкой соединения: 'host=name_or_ip dbname=postgres port=5432 user=profile_mon password=strong_pwd'

Управление базовыми границами

Базовая граница — это именованная последовательность снимков, имеющая собственные настройки хранения. Базовую границу можно использовать в функциях построения отчетов в качестве интервала выборки. Неопределенное базовое удержание означает бесконечное удержание. Базовые границы используются для сохранения информации о рабочей нагрузке базы данных за определенный период времени. Например, можно сохранить образцы, собранные во время нагрузочного тестирования или во время обычной загрузки системы, для дальнейшего использования.

CREATE_BASELINE

Синтаксис функции:

```
create_baseline([server name,] baseline_name varchar(25), start_id integer, end_id integer [, days integer])
```

Функция создает базовую границу для сервера `server` (если значение не указано, по умолчанию используется `local`) с названием `baseline_name` для снимков с `start_id` по `end_id`, со временем хранения `days` дней (если параметр не указан, выборка не удаляется).

DROP_BASELINE

Синтаксис функции:

```
drop_baseline([server name,] name varchar(25))
```

Функция удаляет базовую границу для сервера `server` (если значение не указано, по умолчанию используется `local`) с названием `name`.

KEEP_BASELINE

Синтаксис функции:

```
keep_baseline([server name,] name varchar(25) [, days integer])
```

Функция изменяет время хранения для базовой границы с названием `name` для сервера `server` (если значение не указано, по умолчанию используется `local`) на `days` дней (если параметр не указан, выборка не удаляется) с момента `now()`.

SHOW_BASELINES

Синтаксис функции:

```
show_baselines([server name])
```

Функция выводит все базовые границы для сервера `server` (если значение не указано, по умолчанию используется `local`).

Настройка сбора метрик ОС

SET_OS_METRIC

Синтаксис функции:

```
set_os_metric(metric_name text, enabled boolean)
```

Функция, в зависимости от параметра `enabled`, включает/исключает метрику `metric_name` в обработку. Если `metric_name` = NULL, все метрики установятся в значение `enabled`.

SHOW_OS_METRIC

Синтаксис функции:

```
show_os_metric()
```

Функция отображает список метрик операционной системы.

Сбор метрик

TAKE_SAMPLE

Синтаксис функции:

```
take_sample(server name [, skip_sizes boolean])
```

Функция собирает статистику для указанного сервера `server`. Размеры объектов собираются только при значении параметра `skip_sizes`, равном `FALSE`; по умолчанию значение `TRUE`. Также в функцию `take_sample` в качестве первого параметра можно передавать id сервера.

SHOW_SAMPLES

Синтаксис функции:

```
show_samples([server name,] [days integer])
```

Функция выводит список снимков для сервера `server` за последние `days` дней. По умолчанию выводит все снимки для сервера `local`.

DELETE_SAMPLES

Синтаксис функции:

```
delete_samples([server name] [, start_id integer, end_id integer])
```

Функция удаляет снимки с `start_id` по `end_id` для сервера `server`. По умолчанию удаляются все снимки, кроме последнего, для сервера `local`.

Отчеты

Собранные метрики можно выгрузить в HTML-отчет.

GET_REPORT

Синтаксис функции:

```
get_report([server name,] min_sample_id integer, max_sample_id integer)
get_report([server_id integer,] min_sample_id integer, max_sample_id integer)
get_report([server name,] time_range tstzrange)
get_report([server_id integer,] time_range tstzrange)
get_report([server name,] baseline varchar(25))
get_report([server_id integer,] baseline varchar(25))
```

Функция генерирует HTML-код отчета для сервера с именем `server` или с идентификатором `server_id` для данных со снимка `min_sample_id` по `max_sample_id`, либо за период `time_range`, либо для базовой границы `baseline`. Также можно непосредственно сформировать файл отчета на сервере через командную строку:

```
psql -Aqtc "SELECT ftd_profile.get_report('remote',12,14)" -o report_12_14.html
```

GET_REPORT_LATEST

Синтаксис функции:

```
get_report_latest([server name])
```

Функция генерирует HTML-код отчета для сервера `server` для двух последних снимков. Если не указан входной параметр, то предполагается локальный сервер. Также можно непосредственно сформировать файл отчета на сервере через командную строку:

```
psql -Aqtc "SELECT ftd_profile.get_report_latest()" -o report_latest.html
```

Экспорт и импорт данных

Собранные данные можно экспортировать из одного экземпляра `ftd_profile`, а затем

загрузить в другой.

Экспорт

EXPORT_DATA

Синтаксис функции:

```
export_data([server, [min_sample_id integer,] [max_sample_id integer]] [, obfuscate_queries boolean])
```

Функция выводит в табличном виде собранные данные для указанного сервера `server` со снимка `min_sample_id` по `max_sample_id`. Параметр `obfuscate_queries` при значении `TRUE` подменяет текст запроса на хэш MD5. Если опустить все параметры, функция вернет все собранные данные по всем серверам. Результат функции можно выгрузить в файл командой `copy`:

```
postgres=# \copy (select * from export_data()) to 'export.csv'
```

Импорт

Для импорта данных необходимо создать промежуточную таблицу и загрузить в нее данные:

```
postgres=# CREATE TABLE import (section_id bigint, row_data json);
postgres=# \copy import from 'export.csv'
```

После этого надо запустить функцию `import_data`.

IMPORT_DATA

Синтаксис функции:

```
import_data (data regclass)
```

Функция обрабатывает данные, загруженные в таблицу `data`.

Внимание! В локальной базе `ftd_profile` имена серверов могут совпадать с импортируемыми именами. Для этого следует переименовать или удалить локальные сервера в таблице `servers`.

Описание метрик ОС

Метрики ОС собираются через механизм внешних таблиц (FOREIGN TABLE) при помощи расширения `file_fdw` из данных операционной системы и результатов команд.

Название метрики	Источник данных	Описание
CPU usage	Нагрузка на процессор. Данные из /proc/loadavg.	loadavg
Virtual Memory	Состояние виртуальной памяти. Данные из /proc/vmstat.	vmstat
Meminfo	Состояние физической памяти. Данные из /proc/meminfo.	meminfo
Device statistic	Информация об операциях ввода-вывода на блочном уровне. Данные из /sys/block/*/stat/.	devstat
Disk space	Информация об используемом пространстве на дисках. Данные утилиты df.	`man df` *
Page cache hit/miss	Мониторинг системы ввода-вывода, загрузки устройств. Данные из утилиты iostat.	`man iostat` *
Traffic Statistic	Входящий-исходящий траффик. Данные из /proc/net/dev.	netdev
Connections	Соединения. Данные утилиты netstat	`man netstat` *
Top I/O Process	ТОП процессов ввода-вывода. Данные утилиты pidstat.	`man pidstat` *
Process by open files	ТОП процессов по открытым файлам. Данные утилиты lsof.	`man lsof` *
Top Process by CPU load	ТОП процессов по нагрузке процессора. Данные утилиты ps.	`man ps` *
Top Process by RAM load	ТОП процессов по загрузке памяти. Данные утилиты ps.	`man ps` *
Process by Postgres	Процессы, запущенные пользователем postgres. Отчет на основе данных утилиты ps.	`man ps` *

*Примечание: для стандартных системных команд Linux справку можно получить в терминале Linux, используя команду:

```
man command_name
```

Статистика по внутренним метрикам экземпляра FTData

Название метрики	Источник данных	Описание
Subtransactions	Данные по количеству субтранзакций на момент получения снимка.	Смотри раздел "Мониторинг СУБД FTData. Мониторинг статистики" официальной документации СУБД FTData.
Блокировки	Данные по блокировкам на момент получения снимка.	Данные из стандартного представления pg_locks.
Горизонт определения видимости кортежей	Горизонт определения видимости кортежей (`xmin`, `xmax`) на момент получения снимка.	Данные из стандартного представления pg_stat_activity.
Счетчики кортежей	Счетчики количества обработанных кортежей для поиска родительских транзакций на момент получения снимка.	Смотри раздел "Мониторинг СУБД FTData. Мониторинг использования кэша субтранзакций" официальной документации СУБД FTData.

Расширение ftd_trace

Общие сведения

Расширение `ftd_trace` предназначено для отслеживания в реальном времени работы сеансов PostgreSQL.

Для того чтобы воспользоваться расширением, необходимо добавить `ftd_trace` в список `shared_preload_libraries` в конфигурационном файле `postgresql.conf`:

```
shared_preload_libraries = 'ftd_trace'
```

и установить его в FTData командой:

```
CREATE EXTENSION ftd_trace;
```

После этого трассировка может быть запущена для текущей сессии командой:

```
SELECT ftd_trace_start();
```

или для другой сессии с указанием pid ее серверного процесса:

```
SELECT ftd_trace_start(<pid>);
```

где pid серверного процесса можно узнать, выполнив:

```
SELECT pg_backend_pid();
```

или для всех запущенных в данный момент сессий - командой:

```
SELECT ftd_trace_start_all();
```

После того как, трассировка запущена, в каталоге, заданном параметром `log_directory`, будет создан файл с именем, соответствующим маске: ftd_trace_<pid>_<year>-<month>-<day>_<hour><minute><second>.trace, в который будет сохранятся трассировка действий серверного процесса.

Узнать список всех трассируемых в данный момент сессий, а также время начала трассировки и путь к файлу, в который сохраняется трассировка, можно, выполнив:

```
SELECT * from ftd_trace_activity;
```

Пример списка трассируемых сессий:

```
alex=# SELECT * from ftd_trace_activity;
pid  |           start            |                    filename
-------+----------------------------+---------------------------------------------
58662 | 2024-02-02 09:29:24.081212 | log/ftd_trace_58662_2024-02-02_162924.trace
58658 | 2024-02-02 09:29:24.088548 | log/ftd_trace_58658_2024-02-02_162924.trace
58660 | 2024-02-02 09:29:24.086741 | log/ftd_trace_58660_2024-02-02_162924.trace
58656 | 2024-02-02 09:29:24.086695 | log/ftd_trace_58656_2024-02-02_162924.trace
58659 | 2024-02-02 09:29:24.086324 | log/ftd_trace_58659_2024-02-02_162924.trace
58661 | 2024-02-02 09:29:24.086546 | log/ftd_trace_58661_2024-02-02_162924.trace
58657 | 2024-02-02 09:29:24.088435 | log/ftd_trace_58657_2024-02-02_162924.trace
(7 rows)
```

Для завершения трассировки текущей сессии нужно выполнить:

```
SELECT ftd_trace_stop();
```

Для завершения трассировки другой сессии нужно указать pid ее серверного процесса:

```
SELECT ftd_trace_stop(<pid>);
```

Для завершения трассировки всех трассируемых сессий нужно выполнить:

```
SELECT ftd_trace_stop_all();
```

Расширение позволяет отслеживать следующие события:

- поступление SQL-запроса;
- начало и завершение выполнения SQL-запроса (SELECT/INSERT/UPDATE/DELETE);
- начало и завершение транзакции (BEGIN/COMMIT/ROLLBACK);
- начало и завершение DDL операций (CREATE/ALTER/DROP);
- ожидания;
- нахождение сервера в процессе выполнения запроса с выводом количества обработанных строк;
- начало и завершение сервисных операций (checkpoint, vacuum и др.).

После окончания каждой операции выводится статистика использования процессорного времени и всего времени, затраченного на выполнение операции.

Подробность отображаемой информации можно настраивать, используя GUC-переменные:

- `ftd_trace.explain` = on|off - включение/отключение вывода плана выполнения запроса;
- `ftd_trace.waitings` = on|off - включение/отключение вывода ожиданий;
- `ftd_trace.buffers` = on|off - включение/отключение вывода статистики использования буферов;
- `ftd_trace.process_rows_timeout` = <интервал в секундах> - интервал, через который расширение будет трассировать количество обработанных в процессе выполнения запроса строк. По умолчанию: 0 - трассировка строк отключена.

В процессе работы трассировщика, в зависимости от подробности отображаемой информации, может генерироваться файл трассировки значительного размера. В трассировщике предусмотрена возможность ротации файлов трассировки. Ротация происходит, используя те же настройки, что и ротация логов сервера FTData:

- `log_directory` = <путь> - каталог, в котором расширение будет сохранять файлы трассировки;
- `log_rotation_age` = <период времени> - определяет максимальное время жизни файла трассировки;
- `log_rotation_size` = <размер файла> - определяет максимальный размер файла трассировки.

Следует иметь в виду, что само по себе расширение не влияет на производительность сервера FTData, однако включение трассировки снижает производительность серверного процесса на 15-20% (измерение сделано на стандартном тесте pgbench).

Расширение oracle_fdw

Общие сведения

Расширение предназначено для получения доступа к данным из СУБД Oracle Database. Оно позволяет передавать условия WHERE и требуемые колонки, а также поддерживает EXPLAIN.

- Расширение разработано на языках C и SQL.

Для работы расширения требуются библиотеки **Oracle Instantclient Basic**

Установка расширения

Расширение поставляется в составе СУБД FTData.

Установка на дистрибутивы Astra Linux

- Скачать rpm-пакет: https://download.oracle.com/otn_software/linux/instantclient/1923000/oracle-instantclient19.23-basic-19.23.0.0.0-1.x86_64.rpm
- Установить пакет Alien:

```
sudo apt install -y alien
```

- Установить скачанный rpm с библиотеками Oracle:

```
sudo alien -i oracle-instantclient19.23-basic-19.23.0.0.0-1.x86_64.rpm
```

- Установить пакет расширения:

```
sudo apt install -y ./ftdata-17-oracle-fdw*.deb
```

Установка на дистрибутивы Ред ОС

- Скачать rpm-пакет: https://download.oracle.com/otn_software/linux/instantclient/1923000/oracle-instantclient19.23-basic-19.23.0.0.0-1.x86_64.rpm
- Установить скачанный rpm с библиотеками Oracle Instantclient Basic:
- Установить скачанный rpm с библиотеками Oracle Instantclient Basic:

```
sudo yum install -y ./oracle-instantclient19.23-basic-19.23.0.0.0-1.x86_64.rpm
```

- Установка пакета:

```
sudo yum install -y ./ftdata-17-oracle-fdw*.rpm
```

Поддерживаемые версии сервера Oracle Database зависят от используемой версии клиента (см. Oracle Database Client/Server Interoperability Matrix в Oracle Database Support document 207303.1).

Желательно использовать последний Patch Set для клиента и сервера Oracle Database, особенно для версий Oracle Database с прекращённой поддержкой. Для списка ошибок Oracle Database, о которых известно, что они влияют на использование oracle_fdw, см. раздел Проблемы.

Подключение расширения в БД

```
CREATE EXTENSION oracle_fdw;
```

Удаление расширения

При удалении расширения не удаляются данные БД и настройки пользователя postgres.

Отключение расширения в БД

```
DROP EXTENSION oracle_fdw;
```

Удаление расширения на дистрибутивах Astra Linux

```
sudo apt remove -y ftdata-17-oracle-fdw
```

Удаление расширения на дистрибутивах РЕД ОС

```
sudo yum remove -y ftdata-17-oracle-fdw
```

Обновление расширения

Обновление расширения выполняется в два этапа:

- Обновление пакетов расширения;
- Обновление БД после обновления пакетов расширения.

При обновлении расширения изменение определений функций и представлений может привести к повреждению объектов БД (индексы, представления и хранимые процедуры), построенных на основе старых версий функций и представлений расширения, поэтому требуется внести изменения в БД.

При манипуляциях с пакетами расширения файлы БД и настройки не меняются, то есть после обновления пакета клиенты смогут пользоваться функциями расширения той же версии, которая была установлена до обновления.

Обновление пакетов расширения

Для обновления пакетов расширения следует удалить старую версию расширения, а затем установить новую версию.

Обновление расширения на дистрибутивах Astra Linux

Удаление старого пакета и установка нового:

```
sudo apt remove -y ftdata-17-oracle-fdw
sudo apt install -y ./ftdata-17-oracle-fdw-{новая версия расширения}*.deb
```

Обновление расширения на дистрибутивах РЕД ОС

Удаление старого пакета и установка нового:

```
sudo yum remove -y ftdata-17-oracle-fdw
sudo yum install -y ./ftdata-17-oracle-fdw-{новая версия расширения}*.rpm
```

Обновление БД после обновления пакетов расширения

Команда обновления расширения:

```
ALTER EXTENSION oracle_fdw UPDATE;
```

версия расширения, как показывается командой psql `\dx` или системным каталогом `pg_available_extensions`, - это **не** установленная версия oracle_fdw. Чтобы получить версию oracle_fdw, используйте функцию `oracle_diag`.

Пример использования

Это простой пример, как использовать oracle_fdw. Более детализированная информация - в разделах Опции и Использование. Вам следует также прочитать PostgreSQL documentation

on foreign data про команды, упоминаемые там.

Предположим, что вы можете присоединиться, как пользователь `postgres` операционной системы (либо любой другой, который запускает сервер FTData), следующей командой:

```
sqlplus orauser/orapwd@//dbserver.mydomain.com:1521/ORADB
```

Это означает, что клиент Oracle Database и окружение настроены правильно. Предположим, что пакет расширения oracle_fdw уже был установлен. (См. раздел Установка расширения.)

Мы хотим получить доступ к таблице, заданной так:

```
SQL> DESCRIBE oratab
Name                            Null?    Type
------------------------------- -------- ------------
ID                              NOT NULL NUMBER(5)
TEXT                                     VARCHAR2(30)
FLOATING                        NOT NULL NUMBER(7,2)
```

Тогда настроим oracle_fdw от суперпользователя FTData:

```
CREATE EXTENSION oracle_fdw;
CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver '//dbserver.mydomain.com:1521/ORADB');
```

Вы можете использовать другие способы именования локальных соединений; см. описание опции **dbserver** ниже.

Следует использовать суперпользователя только, когда это действительно необходимо, поэтому разрешим обычному пользователю использовать внешний сервер:

```
GRANT USAGE ON FOREIGN SERVER oradb TO pguser;
```

Потом вы можете соединиться с FTData, как `pguser`, и определить:

```
CREATE USER MAPPING FOR pguser SERVER oradb OPTIONS (user 'orauser', password 'orapwd');
```

Вы можете использовать внешнюю аутентификацию для того, чтобы избежать сохранения паролей Oracle Database; см. ниже.

```
CREATE FOREIGN TABLE oratab (
   id        integer           OPTIONS (key 'true')  NOT NULL,
   text      character varying(30),
   floating  double precision  NOT NULL
) SERVER oradb OPTIONS (schema 'ORAUSER', table 'ORATAB');
```

имена таблицы и схемы (схемы - необязательно) должны обычно быть в верхнем регистре.

Теперь вы можете использовать таблицу, как обычную таблицу PostgreSQL.

Описание объектов и функций расширения

```
FUNCTION oracle_fdw_handler() RETURNS fdw_handler
```

Функция - обработчик для создания обёртки сторонних данных.

```
FUNCTION oracle_fdw_validator(text[], oid) RETURNS void
```

Функция проверки процедуры создания обёртки сторонних данных.

```
FOREIGN DATA WRAPPER oracle_fdw
  HANDLER oracle_fdw_handler
  VALIDATOR oracle_fdw_validator
```

Расширение автоматически создаёт обёртку сторонних данных с именем `oracle_fdw`. Этого достаточно для создания сторонних серверов. Вы можете создать дополнительные обёртки сторонних данных Oracle Database, например, если вам нужно задать опцию `nls_lang` (или изменить существующую обёртку `oracle_fdw`, но все изменения будут утеряны после dump/restore).

```
FUNCTION oracle_close_connections() RETURNS void
```

Эта функция применяется со всеми открытыми соединениями Oracle Database в этой сессии. См. раздел Использование для дальнейшего описания.

```
FUNCTION oracle_diag(name DEFAULT NULL) RETURNS text
```

Эта функция используется только для диагностических целей. Она возвращает версии oracle_fdw, сервера FTData и клиента Oracle Database. Если она вызывается без аргументов или с NULL, то она дополнительно вернёт значения некоторых переменных окружения, используемых для создания подключений к Oracle Database. Если она вызывается с именем

стороннего сервера, то она дополнительно вернёт версию сервера Oracle Database.

```
FUNCTION oracle_execute(server name, stmt text) RETURNS void
```

Эта функция инициирует выполнение выражений SQL на удалённом сервере Oracle Database. Она будет работать только с выражениями, которые не возвращают результаты (обычно, это выражения DDL).

Будьте осторожны, когда используете эту функцию, так как она может нарушить организацию транзакций oracle_fdw. Помните, что выполнение выражений DDL в Oracle Database выполнит неявный COMMIT. Рекомендуется использовать эту функцию без транзакций со множеством выражений.

Опции

Опции обёртки сторонних данных

> Если вы измените обёртку сторонних данных `oracle_fdw`, любые эти изменения будут утеряны при dump/restore. Если вы хотите, чтобы новые опции были постоянны, создайте новую обёртку с ними. Тогда созданные SQL скрипты будут содержать команду CREATE FOREIGN DATA WRAPPER, которую вы вызывали.

Если вы хотите, чтобы новые опции были постоянны, создайте новую обёртку с ними. Тогда созданные SQL скрипты будут содержать команду:

nls_lang

необязательное

Устанавливает переменную окружения NLS_LANG для Oracle Database в формате "язык_территория.кодировка" (например, AMERICAN_AMERICA.AL32UTF8 или RUSSIAN_RUSSIA.CL8MSWIN1251). Это должно совпадать с кодировкой вашей базы данных. В случае, если переменная не установлена, то oracle_fdw автоматически использует корректное значение, если сможет, иначе выдаст предупреждение. Задавайте это значение только, если знаете, что делаете. См. раздел Проблемы.

Параметры стороннего сервера

dbserver

обязательный

Строка подключения к удалённой БД Oracle Database. Может быть в любом формате, который поддерживается вашим Oracle Database клиентом. Используйте пустую строку для локальных (`BEQUEATH`) соединений.

isolation_level

необязательное, `serializable` *по умолчанию*

Уровень транзакционной изоляции, который будет использоваться БД Oracle Database. Принимает значения `serializable`, `read_committed` или `read_only`.

Обратите внимание, что таблица Oracle Database может быть запрошена более одного раза в течение одного запроса вложенным циклом PostgreSQL. Для того, чтобы обеспечить согласованность данных, которая может нарушена во время работы конкурирующих транзакций, необходимо, чтобы уровень изоляции транзакции обеспечивал её. Это гарантируется только при уровнях изоляции Oracle Database SERIALIZABLE или READ ONLY. К сожалению, реализация SERIALIZABLE в Oracle Database довольно плоха и вызывает ошибки сериализации (ORA-08177) в неожиданных ситуациях (например, при вставках в таблицу). Использование транзакций READ COMMITTED обходит эту проблему, но **существует риск нарушений целостности**. Если вы хотите использовать READ COMMITTED, то проверьте, что планы запроса будут сканировать внешнюю таблицу только один раз.

nchar

булево, необязательное, по умолчанию `off`

Установка этой опции в `on` выбирает более затратное преобразование символов на стороне Oracle Database. Это требуется, если вы используете однобайтный набор символов БД Oracle Database, но у вас есть колонки NCHAR или NVARCHAR2, содержащие специальные символы, которые не могут быть представлены в кодировке базы данных. Установка `nchar` в `on` имеет заметное влияние на производительность и вызывает ошибки ORA-01461 с выражениями UPDATE, которые задают строки более 2000 байт (или 16383, если и у вас MAX_STRING_SIZE = EXTENDED). Это похоже на проблему Oracle Database.

set_timezone

булево, необязательное, по умолчанию `off`

Установка этой опции в `on` задаёт при соединении с Oracle Database для сессии Oracle Database ту же временную зону, что и у FTData `timezone`. Это полезно только, если вы планируете использовать колонки Oracle Database типа TIMESTAMP WITH LOCAL TIME ZONE и хотите перевести их в `timestamp without time zone` в FTData. Обратите внимание, что если вы измените `timezone` после установки соединения с Oracle Database, oracle_fdw не изменит временную зону сессии Oracle Database. Вы можете вызвать `oracle_close_connections()` в этом случае, и новое соединение будет открыто в следующий раз при доступе к сторонней таблице. Если Oracle Database не определит временную зону, то соединения будут неудачными, с ошибкой вида:

ORA-01882: timezone region not found

В этом случае либо используйте другую `timezone`, либо оставьте опцию в `off` и установите переменную окружения ORA_SDTZ в соответствующее значение в окружении сервера FTData.

Опции пользователя

user

обязательное

Имя пользователя Oracle Database для сессии. Задайте, как пустую строку для **внешней аутентификации**, если вы не хотите сохранять реквизиты для входа Oracle Database в БД FTData (один из простых способов - использовать **внешнее хранилище паролей**).

password

обязательное

Пароль пользователя Oracle Database.

Опции сторонней таблицы

table

обязательное

Имя таблицы Oracle Database. Это имя должно быть написано в точности, как оно хранится в системном каталоге Oracle Database. Обычно оно состоит только из заглавных букв. Чтобы определить стороннюю таблицу, основанную на произвольном запросе Oracle Database, установите эту опцию, заключённую в круглые скобки, то есть:

```
OPTIONS (table '(SELECT col FROM tab WHERE val = ''string'')')
```

При этому нельзя использовать опцию **schema**. INSERT, UPDATE и DELETE будут работать со сторонними таблицами, определёнными простыми запросами; если требуется другое (или нужно избавиться от лишних сообщений об ошибках Oracle Database для более сложных запросов), используйте опцию таблицы **readonly**.

dblink

необязательное

Соединение с БД Oracle Database, через которое происходит доступ к таблице. Это имя должно быть написано в точности, как в системном каталоге Oracle Database. Обычно оно состоит только из заглавных букв.

schema

необязательное

Схема таблицы (или владельца). Полезно для доступа к таблицам, которые не принадлежат подключенному пользователю Oracle Database. Это имя должно быть написано в точности, как в системном каталоге Oracle Database. Обычно оно состоит только из заглавных букв.

max_long

необязательное, по умолчанию "32767"

Максимальная длина любых колонок LONG, LONG RAW и XMLTYPE в таблице Oracle Database. Возможные значения - целые числа от 1 до 1073741823 (максимальный размер `bytea` в FTData). Это количество памяти будет выделено, по меньшей мере, дважды, так что большие значения потребуют много памяти. Если **max_long** меньше, чем длина самого длинного полученного значения, то вы получите сообщение об ошибке `ORA-01406: fetched column value was truncated`.

readonly

необязательное, по умолчанию "false"

INSERT, UPDATE и DELETE разрешены только на таблицах, где эта опция не установлена в yes/on/true.

sample_percent

необязательное, по умолчанию "100"

Эта опция влияет только на обработку ANALYZE и применима для ANALYZE для очень больших таблиц за приемлемое время. Значение должно быть между 0.000001 и 100 и определяет процент блоков таблиц Oracle Database, которые будут случайно выбраны для расчёта статистик таблиц PostgreSQL. Это реализуется предикатом `SAMPLE BLOCK (x)` в Oracle Database.

ANALYZE завершится с ошибкой ORA-00933 для таблиц, определённых запросами Oracle Database, и может завершиться с ошибкой ORA-01446 для таблиц, определённых сложными представлениями Oracle Database.

prefetch

необязательное, по умолчанию "50"

Устанавливает количество строк, которое будет получено единственным возвратным запросом между FTData и Oracle Database во время сканирования сторонней таблицы. Значение должно быть между 1 и 1000. Большие значения могут улучшить производительность, но будут использовать память сервера FTData. Обратите внимание, что предварительной загрузки не будет, если таблица Oracle Database содержит колонки типа `MDSYS.SDO_GEOMETRY`.

lob_prefetch

необязательное, по умолчанию "1048576"

Задаёт количество байт, которое будет предварительно загружено для значений BLOB, CLOB и BFILE. LOB-ы, которые превышают этот размер, потребуют дополнительных двусторонних запросов между FTData и Oracle Database, поэтому установка этого значения больше размера вашего типичного LOB-а положительно скажется на производительности. Если вы выберете большие значения, то на стороне сервера может выделиться больше памяти, но улучшит производительность для больших LOB-ов.

Опции колонок

key

необязательное, по умолчанию "false"

Если установлено в yes/on/true, то соответствующая колонка в сторонней таблице Oracle Database считается колонкой с первичным ключом. Для работы UPDATE и DELETE вы должны установить эту опцию для всех колонок, относящихся к первичному ключу таблицы.

strip_zeros

необязательное, по умолчанию "false"

Если установлено в yes/on/true, символы 0 ASCII будут удалены из строки во время передачи. Такие символы действительны в Oracle Database, но не в PostgreSQL, так что они будут причиной ошибки при чтении с помощью oracle_fdw. Эта опция имеет смысл только для колонок `character`, `character varying` и `text`.

Использование

Разрешения Oracle Database

Пользователь Oracle Database, очевидно, должен иметь право на CREATE SESSION и право читать данные из выбранной таблицы или представления. Обратите внимание, что oracle_fdw во время планирования запроса обращается к таблице Oracle Database, чтобы получить её структуру. Это происходит **до** проверки разрешений на стороннюю таблицу. Следовательно, вы можете получить ошибку Oracle Database, если вы попробуете получить доступ к сторонней таблице, на которую у вас нет разрешений в FTData. Это ожидаемо и не является проблемой безопасности.

Для EXPLAIN VERBOSE пользователь также должен иметь права на SELECT из V$SQL и V$SQL_PLAN.

Соединения

oracle_fdw кэширует соединения Oracle Database, потому что создание новой сессии Oracle Database для каждого запроса затратно. Все соединения автоматически закрываются, когда заканчивается сессия FTData.

Функция `oracle_close_connections()` может быть использована для закрытия всех кэшированных соединений Oracle Database. Это может быть полезно для долговременных сессий, которые не получают доступ к сторонним таблицам постоянно, и вы хотите избежать блокировки ресурсов, нужных для открытия соединения Oracle Database. Вы не можете вызывать эту функцию внутри транзакции, которая изменяет данные в Oracle Database.

Колонки

Когда вы определяете стороннюю таблицу, колонки таблицы Oracle Database соотносятся с колонками PostgreSQL в порядке их определения.

oracle_fdw будет включать только те колонки в запросе Oracle Database, которые действительно нужны запросу PostgreSQL. Таблица PostgreSQL может иметь больше или меньше колонок, чем таблица Oracle Database. Если она имеет больше колонок, и эти колонки используются, то вы получите предупреждение, что будут возвращены значения NULL.

Если вы хотите сделать UPDATE или DELETE, то убедитесь, что опция `key` задана для всех колонок, которые относятся к первичному ключу таблицы. Невыполнение этого требования приведёт к ошибкам.

Типы данных

Вы должны определить колонки PostgreSQL с типами данных, которые oracle_fdw может преобразовать (см. таблицу преобразования ниже). Это ограничение применятся только, если колонка действительно используется, поэтому вы можете определить колонки-заглушки для непреобразованных типов данных, пока вам не требуется доступ к ним (этот способ работает только с SELECT, а не когда изменяются сторонние данные). Если значение в Oracle Database превышает размер колонки PostgreSQL (то есть длину колонки varchar или максимальное целочисленное значение), вы получите ошибку во время выполнения.

Эти преобразования автоматически обрабатываются oracle_fdw:

Тип Oracle Database	Возможные типы PostgreSQL
CHAR	char, varchar, text
NCHAR	char, varchar, text
VARCHAR	char, varchar, text
VARCHAR2	char, varchar, text, json
NVARCHAR2	char, varchar, text
CLOB	char, varchar, text, json
LONG	char, varchar, text
RAW	uuid, bytea
BLOB	bytea
BFILE	bytea (только чтение)
LONG RAW	bytea
NUMBER	numeric, float4, float8, char, varchar, text
NUMBER(n,m) with m⇐0	numeric, float4, float8, int2, int4, int8,
	boolean, char, varchar, text
FLOAT	numeric, float4, float8, char, varchar, text

Тип Oracle Database	Возможные типы PostgreSQL
BINARY_FLOAT	numeric, float4, float8, char, varchar, text
BINARY_DOUBLE	numeric, float4, float8, char, varchar, text
DATE	date, timestamp, timestamptz, char, varchar, text
TIMESTAMP	date, timestamp, timestamptz, char, varchar, text
TIMESTAMP WITH TIME ZONE	date, timestamp, timestamptz, char, varchar, text
TIMESTAMP WITH LOCAL TIME ZONE	date, timestamp, timestamptz, char, varchar, text
INTERVAL YEAR TO MONTH	interval, char, varchar, text
INTERVAL DAY TO SECOND	interval, char, varchar, text
XMLTYPE	xml, char, varchar, text
MDSYS.SDO_GEOMETRY	geometry (см. "поддержка PostGIS" внизу)

Если NUMBER преобразуется в булево, 0 означает `false`, всё остальное - `true`. Вставка или обновление XMLTYPE работает только со значениями, которые не превышают максимальную длину типа данных VARCHAR2 (4000 или 32767, в зависимости от параметра `MAX_STRING_SIZE`). NCLOB в данный момент не поддерживается, потому что Oracle Database не может автоматически преобразовывать его в кодировку клиента.

Если вы хотите преобразовать TIMESTAMP WITH LOCAL TIME ZONE в `timestamp`, необходимо задать опцию `set_timezone` на стороннем сервере. Если вам нужны преобразования, не включающие вышеперечисленные, то определите соответствующее представление в Oracle Database или в PostgreSQL.

Условия WHERE и предикат ORDER BY

FTData будет использовать все применимые части оператора WHERE, как фильтр для сканирования. Запрос Oracle Database, который создаёт oracle_fdw, будет содержать предикат WHERE, соответствующий этим критериям фильтра, при возможности благополучно перевести такие критерии в Oracle Database SQL. Эта особенность, также известная как "перенос (push-down) операторов WHERE", может значительно уменьшить количество строк, полученных от Oracle Database, и может дать возможность оптимизатору Oracle Database выбрать хороший план для доступа к требуемым таблицам.

Если это возможно, то условия сортировки ORDER BY будут тоже перенесены в ORACLE. Обратите внимание, что ни один предикат ORDER BY, сортирующий по строке символов, не будет перенесён, так как порядки сортировок в PostgreSQL и Oracle Database могут различаться.

Чтобы воспользоваться этой оптимизацией, попытайтесь использовать простые условия для сторонней таблицы. Выберите те типы данных колонок PostgreSQL, которые соответствуют типам Oracle Database, потому что в противном случае условия не могут быть перенесены.

Выражения `now()`, `transaction_timestamp()`, `current_timestamp`, `current_date` и `localtimestamp`

будут преобразованы правильно. Вывод EXPLAIN покажет использованный запрос к Oracle Database, так что вы сможете увидеть, какие условия были переведены в Oracle Database и как.

Соединения (Joins) между сторонними таблицами

Начиная с PostgreSQL 9.6, oracle_fdw может переносить соединения в сервер Oracle Database, то есть соединение между двумя сторонними таблицами приведёт к единственному запросу Oracle Database, который выполняет соединение на стороне Oracle Database.

Существует несколько условий, когда это происходит:

- Обе таблицы должны быть определены на одном и том же стороннем сервере.
- Соединения между тремя или более таблицами не будут перенесены.
- Соединение должно быть в выражении SELECT.
- oracle_fdw должен иметь возможность переносить все условия соединения и предикаты WHERE.
- Перекрёстные соединения без условий не будут перенесены (CROSS JOIN).
- Если перенесено соединение, предикаты ORDER BY не будут перенесены.

Для определения лучшей стратегии соединения важно, чтобы статистика для обеих сторонних таблиц была собрана с помощью ANALYZE для PostgreSQL.

Модификация сторонних данных

oracle_fdw поддерживает INSERT, UPDATE и DELETE для сторонних таблиц. Это разрешено по умолчанию и отключается заданием опции таблицы **readonly**.

Для работы UPDATE и DELETE необходимо установить опцию **key** на колонки, принадлежащие первичному ключу. Эти колонки используются для идентификации строки сторонней таблицы, поэтому убедитесь, что опция задана для **всех** колонок, которые относятся к первичному ключу.

Если вы пропустите колонку сторонней таблицы во время INSERT, то эта колонка будет установлена в значение, определённое в предикате DEFAULT для сторонней таблицы FTData (или NULL, если нет предиката DEFAULT). Предикаты DEFAULT в соответствующих колонках Oracle Database не используются. Если сторонняя таблица FTData не включает все колонки таблицы Oracle Database, то операторы DEFAULT Oracle Database будут использованы для колонок, не включенных в определение сторонней таблицы.

Операция RETURNING при INSERT, UPDATE и DELETE поддерживается, кроме колонок Oracle Database с типами LONG и LONG RAW (Oracle Database не поддерживает эти типы данных в предикате RETURNING).

Триггеры на сторонних таблицах поддерживаются, начиная с PostgreSQL 9.4. Триггеры, определённые с AFTER и FOR EACH ROW, требуют, чтобы сторонняя таблица не имела колонок с типами данных Oracle Database LONG или LONG RAW. Это потому, что такие триггеры используют вышеупомянутый предикат RETURNING.

При изменениях сторонних данных падает производительность, когда затронуто много строк. Из-за особенностей обработки сторонних данных каждая строка должна быть обработана индивидуально.

Транзакции перенаправляются в Oracle Database, поэтому BEGIN, COMMIT, ROLLBACK и SAVEPOINT работают, как ожидается. Подготовленные операторы (prepared statement), используемые Oracle Database, не поддерживаются. См. раздел Внутреннее устройство .

Так как oracle_fdw использует сериализованные транзакции по умолчанию, возможно, что выражения, изменяющие данные, могут привести к сбою сериализации:

ORA-08177: can’t serialize access for this transaction

Это может произойти, если конкурентные транзакции изменяют таблицу и - более вероятно - в долго работающих транзакциях. Такие ошибки могут быть идентифицированы по их SQLSTATE (40001). Приложению, использующему oracle_fdw, следует попытаться повторно выполнить операции, остановленные с этой ошибкой.

Можно попробовать использовать другой уровень изоляции транзакций, см. Параметры стороннего сервера.

EXPLAIN

EXPLAIN PostgreSQL покажет запрос, который на самом деле был отправлен Oracle Database. EXPLAIN VERBOSE покажет план выполнения Oracle Database. Это не поддерживается на сервере Oracle Database 9i или старше, см Проблемы.

ANALYZE

Вы можете использовать ANALYZE для сбора статистики по сторонней таблице. Это поддерживается oracle_fdw.

Без статистики у FTData нет способа оценить количество строк для запросов в сторонней таблице, что может привести к выбору плохих планов выполнения запросов.

FTData автоматически **не** будет собирать статистику по сторонним таблицам с помощью демона autovacuum, как он делает для обычных таблиц, так что особенно важно выполнять ANALYZE на сторонних таблицах после создания и в каждом случае, когда удалённая таблица была значительно изменена.

Помните, что анализ сторонней таблицы в Oracle Database приведёт к полному последовательному сканированию таблицы. Вы можете использовать опцию таблицы **sample_percent** для ускорения процесса, используя только часть таблицы Oracle Database.

Поддержка PostGIS

Тип данных `geometry` доступен, только когда установлен PostGIS.

Поддерживаются только типы: POINT, LINE, POLYGON, MULTIPOINT, MULTILINE и MULTIPOLYGON в двух и трёх измерениях. Пустые геометрии PostGIS не поддерживаются,

потому что для них в Oracle Database Spatial нет эквивалента.

Значения NULL для Oracle Database SRID будут преобразованы в '0' и обратно. Для других преобразований между Oracle Database SRID и PostGIS SRID создайте файл `srid.map` в папке FTData `share`. Каждая строка этого файла должна содержать Oracle Database SRID и соответствующий PostGIS SRID, разделённые пробелом. Для хорошей производительности поддерживайте небольшой размер файла.

Поддержка IMPORT FOREIGN SCHEMA

Начиная с PostgreSQL 9.5, IMPORT FOREIGN SCHEMA поддерживается для массового импорта определений всех таблиц в схеме Oracle Database. В дополнение к документации по IMPORT FOREIGN SCHEMA, рассмотрите следующее:

- IMPORT FOREIGN SCHEMA создаст сторонние таблицы для всех объектов, найденных в ALL_TAB_COLUMNS. Это включает таблицы представления и материализованные представления, но не синонимы.
- Вот поддерживаемые опции для IMPORT FOREIGN SCHEMA:
  - **case**: контролирует регистр символов для таблицы и имён колонок во время импорта. Возможные значения:
    - `keep`: оставляет имена, как есть в Oracle Database, обычно в верхнем регистре.
    - `lower`: переводит все имена таблиц и колонок в нижний регистр.
    - `smart`: переводит только имена в верхнем регистре в Oracle Database (это по умолчанию).
  - **collation**: правило сортировки, используемое для регистра символов для значений `lower` и `smart` опции **case**. Значение по умолчанию - `default`, что является правилом сортировки БД по умолчанию. Поддерживаются только правила сортировки в схеме `pg_catalog`. См. значения `collname` в каталоге `pg_collation` для списка возможных значений.
  - **dblink**: соединение с БД Oracle Database, через которое происходит доступ к схеме. Это имя должно быть написано в точности, как оно отображено в системном каталоге Oracle Database; обычно состоит только из заглавных букв.
  - **readonly**: задаёт опцию **readonly** для всех импортированных таблиц. См. Опции.
  - **max_long**: задаёт опцию **max_long** для всех импортированных таблиц. См. Опции.
  - **sample_percent**: задаёт опцию **sample_percent** для всех импортированных таблиц. См. Опции.
  - **prefetch**: задаёт опцию **prefetch** для всех импортированных таблиц. См. Опции.
  - **lob_prefetch**: задаёт опцию **lob_prefetch** для всех импортированных таблиц. См. Опции.
  - **nchar**: задаёт опцию **nchar** для всех импортированных таблиц. См. Опции.
  - **set_timezone**: задаёт опцию **set_timezone** для всех импортированных таблиц. См. Опции.

- Имя схемы Oracle Database должно быть написано в точности, как в Oracle Database, обычно в верхнем регистре. Так как PostgreSQL переводит имена в нижний регистр перед обработкой, вы должны заключить имя схемы в двойные кавычки (например, `"SCOTT"`).
- Имена таблиц в предикатах LIMIT TO или EXCEPT должны быть написаны так, как они выглядят в PostgreSQL после вышеописанного изменения регистра символов.

Обратите внимание, что IMPORT FOREIGN SCHEMA не работает с сервером Oracle Database 8i; см. Проблемы.

Внутреннее устройство

oracle_fdw устанавливает MODULE сессии Oracle Database в `postgres` и ACTION в номер процесса бэкенда. Это может помочь идентифицировать сессию Oracle Database и позволит вам трассировать её с DBMS_MONITOR.SERV_MOD_ACT_TRACE_ENABLE.

oracle_fdw использует интерфейс массивов Oracle Database во избежание ненужных возвратных запросов. Размер массива настраивается опцией таблицы **prefetch** и по умолчанию равен 50.

Кроме использования PLAN_TABLE для планирования выполнения (explain) запроса Oracle Database (который потребует создание такой таблицы в БД Oracle Database), oracle_fdw использует планы выполнения, сохранённые в кэше библиотеки. Для этого запрос Oracle Database **явно описан**, что заставляет Oracle Database разобрать запрос. Трудная часть - это найти SQL_ID и CHILD_NUMBER выражения в V$SQL, потому что колонка SQL_TEXT содержит только первые 1000 байт запроса. Поэтому oracle_fdw добавляет комментарий к запросу, содержащий хэш текста запроса. Это используется для поиска в V$SQL. Действительный план запроса или информация о стоимости получается из V$SQL_PLAN.

oracle_fdw использует уровень изоляции транзакций SERIALIZABLE на стороне Oracle Database, что соответствует REPEATABLE READ в PostgreSQL. Это необходимо потому, что единственное выражение PostgreSQL может привести ко множеству запросов Oracle Database (то есть в течение соединения во вложенном цикле), и все результаты должны быть согласующимися. К сожалению, реализация SERIALIZABLE в Oracle Database имеет определённые особенности; см. раздел Проблемы.

Транзакция Oracle Database фиксируется (commit) сразу после фиксирования локальной транзакции, поэтому завершённая транзакция PostgreSQL гарантирует, что транзакция Oracle Database тоже завершилась. Однако, есть небольшой шанс, что транзакция PostgreSQL не сможет завершиться, даже если транзакция Oracle Database была зафиксирована. Этого нельзя избежать без использования двухфазных транзакций и менеджера транзакций, что за границами возможностей сторонней обёртки данных. Подготовленные выражения Oracle Database не поддерживаются по той же самой причине.

Проблемы

Кодировка

Символы, хранимые в БД Oracle Database, которые не могут быть преобразованы в кодировку БД FTData, будут в фоновом режиме заменены Oracle Database на **заменяющие символы**, обычно на нормальный или перевёрнутый знак вопроса. И вы не получите ни одного предупреждения или сообщения об ошибке.

Если вы используете кодировку БД FTData, о которой не знает Oracle Database(в данный момент, это EUC_CN, EUC_KR, LATIN10, MULE_INTERNAL, WIN874 и SQL_ASCII), не-ASCII символы не могут быть преобразованы правильно. В этом случае вы получите предупреждение, и символы будут заменены на заменяющие символы, как описано выше.

Вы можете задать опцию **nls_lang** для обёртки сторонних данных для того, чтобы заставить Oracle Database использовать определённую кодировку, но результирующие символы, скорее всего, будут неправильными и приведут к ошибкам PostgreSQL. Вероятно, это будет полезно только для кодировки SQL_ASCII, если вы знаете, что делаете. См. раздел Опции.

Ограниченная функциональность в старых версиях Oracle Database

- Определение системных каталогов Oracle Database V$SQL и V$SQL_PLAN изменилось, начиная с Oracle Database 10.1. Использование EXPLAIN VERBOSE с более старыми версиями сервера Oracle Database приведёт к ошибкам вида:

```
ERROR:  error describing query: OCIStmtExecute failed to execute remote query for sql_id
DETAIL:  ORA-00904: "LAST_ACTIVE_TIME": invalid identifier
```

Это не планируется исправлять, так как Oracle Database 9i был исключён из Extended Support с 2010, и функциональность несущественна.

- IMPORT FOREIGN SCHEMA выдаёт следующую ошибку с сервером Oracle Database 8i:

```
ERROR:  error importing foreign schema: OCIStmtExecute failed to execute column query
DETAIL:  ORA-00904: invalid column name
```

Это потому, что представление ALL_TAB_COLUMNS не имеет колонки CHAR_LENGTH, которая была добавлена, начиная с Oracle Database 9i.

Библиотеки LDAP

Разделяемая клиентская библиотека Oracle Database поставляется с собственной реализацией клиента LDAP, в соответствии с RFC 1823, поэтому эти функции имеют те же имена, что и в OpenLDAP. Это приведёт к совпадению имён, так как сервер FTData был сконфигурирован с `--with-ldap`.

Совпадение имён не будет обнаружено, потому что oracle_fdw загружается во время выполнения, но возникнет проблема, если кто-нибудь вызовет функцию LDAP. Обычно сначала загружается OpenLDAP, так что при вызове функции LDAP из Oracle Database

(например, если вы используете для разрешения имён **Активный каталог**), то бэкенд завершится с ошибкой. Это приведёт к сообщениям следующего вида (замечено в Linux) в логе сервера FTData:

```
../../../libraries/libldap/getentry.c:29: ldap_first_entry:
 Assertion `( (ld)->ld_options.ldo_valid == 0x2 )' failed.
```

Расширение может работать, пока не используется какая-либо функциональность LDAP в Oracle Database. На некоторых платформах вы можете принудительно загружать разделяемую клиентскую библиотеку Oracle до запуска сервера FTData (LD_PRELOAD в Linux). Тогда будут использоваться функции LDAP от Oracle. В этом случае Oracle Database сможет использовать функциональность LDAP, но использование LDAP в FTData приведет в ошибке бэкенда. Вы не можете одновременно использовать функциональность LDAP для FTData и для Oracle Database.

Ошибки сериализации

В Oracle Database 11.2 и новее вставка первой строки во вновь созданную таблицу Oracle Database с oracle_fdw приведёт к ошибке сериализации.

Это происходит из-за свойства Oracle Database, называемого **deferred segment creation** (отсроченное создание сегмента), которое откладывает выделение места для хранения новой таблицы - до вставки первой строки. Это вызывает сбой в сериализации с сериализуемыми транзакциями (см. документ 1285464.1 в базе знаний Oracle Database).

Это не является серьёзной проблемой; вы можете обойти это, либо игнорируя эту первую ошибку, либо создавая таблицу с SEGMENT CREATION IMMEDIATE.

Более серьёзная проблема - это то, что конкурентные вставки могут иногда приводить к ошибкам сериализации, когда страница индекса разделяется одновременно с изменяющей сериализуемой транзакцией (см. документ Oracle Database 160593.1).

Oracle Database заявляет, что это не ошибка, и предполагаемое решение - это перезапустить транзакцию, которая вызвала ошибку сериализации.

Ошибки Oracle Database

Вот список ошибок Oracle Database, которые связаны или были связаны с oracle_fdw.

Ошибка 2728408 может вызывать `ORA-8177 cannot serialize access for this transaction`, даже если не было никаких попыток изменить удалённые данные. Она появляется на сервере Oracle Database 8.1.7.4 (установите одноразовый патч 2728408) или на сервере Oracle Database 9.2 (установите Patch Set 9.2.0.4 или более поздний).

Известно, что клиент Oracle Database версии 21c не работает с колонками CLOB (они оказываются пустыми). Нет точного подтверждения, что это проблема Oracle Database, но другие версии работают хорошо.

Расширение pbl

Общие сведения

Расширение `pbl` (от plan baselines) предназначено для кэширования и повторного использования планов запросов в рамках кластера СУБД.

Расширением обрабатываются запросы с SQL-командами: `SELECT`, `UPDATE`, `INSERT`, `DELETE`, `MERGE`.

Кэш расширения хранит исключительно общие (generic) планы (см. Общие и частные планы) для запросов с параметрами. Для запросов без параметров все планы сохраняются.

Запись в кэше для запроса содержит свойства запроса - SQL-текст, статистику запроса (данные по всем планам, которые строились и использовались для запроса). К каждой записи запроса может быть прикреплено несколько записей плана. Запись плана в кэше содержит готовый план (результат работы планировщика), флаг разрешения использования плана, флаги состояния плана, текст EXPLAIN для плана, статистику плана.

Запросы (и планы) в кэше делятся на две категории:

- `user` - запросы, помеченные пользователем. Запросы (но не их планы) из этой категории нельзя удалить из глобального кэша - только перевести в категорию `auto`. Запросы добавляются в кэш вручную (см. PBL_TRACK_QUERY), а планы для них собираются во время работы приложения.
- `auto` - наиболее часто исполняемые запросы, собранные в автоматическом режиме. Запросы (и их планы) из этой категории можно удалить из глобального кэша вручную. Эти планы и запросы собираются в оставшееся от `user`-планов и запросов свободное место. Среди таких планов можно выявить кандидаты на повторное использование (в качестве `user`-запросов). Автоматический сбор планов управляется параметром конфигурации `pbl.auto_collect`.

Повторное использование планов

Повторное использование планов - выполнение ранее выбранного плана для определённого запроса без повторного планирования.

Расширение заменяет запуск планировщика загрузкой плана из кэша по следующему алгоритму:

- Перед запуском планировщика для запроса производится поиск готового плана в кэше расширения.
- Если в кэше найден план, пригодный для использования, планировщик не запускается, используется загруженный из кэша план. План будет использован повторно (загружен из кэша) при следующих условиях:
    - План не помечен как непригодный для повторного использования (см.

Отслеживание зависимостей планов);

- План не помечен как повреждённый;
- Разрешено повторное использование плана:
  - План помечен пользователем как фиксированный (см. описание функций PBL_FIX_PLAN и PBL_UNFIX_PLAN);
  - План собран для `user`-запроса, и включена автоматическая фиксация `user`-планов (`pbl.reuse_unfixed_user = on`);
  - План собран для `auto`-запроса, и включена автоматическая фиксация `auto`-планов (`pbl.reuse_unfixed_auto = on`).

- Если потребовалось запустить планировщик (план не загружен из кэша), но запрос есть в кэше (добавлен пользователем или собран автоматически), то результат работы планировщика будет сохранён в кэш в том случае, если ранее такой план в кэше отсутствовал. Для плана, который присутствовал в кэше, но не был повторно использован, обновится статистика планирований.

Если разрешено повторное использование нескольких планов, то будет выбран план, построенный последним (в представлении PBL_PLANS такой план будет приведён выше остальных). Если план будет повторно построен с тем же `planid`, то вновь построенный план заменит предыдущий и попадёт в начало списка планов запроса.

Неподдерживаемые типы запросов

Кэш `pbl` не сохраняет `auto`-запросы и не собирает планы `user`-запросов:

- Для запросов, в которых есть обращение к временным таблицам (такие запросы имеет смысл сохранять в кэше подготовленных выражений сессии);
- Для запросов, в которых используются объекты с политиками Row Level Security.
- Поддержка запросов PL/pgSQL, выполняемых в анонимных блоках (см. `DO`), ограничена. Запросы, добавленные функцией `PBL_TRACK_QUERY`, могут не совпасть с запросами, которые передаются планировщику интерпретатором PL/pgSQL. Для фиксации таких запросов можно включить автоматический сбор запросов (`SET pbl.auto_collect = on`) для текущей сессии и выполнить запрос (DO-блок).

Ручная фиксация планов

Одно из ключевых применений `pbl` - фиксация отобранных администратором БД планов.

Порядок работы с кэшем для ручной фиксации планов:

- Подключиться к целевой БД от имени пользователя, для которого необходимо настроить запрос.
- Добавить `user`-запрос, используя функцию PBL_TRACK_QUERY.
- Выполнить запрос с различными настройками планировщика или хинтами (см. раздел

Хинты), добиться того, что в кэш pbl сохранён оптимальный план. Структуры собранных планов можно проверить по полю plan в представлении PBL_PLANS.

- Зафиксировать план user-запроса, используя функцию PBL_FIX_PLAN.

Хинты не влияют на идентификацию запроса. Идентификация запроса описана в разделе Идентификаторы запроса.

Кэш подготовленных выражений, как и pbl, хранит общие планы для запросов. Это может привести к некоторым затруднениям при подборе оптимального user-плана для фиксации. Для упрощения работы следует:

- Для запросов с параметрами использовать команду PREPARE и после того, как кэш pbl сохранил очередной вариант плана, удалять запрос из кэша подготовленных выражений командой DEALLOCATE;
- Для запросов без параметров не использовать PREPARE (тогда при каждом планировании запроса новый вариант плана будет попадать в pbl).

После добавления запроса с параметрами в кэш pbl (см. PBL_TRACK_QUERY) все активные сессии будут генерировать только общие планы для данного запроса (вне зависимости от значения параметра plan_cache_mode).

```
--- Подготовка
-- Подключаемся к целевой БД от имени пользователя, для которого необходимо настроить запрос.
SET ROLE myuser;
SELECT pbl_track_query('myquery'::name,..); -- Добавляем запрос в кэш pbl
--
--- Сбор плана 1 без дополнительных настроек
PREPARE qry(..) AS ..; -- Добавляем запрос в кэш подготовленных выражений
EXECUTE qry(..); -- Запускаем планировщик
DEALLOCATE qry; -- Удаляем запрос и план из кэша подготовленных выражений
--
--- Сбор плана 2 с нестандартными настройками планировщика
SET enable_indexscan = off; -- Настраиваем планировщик
PREPARE qry(..) AS ..;
EXECUTE qry(..);
DEALLOCATE qry; -- Удаляем запрос и план из кэша подготовленных выражений
--
--- Сбор плана 3 с хинтом
PREPARE qry(..) AS /*+ */ ..; -- В текст запроса добавлен хинт, pbl_track_query вызывать не нужно
EXECUTE qry(); -- Выполняем запрос с хинтом
DEALLOCATE qry; -- Удаляем запрос и план из кэша подготовленных выражений
--
--- Проверяем варианты планов и фиксируем нужный
SELECT * FROM pbl_plans WHERE qname = 'myquery';
SELECT pbl_fix_plan('myquery'::name,..);
```

Хранение планов и запросов в кэше

Кэш хранит данные в отдельной области разделяемой памяти сервера. Для каждого плана

выделяется отдельная ячейка памяти.

Уникальные признаки плана для выбора ячейки в кэше:

- `dbid` - oid БД, для которой выполнялся запрос (совпадает с идентификатором в `pg_stat_statements`);
- `userid` - oid пользователя, который выполнял запрос (совпадает с идентификатором в `pg_stat_statements`);
- `queryid` - идентификатор запроса, учитывающий структуру дерева запроса (результата разбора SQL). Учитываются типы констант, использованных в запросе, но не их значения. Совпадает с идентификатором в `pg_stat_statements`;
- `constid` - идентификатор (хэш сумма) значений констант в дереве запроса (см. Идентификаторы запроса);
- `structid` - идентификатор структуры запроса, учитывающий структуру текста исходного SQL-запроса (до его разбора). При этом он игнорирует значения и типы констант и параметров запроса, а также явные преобразования типов для констант и параметров запроса. (см. Идентификаторы запроса);
- `planid` - идентификатор структуры плана (см. Идентификатор структуры плана).

Размер области разделяемой памяти сервера, которая используется для хранения данных кэша, настраивается параметром конфигурации `pbl.storage_size`.

Данные кэша(запросы, планы и explain'ы) могут храниться в сжатом виде. Настройки сжатия меняются параметром `pbl.compression`. Предусмотрена поддержка алгоритмов `pglz` и `lz4`.

Идентификаторы запроса

Идентификатор `queryid` рассчитывается как хэш сумма свойств дерева запроса (`Query`). Подробнее о нём можно прочитать в документации на расширение pg_stat_statements.

`queryid` не учитывает значения констант в теле запроса (это сделано для объединения статистики для сходных запросов в `pg_stat_statements`). Поэтому для однозначной идентификации запросов, возвращающих одинаковый набор данных, добавлен идентификатор `constid` - хэш-сумма значений всех констант в запросе.

`structid` имеет следующие свойства: - не отличает константы от параметров в SQL-тексте запроса; - игнорирует явные преобразования типов (typecast) для констант и параметров в исходном SQL-тексте запроса, а также вложенные явные преобразования констант и параметров; - игнорирует форматирование SQL-текста; - учитывает объекты БД (таблицы, представления, функции, операторы и.т.п.) только по их именам, если в имени не указана схема, то два объекта с одинаковым именем неотличимы.

Например:

```
-- Пусть есть две таблицы в разных схемах:
CREATE TABLE sh1.table(col int);
CREATE TABLE sh2.table(col numeric);
-- Тогда все select ниже будут иметь одинаковый structid
SET search_path=sh1;
-- Запрос - образец
SELECT * FROM table WHERE col = 10;
-- structid не отличает константы от параметров в тексте запроса
PREPARE tst(int) SELECT * FROM table WHERE col = $1;
-- structid игнорирует явные преобразования типов (в т.ч. вложенные) для констант
SELECT * FROM table WHERE col = ('10'::numeric)::int;
-- structid игнорирует форматирование SQL-текста
select *
  from table
 where col = 11;
SET search_path=sh2;
-- structid анализирует SQL-текст запроса и не отличает sh1.table от sh2.table
SELECT * FROM table WHERE col = 15;
```

Например, запросы:

```
SELECT * FROM table WHERE col = true;
SELECT * FROM table WHERE col = false;
```

будут иметь одинаковые `queryid` и `structid`, но разные `constid` (и возвращать разные результаты).

Расчёт `queryid` и `constid` контролируется при помощи GUC-параметра `compute_query_id`. Расчёт `structid` контролируется при помощи GUC-параметра `compute_query_struct_id`. При этом, если `compute_query_id` равен `off`, `structid` не будет высчитан.

Идентификатор структуры плана

Для оптимизации обработки и хранения планов, во время выполнения каждой операции планирования вычисляется идентификатор структуры плана. Идентификатор учитывает структуру, но не берёт в расчёт условия выборки и соединения (так как константные выражения в таких условиях разрешены для частных планов и не разрешены для общих). Такой подход позволяет группировать общие и частные планы с одинаковой структурой для одного запроса.

В расчёте идентификатора структуры плана участвуют:

- Тип узла плана;
- Параметры узла (например, направление сканирования для IndexScan);
- Списки параметров и колонок, которые обрабатывает узел;
- Идентификаторы таблиц и индексов, которые использует узел.

Не гарантируется, что не найдётся такое сочетание двух разных запросов с двумя разными планами, идентификаторы структуры плана которых совпадут. Уникальным идентификатором плана в рамках одной БД следует считать сочетание `queryid`, `constid`, `planid`.

Отслеживание зависимостей планов

План запроса может оказаться непригодным для повторного использования в случае, если какой-то из объектов, использованных планом (функция, представление, оператор), будет удалён или изменён. `pbl` и кэш подготовленных выражений защищён от подстановки таких планов двумя механизмами:

- При удалении и создании одноимённого объекта изменится `Oid` объекта, который учитывается при расчёте `queryid`, и кэш будет работать уже с другим запросом, сохранив валидный план для нового объекта.
- При изменении структуры объекта (например, добавление секции в секционированную таблицу) ядро СУБД отправит событие инвалидации всем активным процессам (на такие события подписан локальный кэш подготовленных выражений).

При запуске сервера с включенным расширением запускается отдельный процесс `pbl worker` для мониторинга событий инвалидации объектов кэша системного каталога. При появлении события процесс обходит кэш и помечает все планы, в списке зависимостей которых присутствует инвалидированный объект, как непригодные для исполнения - **инвалидирует план**.

Если после инвалидации плана планировщик сформирует такой же план (с тем же `planid`) повторно, то вместо старого плана в кэш будет добавлен новый план, который будет помечен как пригодный для исполнения.

Отслеживание зависимостей поддерживается для объектов каталога: `pg_opclass`, `pg_collation`, `pg_constraint`, `pg_foreign_server`, `pg_operator`, `pg_opfamily`, `pg_proc`, `pg_class`, `pg_statistic_ext`, `pg_type` (кэш подготовленных выражений отслеживает только `pg_proc`, `pg_class`).

Инвалидация плана возможна при изменении данных статистики (например, при выполнении SQL-команды `ANALYZE`) отношения, которое используется в плане:

- `auto`-план будет инвалидирован;
- `user`-план будет инвалидирован только в том случае, если параметр конфигурации `pbl.inval_on_stats_upd` установлен как `on` (значение по умолчанию `off`).

В случае получения события инвалидации кэш подготовленных выражений инвалидирует план и при следующем планировании обратится за планом в кэш `pbl`, где для `user`-запроса план мог сохраниться (при условии: `pbl.inval_on_stats_upd = off`).

Обмен статистикой запросов между сессиями

Данный механизм работает только для параметризованных запросов, хранящихся в кэше подготовленных выражений сессии в режиме `plan_cache_mode = auto`.

Для всех запросов в кэше сохраняется статистика средней стоимости частных (custom) и общих (generic) планов. Эта статистика может быть использована кэшем, хранящим подготовленные выражения в сессии вместо статистики, собранной локально внутри сессии.

При выборе типа плана (общий или частный), который будет использован кэшем, хранящим подготовленные выражения (локальный кэш сессии), выполняется поиск запроса в кэше расширения `pbl`; если находится подходящая запись, то средняя стоимость частного плана и средняя стоимость общего плана из кэша `pbl` будут использованы для выбора типа плана (вместо аналогичных параметров, хранящихся в кэше сессии).

При каждом запуске планировщика статистика запроса, хранящаяся в кэше `pbl`, обновляется с учётом результатов планирования.

При включённом обмене статистикой вновь созданные сессии сразу будут выбирать общий или частный план, не выполняя предварительные проверки локально, и это даст прирост в производительности за счёт снижения количества планирований запросов.

Обмен статистикой запросов управляется параметром конфигурации `pbl.share_stats`.

Сохранение содержания кэша при перезапуске сервера

При завершении работы сервера СУБД FTData данные из кэша сохраняются на диск, и при следующем запуске сервера данные восстановятся.

Данные сохраняются в виде файла дампа:

```
{корневая директория кластера}/pbl/pbl_plan_cache.dump`
```

Во время работы СУБД данные не копируются на диск.

Управление памятью глобального кэша

Область разделяемой памяти для хранения данных кэша ограничена. Поэтому при выделении памяти для новых данных может возникнуть ситуация, при которой потребуется удалить старые данные.

Алгоритм кэша `pbl` размещает планы в двух отдельных очередях - для пользовательских и автоматически собранных планов. В начале очереди находятся недавно использованные планы, в конце - планы, использованные давно.

Данные в кэше разделены на три уровня приоритетов, для каждого из которых выполняется свой алгоритм освобождения места для новой записи:

- Низкий - такие данные будут удаляться в первую очередь, если потребуется выделить данные для более приоритетных категорий. К ним относятся запросы и планы, собранные автоматически. Место для таких данных будет освобождаться удалением наиболее давно использованных автоматических планов; при этом, если окажется, что автоматически собранный запрос не содержит планов, он также будет удалён.
- Средний - такие данные могут быть удалены только при добавлении запросов и планов, определённых пользователем. К ним относятся планы запросов, определённых пользователем. Место для таких данных будет освобождаться сначала удалением всех автоматически собранных планов и запросов, затем удалением наиболее давно использованных пользовательских планов.
- Высокий - такие данные могут быть удалены только пользователем (см. PBL_RESET). К ним относятся запросы, определённые пользователем. Место для таких данных будет освобождаться так же, как и для планов пользовательских запросов.

Взаимодействие pbl с кэшем подготовленных выражений

Кэш подготовленных выражений

> Кэш подготовленных выражений - это часть PostgreSQL. `pbl` её не модифицирует, но активно пользуется решениями, связанными с идентификацией и инвалидацией запросов.

Кэш подготовленных выражений используется, в первую очередь, для ускорения обработки часто используемых выражений. При использовании кэша в некоторых ситуациях можно ускорить выполнение выражений, пропустив стадию планирования (кэш сохраняет план выражения и повторно его использует). Использование расширенного протокола даёт некоторый выигрыш в производительности, по сравнению с простым протоколом, так как для подготовленного выражения не выполняется разбор и переписывание запроса.

Запросы помещаются в кэш приложением (не автоматически) при использовании:

- SQL-команды `PREPARE`;
- `PARSE`-команды расширенного протокола;
- Динамических команд PL/pgSQL.

> По умолчанию JDBC-драйвер использует расширенный протокол (параметр `preferQueryMode` установлен как `extended`), то есть все запросы кэшируются. JDBC-драйвер может автоматически подготавливать часто используемые запросы без явного участия приложения.

Общие и частные планы

Кэш различает общие и частные планы:

- **Общий (generic) план** - план, при построении которого не участвуют конкретные значения параметров. Такой план не всегда оптимален для конкретных значений параметров, так как неточно оценивает селективность условий, в которых участвуют параметры выражения.
- **Частный (custom) план** - план, при построении которого участвуют конкретные значения параметров. Селективность условий, в которых участвуют параметры выражения, оценивается более точно, и может быть выбран более быстрый путь выполнения запроса. Частные планы строятся для каждой операции выполнения выражения из кэша (при выполнении SQL-команды `EXECUTE` или команды `BIND` расширенного протокола).

> При построении частного плана разрешаются константные выражения с участием параметров (результаты выполнения IMMUTABLE-функций и операторов на их основе). В результате дерево плана преобразуется так, что, в общем случае, его нельзя использовать повторно для других значений параметров. Таким образом, частные планы не имеет смысла хранить в кэше, так как в большинстве случаев их невозможно повторно использовать.

Общий план, построенный один раз, будет использоваться до конца сессии или удаления выражения из кэша (SQL-командой `DEALLOCATE`). Общий план будет построен повторно при выполнении DDL-операций над объектами (таблицами, индексами, процедурами и тому подобными), использованными в плане.

Выбор общего или частного плана

Режим кэширования планов для подготовленных выражений настраивается параметром plan_cache_mode.

Режим auto

Режим по умолчанию; если запрос не найден в кэше `pbl`, то кэш подготовленных выражений выбирает оптимальный способ выполнения для каждого подготовленного выражения (в каждой сессии) по следующему алгоритму:

- За первые четыре выполнения выражения строятся и выполняются частные планы.
- На шестом выполнении строится общий план:
  - Если стоимость общего плана больше средней стоимости ранее построенных частных планов, то будет построен и выполнен частный план.
  - Если стоимость общего плана меньше средней стоимости частных планов, то будет выполнен общий план.
- На седьмом и последующих выполнениях:

- Если стоимость общего плана больше средней стоимости частных планов, то будет построен и выполнен частный план.
- Если стоимость общего плана меньше средней стоимости частных планов, то будет выполнен общий план (частные планы перестанут строиться).

Если запрос будет найден в кэше pbl:

- Для `user`-запроса будет использован общий план, и будет собрана статистика планирований (даже если кэш подготовленных выражений запросил построение частного плана).
- Для `auto`-запроса будет построен частный или общий план (как описано выше для запроса, не попавшего в кэш `pbl`), будет собрана статистика планирований.

Если для запроса из кэша `pbl` выполняются условия повторного использования планов (см. PBL_FIX_PLAN), то для него будет выбран общий план (даже если кэш подготовленных выражений запросил построение частного плана).

При включенном обмене статистикой между сессиями (`pbl.share_stats = on`) для выбора частного или общего плана будет использован описанный выше алгоритм. Но исходные данные - средняя полная стоимость частных планов и полная стоимость общего плана - будут загружены из кэша `pbl`.

Режим force_generic_plan

При первом выполнении выражения строится общий план, и он используется всегда.

Если запрос будет найден в кэше `pbl`, будет собрана статистика планирований.

Обмен статистикой между сессиями (`pbl.share_stats = on`) не влияет на работу кэша подготовленных выражений в этом режиме.

Режим force_custom_plan

Если запрос не найден в кэше `pbl`, то при каждом выполнении выражения строится и выполняется новый частный план.

Если запрос будет найден в кэше pbl:

- Для `user`-запроса будет использован общий план, и будет собрана статистика планирований.
- Для `auto`-запроса будет построен частный план, но будет собрана статистика планирований (без текста EXPLAIN и плана).

Обмен статистикой между сессиями (`pbl.share_stats = on`) не влияет на работу кэша подготовленных выражений в этом режиме.

Параметры конфигурации сервера

Расширение настраивается параметрами конфигурации `postgresql.conf` или SQL-

командами.

Имя параметра	Назначение	Значение по умолчанию	Диапазон
`pbl.enable`	Флаг разрешения на работу кэша. Требуется перезагрузка сервера.	`off`	`on` / `off`
`pbl.storage_size`	Размер кэша в разделяемой памяти в мегабайтах (см. Хранение планов и запросов в кэше). Требуется перезагрузка сервера.	`32`	`1 .. 1024`
`pbl.auto_collect`	Флаг включения автоматического сбора планов. Перезагрузка сервера не требуется. Параметр может быть уставлен отдельно для сессии или БД.	`off`	`on` / `off`
`pbl.reuse_unfixed_auto`	Флаг включения автоматического повторного использования не зафиксированных (см. PBL_FIX_PLAN) планов для `auto`-запросов (см. Повторное использование планов). Перезагрузка сервера не требуется. Параметр может быть уставлен отдельно для сессии или БД.	`off`	`on` / `off`
`pbl.reuse_unfixed_user`	Флаг включения автоматического повторного использования не зафиксированных (см. PBL_FIX_PLAN) планов для `user`-запросов (см. Повторное использование планов). Перезагрузка сервера не требуется. Параметр может быть уставлен отдельно для сессии или БД.	`off`	`on` / `off`
`pbl.share_stats`	Флаг включения обмена статистикой запросов между сессиями (см. Обмен статистикой запросов между сессиями). Перезагрузка сервера не требуется. Параметр может быть уставлен отдельно для сессии или БД.	`off`	`on` / `off`
`pbl.inval_on_stats_upd`	Флаг разрешения инвалидации планов `user`-запросов при обновлении статистики представления (см. Отслеживание зависимостей планов). Перезагрузка сервера не требуется. Параметр может быть уставлен отдельно для сессии или БД только суперпользователем.	`off`	`on` / `off`

Имя параметра	Назначение	Значение по умолчанию	Диапазон
pbl.compression	Сжатие данных кэша.(см. Хранение планов и запросов в кэше) Перезагрузка сервера не требуется.	off	on / off / pglz / lz4

Установка расширения

Для того, чтобы воспользоваться расширением, необходимо добавить pbl в список shared_preload_libraries и настроить расширение в конфигурационном файле postgresql.conf:

```
shared_preload_libraries = 'pbl'
pbl.enable = on
```

и установить расширение SQL-командой:

```
CREATE EXTENSION pbl;
```

Расширение работает на уровне кластера СУБД; при установке его для одной из БД оно будет доступно в любой БД в кластере.

Удаление расширения

Достаточно удалить расширение SQL-командой:

```
DROP EXTENSION pbl;
```

Удалять расширение следует в той же БД, в которой оно было установлено; при этом оно перестанет быть доступным во всех БД кластера.

SQL-интерфейс расширения

Разграничение доступа

Манипуляции с записями планов и запросов доступны только пользователям, от которых выполняется запрос, и пользователям с ролью pg_read_all_stats.

Получить роль pg_read_all_stats можно, выполнив SQL-команду:

```
GRANT pg_read_all_stats TO current_user;
```

Если прав для изменения записи в кэше недостаточно, возникнет ошибка (SQL-код ошибки: 42501). В представлениях поля query и explain будут содержать текст <insufficient privilege> для записей других пользователей (если у текущего пользователя нет роли pg_read_all_stats).

Пользовательские метки запросов

Для упрощения операций с кэшем поддерживается возможность добавить имя для запроса и управлять им, используя это имя вместо сочетания идентификаторов dbid, userid, queryid, constid. При сохранении запроса без имени формируется уникальное имя по шаблону:

```
[AAAA].[BBBB].[CCCCCCCC].[D]
```

где:

- [AAAA] - хэш сумма от строки имени пользователя (четыре символа);
- [BBBB] - хэш сумма от строки имени БД (четыре символа);
- [CCCCCCCC] - хэш сумма от SQL-текста запроса (восемь символов);
- [D] - дополнительный маркер уникальности (один символ).

Функции

PBL_IDENTIFY_QUERY

Синтаксис функции:

```
PBL_IDENTIFY_QUERY(query [, args])
```

Функция выполняет разбор SQL-текста запроса и анализирует дерево запроса.

> Разбор и анализ запроса учитывает значение переменной search_path и выполняется для БД, к которой подключена сессия от имени текущего пользователя. При изменении пользователя, БД или search_path результат разбора может измениться.

> Если запрос query является utility-запросом, для него идентификаторы запроса queryid, constid и structid будут заданы следующим образом:

- если включён расчёт queryid (GUC compute_query_id) - queryid и constid будут иметь значения 2. В таком случае, если включён расчёт structid (GUC compute_query_struct_id), он будет иметь значение 2, иначе - 0;
- если выключен расчёт queryid, то queryid, constid и structid будут иметь значения 0.

Аргументы функции:

- `query` - SQL-текст запроса. Запрос может иметь параметры, в тексте они обозначаются как `$n`, где `n` - номер параметра (начиная с 1). Тип аргумента - `text`.
- `args` - массив типов параметров. Для запроса без параметров - `NULL`. Типы параметров могут быть выведены из текста запроса, когда это возможно. Для запроса с параметрами можно передать пустой или неполный массив (начиная с типа первого параметра). Значение по умолчанию - `NULL`.

Функция возвращает композитный тип `RECORD` с полями:

- `queryid` - идентификатор запроса, учитывающий структуру дерева запроса (результата разбора SQL) без констант, использованных в запросе. Тип поля - `bigint`.
- `constid` - идентификатор значений констант в дереве запроса (результат разбора SQL). Тип поля - `int`.
- `structid` - идентификатор запроса, учитывающий структуру текста исходного SQL-запроса (до его разбора) без учёта значений и типов констант, и параметров, использованных в запросе. Тип поля - `bigint`.
- `args` - массив типов параметров. Для запроса без параметров - `NULL`. Типы параметров могут быть выведены из текста запроса, когда это возможно. Для запроса с параметрами можно передать пустой или неполный массив (начиная с типа первого параметра). Тип поля - `oid[]` или `text[]`.
- `norm_query` - нормализованный текст запроса, в котором однозначно описаны все объекты запроса (указаны схемы для отношений, функций, типов и других объектов). Типы параметров могут быть выведены из текста запроса, когда это возможно. Для запроса с параметрами можно передать пустой или неполный массив (начиная с типа первого параметра). Тип поля - `text`.

Категория изменчивости:

```
VOLATILE
```

Варианты типов аргументов функции:

Типы данных входных параметров	Возвращаемый тип данных
(text, oid[])	record (bigint, int, oid[], text)
(text, text[])	record (bigint, int, oid[], text)

PBL_RESET

Синтаксис функции:

```
PBL_RESET([user_defined])
PBL_RESET(qname [, planid])
PBL_RESET(queryid, constid [, planid [, dbid [, userid ]]])
```

Функция очищает кэш кластера от запросов и их планов. Аргумент функции:

- `user_defined` - Удалить пользовательские запросы (с атрибутом `usr_def = true`).
- `qname` - имя запроса, заданное при добавлении запроса. Тип аргумента - `name`.
- `planid` - идентификатор структуры плана. При передаче значения 0 будет удалена запись запроса и все его планы. Значение по умолчанию - `0`. Тип аргумента - `bigint`.
- `queryid` - идентификатор запроса, учитывающий структуру дерева запроса (результата разбора SQL) без констант, использованных в запросе. Тип аргумента - `bigint`.
- `constid` - идентификатор значений констант в дереве запроса (результат разбора SQL). Тип аргумента - `int`.
- `dbid` - oid БД, для которой будет выполнен запрос. Значение по умолчанию - oid БД, к которой подключен клиент.
- `userid` - oid пользователя, который будет выполнять запрос. Значение по умолчанию - oid текущего пользователя.

Категория изменчивости:

```
VOLATILE
```

Варианты типов аргументов функции:

Типы данных входных параметров	Возвращаемый тип данных
(bool)	void
(bigint, int, bigint, oid, oid)	void
(name, bigint)	void

PBL_TRACK_QUERY

Синтаксис функции:

```
PBL_TRACK_QUERY(qname, query [, args [, dbid [, userid]]])
PBL_TRACK_QUERY(query [, args [, dbid [, userid]]])
```

Категория изменчивости:

```
VOLATILE
```

Функция добавляет пользовательский запрос в кэш планов кластера.

Если такой запрос был собран автоматически, он переводится в категорию `user`. Добавленный запрос будет иметь атрибут `usr_def = true`, для него будут собираться варианты планов.

При добавлении запроса, не пригодного для кэширования (см. раздел Неподдерживаемые типы запросов), будет выдано предупреждение; планы для такого запроса собираться не будут.

После добавления запроса с параметрами в кэш `pbl` (см. PBL_TRACK_QUERY) все активные сессии будут генерировать только общие планы для данного запроса (вне зависимости от значения параметра `plan_cache_mode`).

Если в кэше `pbl` уже присутствует указанный запрос, и передан аргумент `qname`, то запрос будет переименован. В лог клиента будет передано сообщение (с уровнем `NOTICE`) с информацией о старом имени запроса и его ключе (`userid`, `dbid`, `queryid`, `constid`).

Аргументы функции:

- `qname` - имя запроса, определяемое пользователем, должно быть уникально. Используется для упрощённой идентификации запроса. Если имя запроса не указано, будет сгенерировано уникальное имя на основе строки имени БД, имени пользователя и SQL-текста запроса. Тип аргумента - `name`.
- `query` - SQL-текст запроса. Запрос может иметь параметры, в тексте они обозначаются как `$n`, где `n` - номер параметра (начиная с 1). Тип аргумента - `text`.
- `args` - массив типов параметров. Для запроса без параметров - `NULL`. Типы параметров могут быть выведены из текста запроса, когда это возможно. Для запроса с параметрами можно передать пустой или неполный массив (начиная с типа первого параметра). Значение по умолчанию - `NULL::oid[]`. Тип аргумента - `oid[]` или `text[]`.
- `dbid` - oid БД, для которой будет выполнен запрос. Значение по умолчанию - oid БД, к которой подключен клиент.
- `userid` - oid пользователя, который будет выполнять запрос. Значение по умолчанию - oid текущего пользователя.

Варианты типов аргументов функции:

Типы данных входных параметров	Возвращаемый тип данных
(name, text, oid[], oid, oid)	void
(name, text, text[], oid, oid)	void
(text, oid[], oid, oid)	void
(text, text[], oid, oid)	void

PBL_UNTRACK_QUERY

Синтаксис функции:

```
PBL_UNTRACK_QUERY(qname [, planid])
PBL_UNTRACK_QUERY(queryid, constid [, planid [, dbid [, userid ]]])
```

Категория изменчивости:

```
VOLATILE
```

Функция переводит запрос из категории `user` в категорию `auto`. Запрос будет иметь атрибут `usr_def = false`.

Если в кэше `pbl` отсутствует указанный запрос, и передан аргумент `qname`, то запрос будет переименован. В лог клиента будет передано сообщение (с уровнем `NOTICE`) с информацией о старом имени запроса и его ключе (`userid`, `dbid`, `queryid`, `constid`).

Аргументы функции:

- `qname` - имя запроса, заданное при добавлении запроса. Тип аргумента - `name`.
- `queryid` - идентификатор запроса, учитывающий структуру дерева запроса (результата разбора SQL) без констант, использованных в запросе. Тип аргумента - `bigint`.
- `constid` - идентификатор значений констант в дереве запроса (результат разбора SQL). Тип аргумента - `int`.
- `planid` - идентификатор структуры плана. При передаче значения 0 все планы запроса будут зафиксированы. Значение по умолчанию - `0`. Тип аргумента - `bigint`.
- `dbid` - oid БД, для которой будет выполнен запрос. Значение по умолчанию - oid БД, к которой подключен клиент.
- `userid` - oid пользователя, который будет выполнять запрос. Значение по умолчанию - oid текущего пользователя.

Варианты типов аргументов функции:

Типы данных входных параметров	Возвращаемый тип данных
(bigint, int, bigint, oid, oid)	void
(name, bigint)	void

PBL_FIX_PLAN

Синтаксис функции:

```
PBL_FIX_PLAN(qname [, planid])
PBL_FIX_PLAN(queryid, constid [, planid [, dbid [, userid ]]])
```

Функция фиксирует план для запроса. `auto`-запрос будет переведён в категорию `user`. План будет иметь атрибут `fixed = true`. При следующем запуске планировщика для запроса будет загружен фиксированный план (планирование запроса не будет выполняться).

Аргументы функции совпадают с аргументами PBL_UNTRACK_QUERY.

Категория изменчивости:

```
VOLATILE
```

Варианты типов аргументов функции:

Типы данных входных параметров	Возвращаемый тип данных
(bigint, int, bigint, oid, oid)	void
(name, bigint)	void

PBL_UNFIX_PLAN

Синтаксис функции:

```
PBL_UNFIX_PLAN(qname [, planid])
PBL_UNFIX_PLAN(queryid, constid [, planid [, dbid [, userid ]]])
```

Функция отменяет фиксацию плана для запроса. План будет иметь атрибут `fixed = false`.

Аргументы функции совпадают с аргументами PBL_UNTRACK_QUERY.

Категория изменчивости:

```
VOLATILE
```

Варианты типов аргументов функции:

Типы данных входных параметров	Возвращаемый тип данных
(bigint, int, bigint, oid, oid)	void
(name, bigint)	void

PBL_TYPE_NAMES_TO_OID

Синтаксис функции:

```
PBL_TYPE_NAMES_TO_OID(names)
```

Категория изменчивости:

```
IMMUTABLE
```

Функция преобразует массив значений типа `text`, содержащих имена типов (описанных в `pg_type`), в массив значений типа `oid` соответствующих типов.

Типы данных входных параметров	Возвращаемый тип данных
(text[])	oid[]

Представления

PBL_QUERIES

Представление содержит данные о запросах (пользовательских и собранных автоматически) и разделяемую статистику запросов для локальных кэшей планов сессий. Это представление содержит отдельные строки для каждой комбинации идентификатора пользователя, идентификатора базы данных, идентификатора запроса, идентификатора значений констант запроса.

Формат строк представления:

Имя колонки	Тип данных	Описание
`userid`	`oid`	OID пользователя, выполнявшего запрос.
`dbid`	`oid`	OID базы данных, в которой выполнялся запрос.
`queryid`	`bigint`	Идентификатор запроса - хеш-код, вычисленный по дереву запроса (Query).
`constid`	`int`	Идентификатор значений констант запроса - хеш-код, вычисленный по дереву запроса (Query).
`structid`	`bigint`	Идентификатор структуры запроса - хеш-код, вычисленный по тексту исходного SQL-запроса (до его разбора) без учёта значений и типов констант и параметров в запросе.
`usr_def`	`boolean`	Флаг запроса, определённого пользователем. `true` - если запрос добавлен пользователем (при вызове функций PBL_TRACK_QUERY или PBL_FIX_PLAN).
`qname`	`name`	Пользовательская метка запроса (см. Пользовательские метки запросов).
`total`	`int2`	Количество планов для запроса, хранящихся в кэше.
`fixed`	`int2`	Количество фиксированных планов для запроса.
`valid`	`int2`	Количество планов, которые могут быть выполнены.
`error`	`int2`	Количество планов, которые не могут быть выполнены из-за ошибки в плане.
`gen_plans`	`bigint`	Количество операций построения общих (generic) планов для этого запроса (для всех его планов, в том числе вытесненных из кэша).
`gen_loads`	`bigint`	Количество операций подстановки общих (generic) планов для этого запроса (для всех его планов, в том числе вытесненных из кэша).

Имя колонки	Тип данных	Описание
gen_mean_total_cost	float8	Средняя стоимость выполнения планов (total cost), рассчитанная при построении общих (generic) планов для этого запроса (для всех его планов, в том числе вытесненных из кэша).
cst_plans	bigint	Количество операций построения частных (custom) планов для этого запроса (для всех его планов, в том числе вытесненных из кэша).
cst_mean_total_cost	float8	Средняя стоимость выполнения планов (total cost), рассчитанная при построении частных (custom) планов для этого запроса (для всех его планов, в том числе вытесненных из кэша).
args	oid[]	Массив типов параметров запроса. Для запроса без параметров - NULL.
query	text	SQL-текст запроса.
mem_usage	int	Объем памяти, занимаемый информацией о запросе в кэше (в разделяемой памяти), в байтах. (без учёта памяти, занимаемой планами запроса)

PBL_PLANS

Представление содержит данные о собранных планах для запросов, представленных в PBL_QUERIES. Это представление содержит отдельные строки для каждой комбинации идентификатора пользователя, идентификатора базы данных, идентификатора запроса, идентификатора значений констант запроса и идентификатора структуры плана (один запрос может иметь несколько вариантов значений констант и / или планов).

Планы для одного запроса перечислены в том же порядке, в котором они попали в кэш pbl. Если план будет повторно построен с тем же planid, то вновь построенный план заменит предыдущий и попадёт в начало списка планов запроса.

Формат строк представления:

Имя колонки	Тип данных	Описание
userid	oid	OID пользователя, выполнявшего запрос.
dbid	oid	OID базы данных, в которой выполнялся запрос.
queryid	bigint	Идентификатор запроса - хеш-код, вычисленный по дереву запроса (Query).
constid	int	Идентификатор значений констант запроса - хеш-код, вычисленный по дереву запроса (Query).

Имя колонки	Тип данных	Описание
structid	bigint	Идентификатор структуры запроса - хеш-код, вычисленный по тексту исходного SQL-запроса (до его разбора) без учёта значений и типов констант и параметров в запросе.
planid	bigint	Идентификатор запроса структуры плана - хеш-код, вычисленный по дереву плана запроса (см. Идентификатор структуры плана).
usr_def	boolean	Флаг запроса, определённого пользователем. true - если запрос добавлен пользователем (при вызове функций PBL_TRACK_QUERY или PBL_FIX_PLAN).
qname	name	Пользовательская метка запроса (см. Пользовательские метки запросов).
fixed	boolean	Флаг фиксации плана. true - если план зафиксирован (при вызове функции PBL_FIX_PLAN).
valid	boolean	Флаг валидности плана. true - если план может быть выполнен (все объекты, от которых зависит план, доступны).
error	boolean	Флаг ошибки плана. true - если план не может быть использован.
gen_plans	bigint	Количество операций построения общего (generic) плана для этой структуры плана.
gen_loads	bigint	Количество операций подстановки общего (generic) плана из кэша для этой структуры плана.
gen_mean_total_cost	float8	Средняя стоимость выполнения плана (total cost), рассчитанная при построении общего (generic) плана для этой структуры плана.
cst_plans	bigint	Количество операций построения частного (custom) плана для этой структуры плана.
cst_mean_total_cost	float8	Средняя стоимость выполнения плана (total cost), рассчитанная при построении частного (custom) плана для этой структуры плана.
args	oid[]	Массив типов параметров запроса. Для запроса без параметров NULL.
query	text	SQL-текст запроса.
explain	text	План запроса в формате, аналогичном выдаче команды EXPLAIN.

Имя колонки	Тип данных	Описание
`dependency`	`json`	Список зависимостей плана в виде JSON-массива объектов. В каждом объекте два поля: `cat` - окончание имени представления `pg_ctalog`, `oid` - идентификатор объекта в этом представлении. Например, для таблицы с oid 17000 JSON-элемент массива будет выглядеть так: `{ "cat": "class", "oid": 17000 }`
`plan_l`	`int`	Объем памяти, занимаемый текстом плана в кэше (в разделяемой памяти), в байтах.
`mem_usage`	`int`	Объем памяти, занимаемый информацией в кэше (в разделяемой памяти), в байтах.

PBL_MEM_USAGE

Представление содержит данные об использовании кэшем разделяемой памяти.

Имя колонки	Тип данных	Описание
`usage`	`bigint`	Размер памяти, занимаемый использованными страницами глобального кэша (кратен размеру страницы - 8192 байта), в байтах.
`usage_max`	`bigint`	Максимальный размер памяти, занимаемый использованными страницами глобального кэша (кратен размеру страницы - 8192 байта), в байтах.
`total`	`bigint`	Полный размер глобального кэша (кратен размеру страницы - 8192 байта), в байтах.
`allocs`	`bigint`	Счётчик количества выделенных для хранения статистики (для одной структуры плана одного запроса) участков памяти.
`envicted`	`bigint`	Счётчик количества удалённых для повторного использования участков памяти.

По соотношению `usage_max` и `total` можно оценить, насколько точно подобран параметр `pbl.storage_size`. Чем ближе `usage_max` к `total`, тем рациональнее используется разделяемая память.

Если `usage_max` составляет 97% и более от `total`, следует рассмотреть возможность увеличения `pbl.storage_size`. Если `envicted` растёт в течение нескольких часов с относительной скоростью (разы в единицу времени), сопоставимой с `allocs`, следует увеличить `pbl.storage_size` (не все варианты планов помещаются в глобальный кэш).

Ограничение количества используемых процессорных ядер

Параметр `cpu_core_limit` предназначен для ограничения и отслеживания в реальном времени количества и списка ядер cpu, на которых работает СУБД FTData.

Количество ядер cpu устанавливается строго на основе лицензии и не может быть изменено пользователем.

Список ядер cpu, на которых работает СУБД, может быть установлен пользователем. Для этого используется GUC-параметр `ftd_core_list`, в котором через запятую перечисляются номера ядер cpu.

Примеры:

- `ftd_core_list` = '0, 1' - все процессы СУБД запускаются на ядрах 0, 1;
- `ftd_core_list` = '0, 1, 4, 8' - все процессы СУБД запускаются на ядрах 0, 1, 4, 8.

Следует иметь в виду следующие особые случаи:

- если общее количество ядер в значении GUC-параметра `ftd_core_list` превышает физическое количество ядер cpu или количество ядер, установленное лицензией, то из списка берутся первые n ядер, в соответствии с ограничением;
- если общее количество ядер в значении GUC-параметра `ftd_core_list` меньше физического количество ядер cpu или количества ядер, установленного лицензией, к списку добавляются недостающие ядра, начиная с первого по порядку, не занятого текущим экземпляром СУБД FTData;
- если GUC параметр `ftd_core_list` отсутствует или содержит пустой список, СУБД использует то количество ядер cpu, которое определено в лицензии или ограничено количеством физических ядер cpu, начиная с первого по порядку, не занятого текущим экземпляром СУБД FTData.

Изменения в выпусках

Версия 17.5.3

- СУБД FTData:
  - Работа с лицензией СУБД FTData
    - Добавлена поддержка проверки пользовательских лицензий.
    - Добавлена возможность установки лимита и списка ядер СУБД FTData.
    - Добавлена возможность работы СУБД FTData в ознакомительном (`Trial`) режиме.

Версия 17.5.2

- СУБД FTData:
  - Первый выпуск СУБД FTData на основе PostgreSQL 17.5.
  - Автоматическое интервальное секционирование:
    - Добавлена поддержка типов `NUMERIC`, `SMALLINT` (`INT2`) и `BIGINT` (`INT8`) для ключей автоматического интервального секционирования.
    - Изменено правило именования автоматически создаваемых секций.
    - Добавлена поддержка автоматического создания секций пользователем, не владельцем секционированной таблицы.
    - Добавлена возможность автоматического создания секций сокращённого размера для избежания конфликта с существующими секциями.
  - Добавлен параметр конфигурации `coredump_store_shared_buffers`. Позволяет исключить `shared_buffers` из файла дампа памяти, генерируемого при сбое.
- Расширение pbl:
  - Добавлена поддержка вычисления structid.
  - Добавлена поддержка сжатия данных кэша алгоритмами `pglz` и `lz4`.
  - Добавлена поддержка кодировки клиента для полей: query, explain и plan.

Версия 17.2.2

- Расширение pbl:
  - Добавлена поддержка кодировки клиента для полей: query, explain и plan.
  - Добавлена поддержка вычисления structid.
  - Добавлена поддержка сжатия данных кэша алгоритмами `pglz` и `lz4`.

Версия 17.2.1

- СУБД FTData:
  - Первый выпуск СУБД FTData на основе ядра PostgreSQL 17.2, соответствует версии 16.6.3 с доработкой:
    - Добавлена функция сжатия данных таблиц и индексов.