Задачи администрирования

Управление расширениями

Авторы: Егор Рогов, Павел Лузанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Расширения в PostgreSQL

Создание и управление расширениями

Обновление расширений

Особенности работы pg_dump

Расширяемость PostgreSQL

Возможности

- функции и языки программирования
- типы данных, операторы, методы доступа
- обертки сторонних данных (FDW)

Механизмы

- изменяемый системный каталог
- API для подключения внешних обработчиков
- загрузка и выполнение пользовательского кода

Расширяемость — важнейшая черта PostgreSQL — это возможность подключать «на лету» новый функционал без изменения кода сервера.

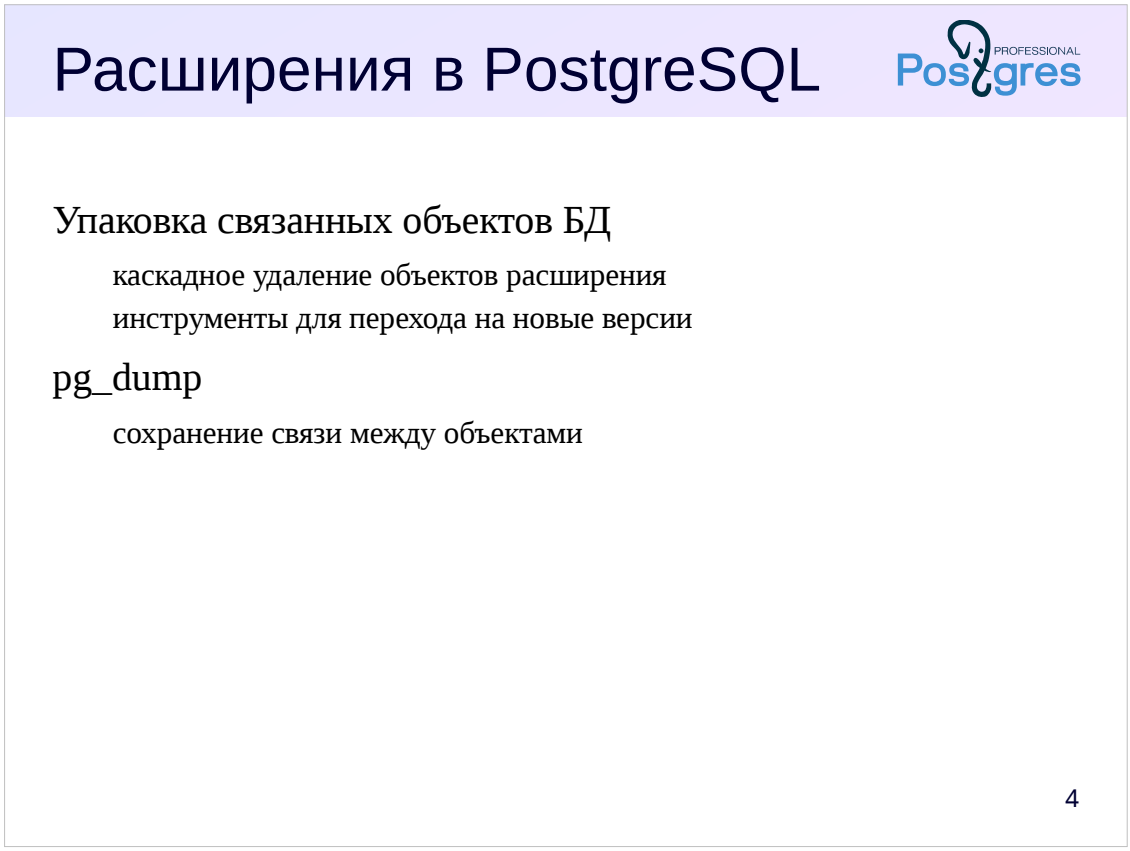
Таким образом можно добавлять языки программирования и разрабатывать на них функции, определять новые типы данных и операторы для работы с ними, создавать новые методы доступа для типов данных, разрабатывать обертки сторонних данных для подключения к внешним источникам.

Для того чтобы это было возможным, системный каталог PostgreSQL хранит большое количество информации об объектах БД. Эта информация не жестко зашита в код сервера. Пользователи могут изменять таблицы системного каталога, тем самые добавляя новые объекты и связанный с ними функционал.

Кроме того, в исходном коде PostgreSQL встроено большое количество хуков и различных API для подключения пользовательских функций. Это делает возможным разрабатывать такие расширения как pg_stat_statements, auto_explain, pldebugger и многие, многие другие.

Завершает картину возможность загружать в серверные процессы пользовательский код. Например, можно написать разделяемую библиотеку и подключать ее по ходу работы.

https://postgrespro.ru/docs/postgresql/10/extend-how
В качестве предостережения следует отметить, что выполнение процессами сервера неправильно написанного пользовательского кода может привести к катастрофическим последствиям. Следует доверять только проверенному коду из надежных источников.



Часто требуется добавить в базу данных связанные между собой объекты. Например, для нового типа данных потребуются функции и операторы.

Чтобы установить зависимости между отдельными объектами, в PostgreSQL используется механизм расширений. Связанные объекты упаковываются в единое расширение, что облегчает управление:

- отдельный объект расширения нельзя удалить, а удаление расширения автоматически удаляет все его объекты;
- специальные инструменты облегчают переход на новые версии.

Копия базы данных, выполненная утилитой pg_dump, не может просто выгружать описания отдельных объектов расширения. Ведь при восстановлении из такой копии принадлежность к расширению потеряется. Поэтому утилита pg_dump должна сохранять связь между объектами расширения.

https://postgrespro.ru/docs/postgresql/10/extend-extensions

Источники

Каталог contrib

- дополнительно поставляемые модули
- дополнительно поставляемые программы

PGXN — сеть расширений

Несколько десятков расширений распространяются вместе с СУБД. Они расположены в каталоге contrib дистрибутива PostgreSQL. Расширения из этого списка поддерживаются разработчиками PostgreSQL.

https://postgrespro.ru/docs/postgresql/10/contrib

https://postgrespro.ru/docs/postgresql/10/contrib-prog

Другой источник — PostgreSQL Extension Network (PGXN) — сеть расширений созданная по аналогии с сетью CPAN для Perl.

https://pgxn.org/

Расширения могут распространяться и другими способами, в том числе через пакетные репозитории дистрибутивов ОС.

А можно разработать собственное расширение. Возможности и свойства расширений будут рассмотрены в демонстрации на примере учебного расширения uom.

Управление расширением

Команды управления расширениями

- загрузка объектов в базу данных — CREATE EXTENSION
- обновление версии — ALTER EXTENSION
- удаление всех объектов расширения — DROP EXTENSION

Файлы расширений

- управляющие файлы
- файлы SQL

Для управления расширениями в PostgreSQL используются команды SQL.

Так, загрузка объектов расширения в базу данных выполняется при создании расширения (CREATE EXTENSION). Переход на новую версию — ALTER EXTENSION.

А удалить расширения со всеми его объектами можно командой DROP EXTENSION. Не требуется разрабатывать отдельный uninstall-скрипт.

Для работы этих команд в составе расширений должны быть специальные файлы. К ним относятся управляющие файлы, содержащие важную информацию о свойствах расширения, а также файлы SQL с командами создания объектов расширения.

Как правило расширения содержат и другие файлы, например разделяемые библиотеки с функциями для поддержки создаваемых объектов. Но, с точки зрения управления расширениями, такие файлы не важны.

Управляющие файлы

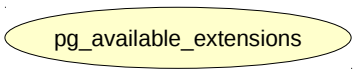
Имя файла: *имя*.control

Расположение: SHAREDIR/extension

```
# Некоторые параметры

directory = 'extension' # каталог для файлов SQL
default_version = 1.0   # версия по умолчанию
relocatable = true      # возможность изменить схему
superuser = true        # создает только суперпользователь
encoding = UTF8         # кодировка файлов SQL
#requires = ''            должны быть установлены

comment = 'Use only ASCII characters'
```



При выполнении команды CREATE EXTENSION механизм расширений проверяет наличие управляющего файла в каталоге SHAREDIR/extension.

Расположение SHAREDIR можно посмотреть командой

```
$ pg_config --sharedir
```

Имя управляющего файла состоит из имени расширения, к которому добавляется «.control».

Управляющий файл имеет формат конфигурационных файлов PostgreSQL и состоит из пар «ключ = значение». PostgreSQL не определяет кодировку символов файла, поэтому следует использовать только символы ASCII.

Список доступных расширений находится в таблице системного каталога pg_available_extensions.

Создание расширения

```
CREATE EXTENSION имя [VERSION 'версия'];
```

имя.control

```
default_version = 1.0
...
```

если версия не указана

имя--*версия*.sql

```
\echo Use "CREATE EXTENSION имя" to load this file. \quit

COMMENT ON EXTENSION имя IS 'Описание на русском';

-- Создание объектов расширения
...
```

pg_available_extension_versions

Помимо управляющего файла требуется еще файл SQL с командами на создание объектов БД. Имя файла SQL зависит от устанавливаемой версии расширения.

Версию расширения можно явно указать в предложении VERSION команды CREATE EXTENSION. По умолчанию будет использоваться версия из параметра default_version управляющего файла.

Имя файла SQL строится по шаблону *имя*`--`*версия*`.sql`,

где *версия* не обязательно должна состоять из цифр. Она может включать и другие символы (кроме: «`--`» , а также «`-`» в начале или в конце).

Список доступных версий расширений находится в таблице системного каталога pg_available_extension_versions.

Строки файла SQL, начинающиеся на `\echo`, механизм расширений считает комментариями. Для предотвращения случайного запуска файла из psql, обычно в начало файла SQL добавляют такую строку с предупреждением и завершают ее командой `\quit`.

В файле SQL можно использовать кириллицу, предварительно указав кодировку символов в параметре encoding управляющего файла. Например, можно задать русскоязычный комментарий к расширению в команде COMMENT ON EXTENSION.

Создание расширения

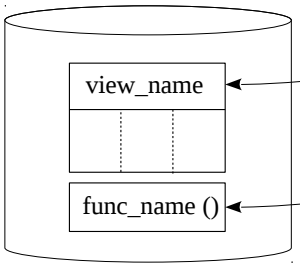
Особый случай: включение существующих объектов

`CREATE EXTENSION имя FROM unpackaged;`

имя--unpackaged--1.0.sql

```
ALTER EXTENSION имя ADD FUNCTION func_name();
ALTER EXTENSION имя ADD VIEW view_name;

-- Создание других объектов
```



Функционал многих расширений из каталога contrib был доступен до появления расширений в PostgreSQL. С появлением расширений потребовалось перенести отдельные объекты в состав расширения.

Это можно сделать командой:

```
ALTER EXTENSION имя ADD object_type name;
```

Отвязать объекты от расширения (без удаления объектов):

```
ALTER EXTENSION имя DROP object_type name;
```

Если эти команды поместить в файл SQL, названный *имя*`--unpackaged--1.0.sql`, и выполнить команду

```
CREATE EXTENSION имя FROM unpackaged;
```

то из существующих объектов будет создано расширение версии 1.0.

В общем случае для команды CREATE EXTENSION с фразой FROM имя файла SQL должно отвечать шаблону *имя*`--`*старая_версия*`--`*новая_версия*`.sql`.

В качестве *старой_версии* принято использовать «unpackaged», хотя это не обязательно; разрешается использовать любое допустимое имя версии.

Для расширений contrib в каталоге SHAREDIR/extension можно увидеть много файлов формата имя`--unpackaged--1.0.sql`, содержащих команды ALTER EXTENSION ... ADD.

Выбор схемы

```
CREATE EXTENSION имя [SCHEMA схема];
```

имя.control

```
relocatable = false
schema = схема
...
```

имя--*версия*.sql

```
SET LOCAL search_path TO @extschema@ ;

... SELECT @extschema@ .function_name()...
```

макроподстановка

Расширение как объект базы данных не принадлежит ни одной схеме. Но все объекты расширения создаются в какой-то схеме. При необходимости в SQL-файле расширения можно создавать новые схемы.

Хотя это и не обязательно, но обычно все объекты расширения размещают в одной схеме.

Схему для размещения объектов можно явно указать в команде CREATE EXTENSION или задать в параметре schema управляющего файла. Иначе будет использоваться первая схема из параметра конфигурации search_path.

Выбранная в результате схема явно устанавливается в search_path в начале выполнения файла SQL. Внутри самого файла можно обращаться к этой схеме, используя макроподстановку @extschema@.

Если параметр расширения relocatable установить в true (по умолчанию false), то объекты расширения можно будет переносить в другую схему командой:

```
ALTER EXTENSION имя SET SCHEMA новая_схема;
```

Обновление: 1.0 → 1.1

имя.control

```
default_version = 1.1
...
```

1. `CREATE EXTENSION` *`имя`*`;`

имя--1.1.sql

```
\echo Use "CREATE EXTENSION имя" to load this file. \quit

-- Создание всех объектов версии 1.1
```

2. `ALTER EXTENSION` *`имя`* `UPDATE;`

имя--1.0--1.1.sql

```
\echo Use "ALTER EXTENSION имя" to load this file. \quit

-- Команды обновления из версии 1.0 в 1.1
```


Если со временем требуется изменить существующие объекты расширения или добавить новые, то выпускается новая версия.

В управляющем файле новой версии расширения обычно меняется параметр default_version. А при установке возможны две ситуации:

1) Предыдущая версия расширения не была установлена. Поэтому можно подготовить файл SQL для новой версии, включающий создание всех объектов расширения.

2) Для тех, у кого установлена предыдущая версия расширения, следует подготовить файл SQL для обновление версии. Формат имени файла обновления следующий:
   `старая_версия``--`*`новая_версия`*`.sql`.
   Внутри такого файла должны быть только команды, обновляющие объекты расширения со старой версии на новую.
   Само обновление выполняется командой:

   `ALTER EXTENSION` *`имя`* `UPDATE;`

Обновление: 1.0 → 1.2

```
ALTER EXTENSION имя UPDATE TO '1.2';
```

текущая версия 1.0

имя--1.0--1.1.sql

```
\echo Use "ALTER EXTENSION имя" to load this file. \quit

-- Команды обновления из версии 1.0 в 1.1
```

имя--1.1--1.2.sql

```
\echo Use "ALTER EXTENSION имя" to load this file. \quit

-- Команды обновления из версии 1.1 в 1.2
```

pg_extension_update_paths ('*имя*')

При наличии нескольких версий расширения, в команде ALTER EXTENSION … UPDATE можно явно указать требуемый номер версии.

Команда ALTER EXTENSION умеет последовательно выполнять несколько файлов обновления для перехода с текущей на запрошенную версию. Например для перехода с версии 1.0 на версию 1.2 сначала будут выполнен файл *имя*`--1.0--1.1.sql`, а затем *имя*`--1.1--1.2.sql`.

Механизм расширений ничего не знает о том, какая версия новее, и строит возможные цепочки обновлений исходя из имеющихся файлов. Это позволит при наличии файла *имя*`--1.2--1.0.sql` перейти с версии 1.2 на 1.0. Если есть несколько путей обновления до запрошенной версии, то выбирается наиболее короткий.

Функция pg_extension_update_paths('*имя*') показывает возможные пути обновления для указанного расширения.

Управляющие файлы

Дополнительные файлы: *имя*--*версия*.control
Расположение: SHAREDIR/extension

имя.control

```
...
#requires = ''            должны быть установлены
```

имя--1.1.control

```
...
requires = 'postgres_fdw'
```


Помимо основного управляющего файла, могут быть еще и дополнительные файлы для каждой версии расширения.

Формат имени таких файлов: *имя*`--`*версия*`.control`.

При создании расширения считывается основной управляющий файл и дополнительный файл этой версии (если он есть). Параметры дополнительного файла имеют предпочтение перед основным.

В приведенном примере, при установке версии расширения 1.1 будет проверяться наличие установленного расширения postgres_fdw, хотя в предыдущей версии этого не требовалось.

pg_dump: команды DDL

```
CREATE EXTENSION имя;
```

имя--*версия*.sql

```
CREATE FUNCTION ...
CREATE VIEW ...
CREATE TABLE ...
```

вывод утилиты pg_dump

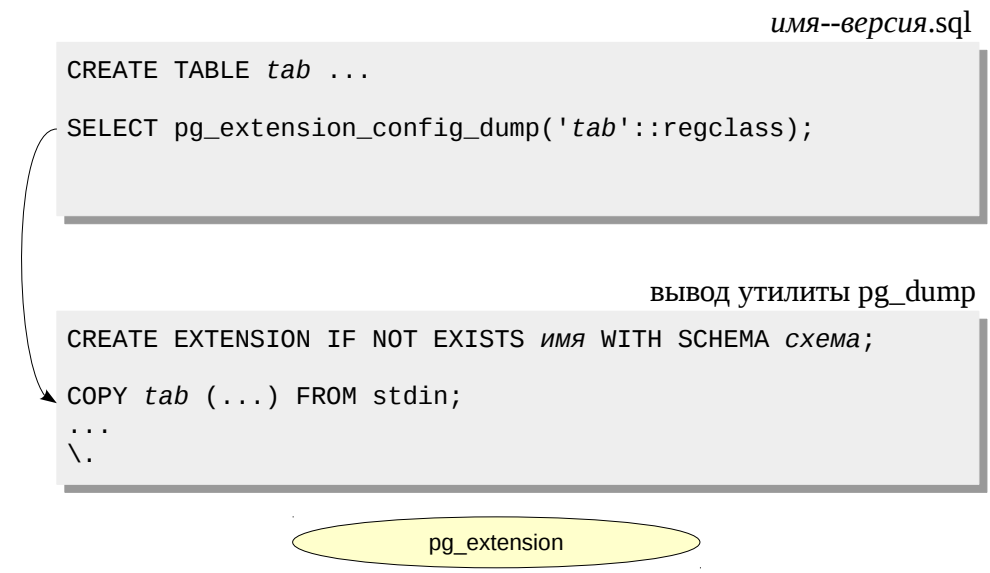
```
CREATE EXTENSION IF NOT EXISTS имя WITH SCHEMA схема;
```


Утилита pg_dump обрабатывает расширения особым образом.

Чтобы не потерять зависимости между объектами, нельзя просто выгружать команды определения объектов расширения (CREATE FUNCTION, CREATE VIEW,…). Поэтому в копию включается команда CREATE EXTENSION. При восстановлении из такой копии расширение будет заново создано со всеми объектами и связями.

Перед восстановлением нужно убедиться, что в системе установлена такая же версия расширения, что и при создании копии.

pg_dump: строки таблиц



В расширение можно включать и таблицы.

По умолчанию строки таблиц, вставленные после установки расширения, не будут выгружаться утилитой pg_dump.

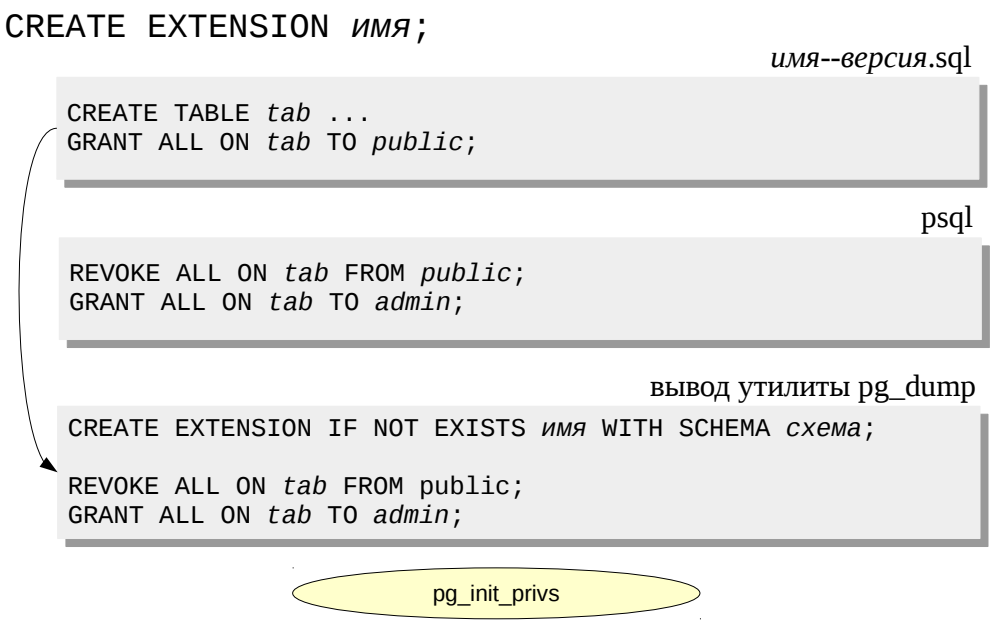
Если же требуется включить содержимое таблиц в выгрузку pg_dump, то в файле SQL расширения нужно вызвать для каждой таблицы функцию pg_extension_config_dump().

Функция имеет два параметра. Первый — это OID таблицы, второй (необязательный) — фраза WHERE, которую pg_dump будет применять к таблице при выгрузке.

Эта же функция используется и для последовательностей, которые могут быть связаны с таблицей. В вывод pg_dump будет записываться вызов функции setval, устанавливающий последнее полученное из последовательности значение.

Функцию pg_extension_config_dump() можно вызывать только из файлов SQL расширения.

pg_dump: привилегии

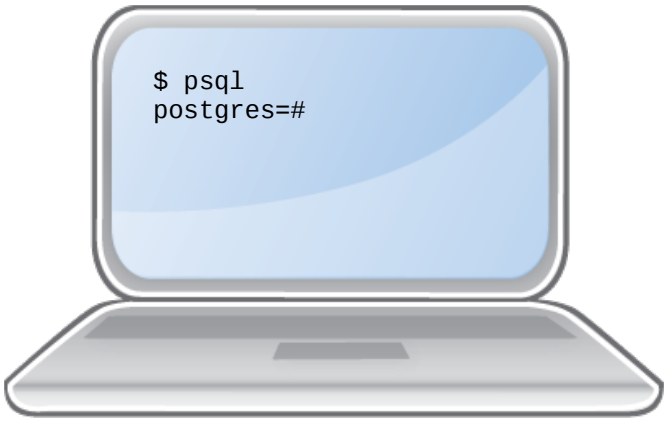


Скрипты расширений могут включать команды GRANT и REVOKE для управления привилегиями на объекты расширения.

Информация о выданных/отозванных расширениями привилегиях сохраняется в системном каталоге pg_init_privs. Это позволяет pg_dump сформировать набор команд GRANT/REVOKE, которые восстанавливают права доступа на объекты такими, какие они были на момент создания копии.

В приведенном примере скрипт расширения выдает права на таблицу псевдороли public. Затем, в процессе эксплуатации, права на таблицу у public отнимаются и передаются роли admin. В копию pg_dump должны попасть не только команда выдачи привилегий на таблицу для роли admin, но отзыв привилегий у псевдороли public, которые будут выданы в результате выполнения CREATE EXTENSION.

Демонстрация



Итоги

Расширяемость — важнейшее свойство PostgreSQL

Расширения — упаковка связанных объектов БД

Механизм расширений содержит инструменты для обновления версий, поддержки работы pg_dump

Практика

1. Установите расширение uom и убедитесь, что оно появилось в списке доступных.
2. Создайте расширение uom, не указывая версию.
   Какая версия создалась и какими скриптами?
3. Добавьте в справочник футы и дюймы.
4. Измените доступ к справочной таблице:
   привилегия SELECT должна быть только у специально созданной роли, а не у всех.
5. Проверьте, как pg_dump выгружает объекты расширения: таблицу, тип, функции и операторы, содержимое таблицы, права доступа.

1. Файлы расширения расположены в подкаталоге uom домашнего каталога пользователя student. Процесс установки аналогичен тому, что использовался в демонстрации.

3. При добавлении записей важно, чтобы у добавленных записей значение столбца predefined было false.

Коэффициенты пересчета:

1 фут = 0.3048 м

1 дюйм = 0.0254 м