

Google Gemini

◆ gemini.google.com/app/70ff5bf15ab31c28



Postgres, как узнать все объекты уровня кластера?

В PostgreSQL, объекты на уровне кластера (или глобальные объекты) — это те, которые существуют вне контекста конкретной базы данных и доступны из любой базы данных в кластере.

Вот основные типы объектов уровня кластера и способы их просмотра:

1. Роли (Roles/Users): Роли в PostgreSQL — это комбинация пользователей и групп. Они определяются на уровне кластера и не привязаны к конкретной базе данных.

Как посмотреть:

- В `psql: \du` или `\du+` (для более подробной информации).
- SQL-запрос:

```
SELECT rolname FROM pg_roles;
```

Или для более полной информации:

```
SELECT * FROM pg_roles;
```

2. Табличные пространства (Tablespaces): Табличные пространства позволяют администраторам определять местоположения в файловой системе, где будут храниться файлы данных объектов баз данных (таблиц, индексов и т.д.). Они также являются объектами уровня кластера.

Как посмотреть:

- В `psql: \db` или `\db+` (для более подробной информации).
- SQL-запрос:

```
SELECT spcname FROM pg_tablespace;
```

Или для более полной информации:

```
SELECT * FROM pg_tablespace;
```

3. Базы данных (Databases): Сами базы данных в кластере являются объектами уровня кластера.

Как посмотреть:

- В `psql: \l` или `\l+` (для более подробной информации).
- SQL-запрос:

```
SELECT datname FROM pg_database;
```

Или для более полной информации:

```
SELECT * FROM pg_database;
```

4. Конфигурационные параметры (Configuration Parameters): Многие глобальные параметры конфигурации PostgreSQL (например, `shared_buffers`, `work_mem` и т.д.) также управляются на уровне кластера.

Как посмотреть:

- В `psql: SHOW ALL;`
- SQL-запрос:

```
SELECT name, setting, unit, category, short_desc FROM pg_settings;
```

Вы также можете просмотреть содержимое файла `postgresql.conf`, который обычно находится в каталоге данных кластера.

5. Расширения (Extensions) - Некоторые аспекты: Хотя расширения устанавливаются в конкретной базе данных, некоторые их компоненты или функции могут иметь глобальное влияние или требования на уровне кластера (например, требование определенной версии PostgreSQL).

6. Утилиты для полного дампа кластера: Утилита `pg_dumpall` используется для создания полного дампа всего кластера PostgreSQL. Она включает в себя глобальные объекты (роли, табличные пространства) и дампы каждой базы данных. Это подтверждает, что эти объекты являются "кластерными".

Важно: При подключении к PostgreSQL вы всегда подключаетесь к конкретной базе данных. Объекты уровня кластера видны из любой базы данных, к которой вы подключены. Мне нужен список всех Postgres-объектов уровня кластера.

Список всех объектов уровня кластера в PostgreSQL:

1. Роли (Roles):

- **Описание:** Определяют пользователей и группы, управляющих доступом к данным и привилегиями. Они глобальны для всего кластера и не привязаны к конкретной базе данных.
- **Системный каталог:** `pg_roles`
- **Как посмотреть:**
 - SQL: `SELECT rolname, rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcanlogin, rolreplication FROM pg_roles;`
 - `psql: \du` или `\du+`

2. Табличные пространства (Tablespaces):

- **Описание:** Позволяют администратору определять местоположение в файловой системе, где будут храниться файлы данных объектов баз данных (таблиц, индексов и т.д.). Они также глобальны для кластера.
- **Системный каталог:** `pg_tablespace`
- **Как посмотреть:**
 - SQL: `SELECT spcname, pg_catalog.pg_tablespace_location(oid) AS location FROM pg_tablespace;`
 - `psql: \db` или `\db+`

3. Базы данных (Databases):

- **Описание:** Хотя каждая база данных содержит свои собственные схемы, таблицы и другие объекты, сама база данных как сущность является объектом уровня кластера.
- **Системный каталог:** `pg_database`
- **Как посмотреть:**
 - SQL: `SELECT datname, datistemplate, datallowconn, datconnlimit, datlastsysoid, datfrozenxid, datminmxid, dattablespace FROM pg_database;`
 - `psql: \l` или `\l+`

4. Конфигурационные параметры (Configuration Parameters):

- **Описание:** Общие параметры конфигурации PostgreSQL, которые применяются ко всему кластеру (например, `shared_buffers`, `max_connections`, `log_destination` и т.д.). Эти параметры обычно настраиваются в файле `postgresql.conf`.
- **Системный каталог:** `pg_settings`
- **Как посмотреть:**
 - SQL: `SELECT name, setting, unit, category, short_desc FROM pg_settings;`
 - `psql: SHOW ALL;`

5. WAL-сегменты (Write-Ahead Log Segments):

- **Описание:** Файлы журнала предзаписи, которые используются для обеспечения целостности данных и восстановления после сбоев. Управляются на уровне кластера.
- **Как посмотреть:**
 - Путем прямого просмотра каталога `pg_wal` (или `pg_xlog` в более старых версиях) внутри каталога данных кластера. Для этого потребуются права доступа к файловой системе.
 - Функции, такие как `pg_current_wal_lsn()` или `pg_walfile_name(lsn)` могут дать информацию о текущем состоянии WAL.

6. Статистика на уровне кластера (Cluster-level Statistics):

- **Описание:** Некоторые представления статистики собирают данные для всего кластера, а не для отдельной базы данных (например, общую активность базы данных).
- **Системные представления:**
 - `pg_stat_database`: Статистика использования по каждой базе данных.
 - `pg_stat_replication`: Статистика репликации (если настроена).
- **Как посмотреть:**
 - SQL: `SELECT * FROM pg_stat_database;`
 - SQL: `SELECT * FROM pg_stat_replication;`

7. Файлы управления кластером:

- **Описание:** Различные внутренние файлы и каталоги в каталоге данных кластера (например, `pg_control`, `postgresql.auto.conf`), которые содержат критически важную информацию о состоянии кластера.
- **Как посмотреть:** Только путем прямого доступа к файловой системе сервера PostgreSQL.

Этот список охватывает основные категории объектов, которые существуют и управляются на уровне всего экземпляра PostgreSQL (кластера), а не только в рамках отдельной базы данных.

Этот список охватывает основные категории объектов - нужен именно полный список

Полный список объектов уровня кластера в PostgreSQL можно получить, если учесть не только "видимые" объекты, но и внутренние системные объекты, каталоги и компоненты, которые управляются на уровне всего экземпляра PostgreSQL.

Вот более полный список:

1. Основные пользовательские объекты уровня кластера:

- **Роли (Roles / Users):**
 - **Описание:** Определяют пользователей и группы, управляющие доступом к данным и привилегиями. Они глобальны для всего кластера и не привязаны к конкретной базе данных.
 - **Системный каталог:** `pg_roles` (ранее `pg_authid`)
 - **Как посмотреть:** `SELECT * FROM pg_roles;` или `\du` в `psql`.
- **Табличные пространства (Tablespaces):**
 - **Описание:** Определяют местоположения в файловой системе для хранения файлов данных объектов баз данных. Глобальны для кластера.
 - **Системный каталог:** `pg_tablespace`
 - **Как посмотреть:** `SELECT * FROM pg_tablespace;` или `\db` в `psql`.
- **Базы данных (Databases):**
 - **Описание:** Сами базы данных, включая системные (`template0`, `template1`, `postgres`), являются объектами уровня кластера.
 - **Системный каталог:** `pg_database`
 - **Как посмотреть:** `SELECT * FROM pg_database;` или `\l` в `psql`.

2. Конфигурация и управление кластером:

- **Конфигурационные параметры (Configuration Parameters):**
 - **Описание:** Все глобальные параметры конфигурации PostgreSQL (например, `shared_buffers`, `max_connections`, `port`, `listen_addresses`, `log_destination` и т.д.), которые применяются ко всему кластеру. Многие из них настраиваются в `postgresql.conf` и `postgresql.auto.conf`.
 - **Системное представление:** `pg_settings`
 - **Как посмотреть:** `SELECT * FROM pg_settings;` или `SHOW ALL;` в `psql`.
- **Файлы аутентификации (Client Authentication Files):**
 - **Описание:** `pg_hba.conf` (Host-Based Authentication) и `pg_ident.conf` (User Name Mapping) определяют, как клиенты могут подключаться к кластеру. Эти файлы глобальны.
 - **Как посмотреть:** Прямой просмотр файлов в каталоге данных кластера (PGDATA).
- **Статистика на уровне кластера (Cluster-level Statistics):**
 - **Описание:** Некоторые представления статистики собирают данные для всего кластера, а не для отдельной базы данных.
 - **Системные представления:**
 - `pg_stat_database`: Статистика использования по каждой базе данных.
 - `pg_stat_replication`: Статистика репликации (если настроена).
 - `pg_stat_activity`: Информация о текущей активности всех сессий в кластере.
 - `pg_stat_wal`: Статистика WAL.
 - **Как посмотреть:** `SELECT * FROM pg_stat_database;`, `SELECT * FROM pg_stat_replication;` и т.д.

- **Настройки ролей для баз данных (Role Database Settings):**
 - **Описание:** Специфические настройки конфигурации, которые применяются к конкретной роли при подключении к конкретной базе данных. Хранятся на уровне кластера.
 - **Системный каталог:** `pg_db_role_setting`
 - **Как посмотреть:** `SELECT * FROM pg_db_role_setting;`

3. Внутренние системные объекты и каталоги:

- **Системные каталоги, разделяемые между базами данных (Shared System Catalogs):**
 - Помимо `pg_roles`, `pg_database`, `pg_tablespace`, существуют и другие системные каталоги, которые хранят глобальную информацию и разделяются между всеми базами данных в кластере. Они хранятся в подкаталоге `global` каталога данных (PGDATA). Примеры:
 - `pg_auth_members`: Членство ролей.
 - `pg_shdepend`: Зависимости разделяемых объектов.
 - `pg_shdescription`: Комментарии к разделяемым объектам.
 - `pg_shseclabel`: Метки безопасности разделяемых объектов.
 - **Как посмотреть:** Запросы к соответствующим `pg_` каталогам.
- **Файл контроля кластера (pg_control file):**
 - **Описание:** Бинарный файл, содержащий критически важную информацию о состоянии кластера (например, версия PostgreSQL, время инициализации, текущее состояние WAL, данные контрольной точки). Это центральный файл управления кластером.
 - **Как посмотреть:** Используется утилита `pg_controldata`.
- **Журнал предзаписи (Write-Ahead Log - WAL):**
 - **Описание:** Файлы журнала предзаписи, которые обеспечивают целостность данных и восстановление после сбоев. Они управляются на уровне кластера и хранятся в каталоге `pg_wal` (или `pg_xlog` в старых версиях).
 - **Как посмотреть:** Просмотр каталога `pg_wal` или использование функций `pg_current_wal_lsn()`, `pg_walfile_name()`.
- **Секреты кластера (Cluster Secrets):**
 - **Описание:** Включает в себя ключи шифрования, если они используются для шифрования данных или WAL. Управляются на уровне кластера.
 - **Как посмотреть:** Зависит от реализации шифрования.
- **Файлы PID (Process ID Files):**
 - **Описание:** Файлы, содержащие идентификаторы процессов (PID) основных процессов PostgreSQL (например, `postmaster.pid`), используемые для управления жизненным циклом кластера.
 - **Как посмотреть:** Прямой просмотр файлов в каталоге данных кластера.

4. Подкаталоги в каталоге данных кластера (PGDATA):

Эти подкаталоги содержат глобальные данные и метаданные, необходимые для работы всего кластера:

- `base`: Содержит подкаталоги для каждой базы данных (по их OID). Хотя данные баз данных находятся внутри, сам каталог `base` и его структура являются частью глобального уровня.
- `global`: Содержит разделяемые системные каталоги.
- `pg_wal` (или `pg_xlog`): Журнал предзаписи.
- `pg_tblspc`: Символические ссылки на табличные пространства.
- `pg_twophase`: Состояние двухфазных транзакций.
- `pg_commit_ts`: Метки времени фиксации транзакций.
- `pg_replslot`: Состояние слотов репликации.
- `pg_logical`: Состояние логической репликации.
- `pg_notify`: Состояние LISTEN/NOTIFY.
- `pg_snapshots`: Состояние снимков для логического декодирования.
- `pg_serial`: Статус сериализуемых транзакций.
- `pg_subtrans`: Статус подтранзакций.

- `pg_multixact`: Состояние многопользовательских транзакций.
- `pg_xact`: Состояние транзакций (частично).
- `pg_stat`: Файлы статистики, которые агрегируются на уровне кластера.

Этот список охватывает большинство сущностей и компонентов, которые существуют и управляются на уровне всего кластера PostgreSQL, а не только внутри отдельной базы данных.

Отличный ответ, как мне его сохранить в PDF?

К сожалению, я не могу напрямую сохранить этот текст в PDF-файл. Я являюсь языковой моделью и не имею доступа к функциям сохранения файлов на вашем компьютере или устройстве.

Однако, вы можете легко сделать это самостоятельно, используя следующие методы:

1. Скопировать и Вставить в Текстовый Редактор

Самый простой способ:

1. **Выделите весь текст**, который я предоставил.
2. **Скопируйте его** (обычно `Ctrl+C` на Windows/Linux или `Cmd+C` на Mac).
3. **Откройте текстовый редактор** (например, Microsoft Word, Google Docs, LibreOffice Writer или даже Блокнот/TextEdit).
4. **Вставьте текст** (обычно `Ctrl+V` на Windows/Linux или `Cmd+V` на Mac).
5. **Сохраните как PDF**:
 - В большинстве редакторов перейдите в меню **"Файл"** (File).
 - Выберите **"Сохранить как"** (Save As) или **"Экспорт"** (Export).
 - В выпадающем списке форматов выберите **"PDF"** (Portable Document Format).
 - Укажите имя файла и место сохранения.

2. Использовать Функцию Печати в PDF

Если у вас нет текстового редактора, вы можете использовать функцию печати:

1. **Откройте эту страницу в браузере.**
2. **Вызовите окно печати** (обычно `Ctrl+P` на Windows/Linux или `Cmd+P` на Mac).
3. В диалоговом окне печати, вместо выбора реального принтера, выберите опцию **"Сохранить как PDF"** (Save as PDF) или **"Microsoft Print to PDF"** (на Windows) или **"Adobe PDF"** (если установлен Adobe Acrobat).
4. Нажмите кнопку **"Печать"** (Print) или **"Сохранить"** (Save).
5. Укажите имя файла и место сохранения.

Оба метода должны работать для сохранения этого полезного списка.

Как используя Postgres, узнать физический размер базы данных, из них - размер всех объектов, сохраненных на диске, ассоциированные с этой базой данных?

Вы совершенно верно подметили, что для получения физического размера базы данных и ее объектов в PostgreSQL нужно использовать системные функции и представления.

Вот как это можно сделать, с пояснениями для каждого шага:

1. Размер всей базы данных

Функция `pg_database_size()` возвращает общий физический размер указанной базы данных, включая все ее таблицы, индексы, WAL-файлы, временные файлы и другую служебную информацию, хранящуюся в каталоге данных этой базы данных.

```
SELECT pg_database_size('ваша_база_данных_тут');
```

- **Пример для текущей базы данных:**

```
SELECT pg_database_size(current_database());
```

- **Пример для всех баз данных в кластере (в байтах):**

```
SELECT datname, pg_database_size(datname) AS total_size_bytes
FROM pg_database
ORDER BY total_size_bytes DESC;
```

- **С форматированием для читабельности (в ГБ, МБ и т.д.):**

```
SELECT datname, pg_size_pretty(pg_database_size(datname)) AS total_size_pretty
FROM pg_database
ORDER BY pg_database_size(datname) DESC;
```

Важно: `pg_database_size()` возвращает размер *каталога* базы данных на диске. Это включает в себя не только данные таблиц и индексов, но и некоторые системные файлы, временные файлы, WAL-файлы, которые относятся к этой базе данных, а также, возможно, данные из табличных пространств, если они используются для этой базы данных.

2. Размер всех объектов, ассоциированных с базой данных

Здесь мы будем использовать другие функции и представления, чтобы детализировать размер таблиц, индексов, TOAST-таблиц и других объектов.

2.1. Размер таблиц (включая TOAST)

`pg_total_relation_size()` возвращает общий дисковый объем, использованный для указанной таблицы, включая данные TOAST-таблиц (хранилище больших объектов) и связанные индексы.

```
SELECT
    relname AS table_name,
    pg_size_pretty(pg_total_relation_size(oid)) AS total_size,
    pg_size_pretty(pg_table_size(oid)) AS table_data_size, -- Только данные таблицы, без индексов и TOAST
    pg_size_pretty(pg_indexes_size(oid)) AS indexes_size, -- Только индексы
    pg_size_pretty(pg_relation_size(reltoastrelid)) AS toast_size -- Размер TOAST-таблицы, если есть
FROM pg_class
WHERE relkind = 'r' -- r = обычная таблица
ORDER BY pg_total_relation_size(oid) DESC;
```

Пояснения к функциям:

- `pg_total_relation_size(oid)`: Общий размер таблицы + индексы + TOAST.
- `pg_table_size(oid)`: Только размер основных данных таблицы (без индексов и TOAST).
- `pg_indexes_size(oid)`: Только размер всех индексов, ассоциированных с таблицей.
- `pg_relation_size(oid)`: Возвращает размер *основного фора* отношения. Для обычных таблиц это будет только основной файл данных, без TOAST и индексов. Это не то же самое, что `pg_table_size()`, которая включает в себя основную часть таблицы и видимость-карту/fsm.
- `reltoastrelid`: Системное поле в `pg_class`, которое содержит OID связанной TOAST-таблицы (если есть).

2.2. Размер отдельных индексов

Если вам нужен размер каждого индекса по отдельности:

```
SELECT
    c.relname AS index_name,
    pg_size_pretty(pg_relation_size(c.oid)) AS index_size,
    t.relname AS table_name
FROM pg_class c
JOIN pg_index i ON i.indexrelid = c.oid
JOIN pg_class t ON t.oid = i.indrelid
WHERE c.relkind = 'i' -- i = индекс
ORDER BY pg_relation_size(c.oid) DESC;
```

2.3. Размер материализованных представлений

Материализованные представления хранят данные на диске, как и таблицы:

```
SELECT
    relname AS materialized_view_name,
    pg_size_pretty(pg_total_relation_size(oid)) AS total_size
FROM pg_class
WHERE relkind = 'm' -- m = материализованное представление
ORDER BY pg_total_relation_size(oid) DESC;
```

2.4. Размер таблиц и индексов по схеме

Чтобы получить размеры объектов в конкретной схеме:

```
SELECT
    n.nspname AS schema_name,
    relname AS object_name,
    CASE relkind
        WHEN 'r' THEN 'table'
        WHEN 'i' THEN 'index'
        WHEN 'm' THEN 'materialized view'
        WHEN 'S' THEN 'sequence'
        WHEN 't' THEN 'TOAST table'
        ELSE relkind::text
    END AS object_type,
    pg_size_pretty(pg_total_relation_size(c.oid)) AS total_size_pretty
FROM pg_class c
JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE n.nspname = 'public' -- Замените на нужную схему
    AND c.relkind IN ('r', 'i', 'm', 't') -- Таблицы, индексы, мат.представления, TOAST-таблицы
ORDER BY pg_total_relation_size(c.oid) DESC;
```

Дополнительные соображения:

- **Табличные пространства (Tablespaces):** Если ваша база данных или ее объекты хранятся в пользовательских табличных пространствах, то `pg_database_size()` будет учитывать и их. Для просмотра размера *конкретного* табличного пространства:

```
SELECT spcname, pg_size_pretty(pg_tablespace_size(oid)) AS tablespace_size
FROM pg_tablespace;
```

- **Временные файлы:** Временные файлы, создаваемые при выполнении запросов (например, для больших сортировок или хешей), не учитываются в размере постоянных объектов, но они временно занимают место на диске. Их размер может быть значительным во время выполнения сложных запросов.
- **WAL-файлы (Write-Ahead Log):** WAL-файлы находятся в каталоге `pg_wal` (или `pg_xlog`), который является частью общего каталога данных кластера. `pg_database_size()` не включает весь объем WAL-файлов, которые могут быть заняты другими базами данных. Она учитывает только те WAL-файлы, которые генерируются или обрабатываются в контексте операций с данной БД.

- **VACUUM и REINDEX:** PostgreSQL не освобождает место на диске немедленно после удаления строк или обновления данных. Место помечается как свободное для повторного использования. Чтобы физически уменьшить размер файлов на диске (если много "мертвого" кортежа), нужно выполнять **VACUUM FULL** или **REINDEX**. **Осторожно:** **VACUUM FULL** блокирует таблицу!
- **Служебные каталоги:** Кроме данных, связанных с конкретными объектами, в каталоге базы данных есть и другие служебные файлы (например, `pg_class`, `pg_attribute` и другие системные каталоги), которые также занимают место. `pg_database_size()` учитывает все эти файлы.

Используя эти запросы, вы получите подробное представление о том, сколько дискового пространства занимает ваша база данных PostgreSQL и ее компоненты.