

# Задачи администрирования Управление расширениями



## **Авторские права**

© Postgres Professional, 2016–2022.

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

## **Использование материалов курса**

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

## **Обратная связь**

Отзывы, замечания и предложения направляйте по адресу:

[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

## **Отказ от ответственности**

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или непрямым, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Расширения в PostgreSQL

Создание и управление расширениями

Обновление расширений

Особенности работы pg\_dump

## 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 зашита жестко в код сервера. Пользователи могут изменять содержимое таблиц системного каталога, тем самым 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/13/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.

## Упаковка связанных объектов БД

связь объектов с расширением, каскадное удаление  
инструменты для перехода на новые версии

### pg\_dump

сохранение связи между объектами

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` сохраняет связь между объектами и расширением.

<https://postgrespro.ru/docs/postgresql/13/extend-extensions>

Пакет дистрибутива и каталог contrib

дополнительно поставляемые модули

дополнительно поставляемые программы

PGXN — сеть расширений

Возможность создания собственного расширения

Несколько десятков расширений распространяются вместе с СУБД и поддерживаются разработчиками PostgreSQL. Обычно они устанавливаются из того же пакета, что и ядро. Исходный код таких расширений находится в подкаталоге contrib.

<https://postgrespro.ru/docs/postgresql/13/contrib>

Кроме расширений, в пакете (и в каталоге contrib) находятся несколько дополнительных программ.

<https://postgrespro.ru/docs/postgresql/13/contrib-prog>

Еще один 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.

## Установка расширения

Файлы с исходным кодом учебного расширения UOM (единицы измерения) расположены в каталоге uom домашнего каталога пользователя student:

```
student$ ls -l /home/student/uom
```

```
total 28
-rw-rw-r-- 1 student student 163 ноя  4 09:25 Makefile
-rw-rw-r-- 1 student student 1132 ноя  4 09:25 README.md
-rw-rw-r-- 1 student student 1644 ноя  4 09:25 uom--1.0--1.1.sql
-rw-rw-r-- 1 student student  956 ноя  4 09:25 uom--1.0.sql
-rw-rw-r-- 1 student student  755 ноя  4 09:25 uom--1.1--1.2.sql
-rw-rw-r-- 1 student student 2794 ноя  4 09:25 uom--1.2.sql
-rw-rw-r-- 1 student student   93 ноя  4 09:25 uom.control
```

Если расширение требует компиляции и используется пакетная сборка PostgreSQL, дополнительно должен быть установлен пакет для разработчиков; его имя обычно содержит слово dev, например postgresql-server-dev-13. Пакет содержит все необходимое для сборки, включая заголовочные файлы PostgreSQL.

Наше расширение содержит только функции на SQL и не требует сборки. Makefile использует инфраструктуру PGXS для установки расширений:

```
student$ cat /home/student/uom/Makefile
```

```
EXTENSION = uom
DATA = uom--1.0.sql uom--1.2.sql uom--1.0--1.1.sql  uom--1.1--1.2.sql

PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
```

Установка расширения обычно выполняется суперпользователем, в окружении которого путь до исполняемых файлов PostgreSQL может быть не указан. Поэтому в команде установки можно явно задать расположение утилиты pg\_config:

```
student$ sudo make install -C /home/student/uom PG_CONFIG=/usr/lib/postgresql/13/bin/pg_config
```

```
make: Entering directory '/home/student/uom'
/bin/mkdir -p '/usr/share/postgresql/13/extension'
/bin/mkdir -p '/usr/share/postgresql/13/extension'
/usr/bin/install -c -m 644 ./uom.control '/usr/share/postgresql/13/extension/'
/usr/bin/install -c -m 644 ./uom--1.0.sql ./uom--1.2.sql ./uom--1.0--1.1.sql ./uom--1.1--1.2.sql  '/usr/share/postgresql/13/extension/'
make: Leaving directory '/home/student/uom'
```

Теперь файлы расширения uom должны появиться в каталоге SHAREDIR/extension. Расположение SHAREDIR можно посмотреть утилитой pg\_config или одноименной функцией SQL:

```
=> SELECT setting || '/extension' AS extension_dir
FROM pg_config()
WHERE name = 'SHAREDIR';

      extension_dir
-----
/usr/share/postgresql/13/extension
(1 row)
```

Кроме uom здесь расположены файлы других установленных расширений.

## 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 DDL.

Загрузка объектов расширения в базу данных выполняется при создании расширения (`CREATE EXTENSION`). Переход на новую версию, перемещение в другую схему — `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.

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'
```

pg\_available\_extensions

8

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.



Список расширений, доступных для загрузки в БД, можно получить запросом:

```
=> SELECT name, default_version, installed_version
FROM pg_available_extensions
ORDER BY name;
```

name	default_version	installed_version
adminpack	2.1	
amcheck	1.2	
autoinc	1.0	
bloom	1.0	
btree_gin	1.3	
btree_gist	1.5	
citext	1.6	
cube	1.4	
dblink	1.2	
dict_int	1.0	
dict_xsyn	1.0	
earthdistance	1.1	
file_fdw	1.0	
fuzzystrmatch	1.1	
hstore	1.7	
insert_username	1.0	
intagg	1.1	
intarray	1.3	
isn	1.2	
lo	1.1	
ltree	1.2	
moddatetime	1.0	
pageinspect	1.8	
pg_buffercache	1.3	
pg_freespacemap	1.2	
pg_prewarm	1.2	
pg_stat_statements	1.8	
pg_trgm	1.5	
pg_visibility	1.2	
pg_wait_sampling	1.1	
pgaudit	1.5.2	
pgcrypto	1.3	
pgrowlocks	1.2	
pgstattuple	1.5	
plpgsql	1.0	1.0
postgres_fdw	1.0	
refint	1.0	
seg	1.3	
sslnfo	1.2	
tablefunc	1.0	
tcn	1.0	
tsm_system_rows	1.0	
tsm_system_time	1.0	
unaccent	1.1	
uom	1.2	
uuid-osp	1.1	
xml2	1.1	

(47 rows)

По умолчанию установлено только расширение plpgsql.

Посмотрим на расширение uom. Список версий расширения:

```
=> SELECT name, version, installed
FROM pg_available_extension_versions
WHERE name = 'uom'
ORDER BY version;
```

name	version	installed
uom	1.0	f
uom	1.1	f
uom	1.2	f

(3 rows)

Версии расширений в PostgreSQL понимаются как имена, а не как номера. Другими словами, 1.1 и 1.2 — это просто две разные версии, их порядок не определен.

Содержимое управляющего файла:

```
student$ cat /usr/share/postgresql/13/extension/uom.control
```

```
default_version = '1.2'  
relocatable = true  
encoding = UTF8  
comment = 'Units of Measurement'
```

Версия по умолчанию 1.2.

# 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

10

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.

Установим версию 1.0 объектов расширения в отдельной базе данных:

```
=> CREATE DATABASE admin_extensions;
```

CREATE DATABASE

```
=> \c admin_extensions
```

You are now connected to database "admin\_extensions" as user "student".

```
=> CREATE EXTENSION uom VERSION '1.0';
```

CREATE EXTENSION

Теперь uom появляется в списке расширений текущей базы данных:

```
=> \dx
```

```

              List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
uom     | 1.0     | public   | Единицы измерения. uom--1.0.sql
(2 rows)
```

Расширение умеет переводить значения из одной единицы длины в другую:

```
=> SELECT uom2uom(1, 'верста', 'сажень') AS "Сажень в версте";
```

```

      Сажень в версте
-----
500.0000000000000000
(1 row)
```

Список поддерживаемых единиц измерения записан в таблице:

```
=> SELECT * FROM uom_ref;
```

```

 name | k
-----+-----
м      | 1
км     | 1000
см     | 0.01
верста | 1066.8
сажень | 2.1336
аршин  | 0.7112
вершок | 0.04445
(7 rows)
```

Справочная таблица и функция входят в состав расширения. В этом легко убедиться:

```
=> \dx+ uom
```

```

      Objects in extension "uom"
      Object description
-----+-----
function uom2uom(numeric,text,text)
table uom_ref
(2 rows)
```

В файле SQL для версии 1.0 находятся команды создания таблицы и функции:

```
student$ cat /usr/share/postgresql/13/extension/uom--1.0.sql
```

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

```
COMMENT ON EXTENSION uom IS 'Единицы измерения. uom--1.0.sql';
```

```
-- Справочник единиц измерения
```

```
CREATE TABLE uom_ref (
  name      text,
  k         numeric
);
```

```
INSERT INTO uom_ref VALUES ('м',1), ('км',1000), ('см',0.01),
  ('верста',1066.8), ('сажень',2.1336), ('аршин',0.7112), ('вершок',0.04445);
```

```
-- Конвертируем значение из одной единицы длины в другую
```

```

CREATE FUNCTION uom2uom (value numeric, name_from text, name_to text)
    RETURNS numeric LANGUAGE SQL
AS $$ SELECT uom2uom.value *
    (SELECT k FROM uom_ref WHERE name = uom2uom.name_from) /
    (SELECT k FROM uom_ref WHERE name = uom2uom.name_to)    $$;

-- Необходимо для использования функции любым пользователем
GRANT SELECT ON uom_ref TO public;

```

Первая строка предотвращает выполнение файла напрямую из psql. Механизм расширений считает комментариями строки, начинающиеся на \echo, а psql выдаст сообщение и закончит работу благодаря \quit в конце.

---

Принадлежность к расширению не дает напрямую удалять объекты:

```
=> DROP FUNCTION uom2uom(numeric,text,text);
```

```

ERROR:  cannot drop function uom2uom(numeric,text,text) because extension uom requires it
HINT:  You can drop extension uom instead.

```

Но механизм расширений не отслеживает изменения объектов. Вносить изменения в объекты расширения не следует, но добавление столбца в таблицу или изменение функции не вызовет ошибки.

```
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

12

Расширение как объект базы данных относится к определенной схеме. Обычно все объекты расширения размещают в этой же схеме, это позволяет расширению иметь отдельное пространство имен для объектов.

Схему расширения можно явно указать в команде CREATE EXTENSION или задать в параметре `schema` управляющего файла. Если ни одно из этих значений не задано, действует общее правило: используется первая доступная схема из параметра конфигурации `search_path`.

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

Если параметр расширения `relocatable` установить в `true` (по умолчанию `false`), то расширение вместе с его объектами можно будет переносить в другую схему командой:

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

# Схема расширения

Расширение и его объекты можно переносить в другую схему (relocatable=true):

=> CREATE SCHEMA util;

CREATE SCHEMA

=> ALTER EXTENSION uom SET SCHEMA util;

ALTER EXTENSION

=> \dx uom

List of installed extensions			
Name	Version	Schema	Description
uom	1.0	util	Единицы измерения. uom--1.0.sql
(1 row)			

=> \df util.uom\*

List of functions			
Schema	Name	Result data type	Argument data types   Type
util	uom2uom	numeric	value numeric, name_from text, name_to text   func
(1 row)			

Вернемся в public:

=> ALTER EXTENSION uom SET SCHEMA public;

ALTER EXTENSION

## 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
```

14

Если со временем потребуется изменить существующие объекты расширения или добавить новые, выпускается следующая версия.

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

#### 1. Предыдущая версия расширения не установлена.

Следует подготовить файл SQL для новой версии, включающий создание всех объектов расширения.

#### 2. Установлена одна из предыдущих версий расширения.

Следует подготовить файл SQL для обновления версии. Формат имени файла обновления следующий:

*имя* - -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')`

15

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.

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, в составе расширения могут быть еще и дополнительные файлы для отдельных версий.

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.

## Обновление расширений

В следующей версии 1.1 добавлен тип данных для единиц длины, операторы сравнения для значений этого типа и класс операторов, позволяющий индексировать столбцы таблиц.

Файл с изменениями:

```
student$ cat /usr/share/postgresql/13/extension/uom--1.0--1.1.sql

\echo Use "ALTER EXTENSION uom UPDATE TO '1.1'" to load this file. \quit

COMMENT ON EXTENSION uom IS 'Единицы измерения. uom--1.0--1.1.sql';

-- Новый тип данных
CREATE TYPE uom AS (
    value numeric,
    name text
);

-- Реализация операторов сравнения для типа
CREATE FUNCTION uom_cmp (a uom, b uom) RETURNS int LANGUAGE SQL
AS 'SELECT CASE WHEN a2b.value > b.value THEN 1
                WHEN a2b.value < b.value THEN -1
                ELSE 0 END
    FROM (SELECT uom2uom(a.value, a.name, b.name)) AS a2b(value)';

CREATE FUNCTION uom_lt(a uom, b uom) RETURNS boolean
AS 'SELECT uom_cmp(a, b) = -1' LANGUAGE sql;

CREATE FUNCTION uom_le(a uom, b uom) RETURNS boolean
AS 'SELECT uom_cmp(a, b) IN (-1,0)' LANGUAGE sql;

CREATE FUNCTION uom_eq(a uom, b uom) RETURNS boolean
AS 'SELECT uom_cmp(a, b) = 0' LANGUAGE sql;

CREATE FUNCTION uom_ge(a uom, b uom) RETURNS boolean
AS 'SELECT uom_cmp(a, b) IN (0,1)' LANGUAGE sql;

CREATE FUNCTION uom_gt(a uom, b uom) RETURNS boolean
AS 'SELECT uom_cmp(a, b) = 1' LANGUAGE sql;

CREATE OPERATOR < (PROCEDURE = uom_lt, LEFTARG = uom, RIGHTARG = uom);
CREATE OPERATOR <= (PROCEDURE = uom_le, LEFTARG = uom, RIGHTARG = uom);
CREATE OPERATOR = (PROCEDURE = uom_eq, LEFTARG = uom, RIGHTARG = uom);
CREATE OPERATOR >= (PROCEDURE = uom_ge, LEFTARG = uom, RIGHTARG = uom);
CREATE OPERATOR > (PROCEDURE = uom_gt, LEFTARG = uom, RIGHTARG = uom);

CREATE OPERATOR CLASS uom_ops DEFAULT FOR TYPE uom USING btree AS
    OPERATOR 1 <,
    OPERATOR 2 <=,
    OPERATOR 3 =,
    OPERATOR 4 >=,
    OPERATOR 5 >,
    FUNCTION 1 uom_cmp(uom,uom);
```

При создании новой версии расширения возможны два сценария:

- Есть предыдущая версия, нужно обновить расширение.
- Предыдущей версии не было, нужно сразу создать более позднюю версию расширения.

Убедимся, что мы можем перейти с версии 1.0 на версию 1.1. Список доступных вариантов обновления:

```
=> SELECT *
FROM pg_extension_update_paths('uom')
WHERE path IS NOT NULL
ORDER BY source, target;
```

source	target	path
1.0	1.1	1.0--1.1
1.0	1.2	1.0--1.1--1.2
1.1	1.2	1.1--1.2

(3 rows)

Выполним обновление:

```
=> ALTER EXTENSION uom UPDATE TO '1.1';
```

```
ALTER EXTENSION
```

```
=> \dx uom
```

```

              List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
  uom  | 1.1     | public | Единицы измерения. uom--1.0--1.1.sql
(1 row)
```

Теперь будем создавать версию 1.1, предварительно удалив расширение. При удалении расширения удаляются и его объекты:

```
=> DROP EXTENSION uom;
```

```
DROP EXTENSION
```

```
=> \df uom*
```

```

              List of functions
 Schema | Name | Result data type | Argument data types | Type
-----+-----+-----+-----+-----
(0 rows)
```

Создаем версию 1.1:

```
=> CREATE EXTENSION uom VERSION '1.1';
```

```
CREATE EXTENSION
```

```
=> \dx uom
```

```

              List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
  uom  | 1.1     | public | Единицы измерения. uom--1.0--1.1.sql
(1 row)
```

Для создания версии 1.1 механизм расширений сначала будет искать файл uom--1.1.sql, которого нет. Но к версии 1.1 можно прийти по цепочке: сначала создать версию 1.0, затем обновить до 1.1. Это самый короткий путь до версии 1.1, а в нашем примере и единственный.

Посмотрим на тип данных uom. Для него определены операторы сравнения единиц длины:

```
=> SELECT ( 5, 'аршин' )::uom < ( 10, 'вершок' )::uom,
         ( 1, 'верста' )::uom <= (500, 'сажень' )::uom,
         ( 1, 'сажень' )::uom = ( 3, 'аршин' )::uom,
         ( 1, 'верста' )::uom >= (500, 'сажень' )::uom,
         (10, 'аршин' )::uom > ( 8, 'м' )::uom;

?column? | ?column? | ?column? | ?column? | ?column?
-----+-----+-----+-----+-----
f         | t         | t         | t         | f
(1 row)
```

Новый тип можно использовать в качестве типа данных для столбца таблицы:

```
=> CREATE TABLE t (len uom);
```

```
CREATE TABLE
```

```
=> INSERT INTO t VALUES
```

```
    ((3, 'сажень')::uom), ((5, 'м')::uom), ((17, 'вершок')::uom), ((10, 'аршин')::uom);
```

```
INSERT 0 4
```

Данные в таблице можно сортировать:

```
=> SELECT * FROM t ORDER BY len;
```

```

   len
-----
(17,вершок)
(5,м)
(3,сажень)
(10,аршин)
(4 rows)
```

И использовать индекс-В-дерево:

```
=> CREATE INDEX ON t USING btree (len);
```

CREATE INDEX

```
=> SET enable_seqscan TO off;
```

SET

```
=> EXPLAIN (costs off)
```

```
SELECT * FROM t ORDER BY len;
```

QUERY PLAN

```
-----  
Index Only Scan using t_len_idx on t  
(1 row)
```

```
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 объектов от расширения, нельзя просто выгружать команды определения отдельных объектов (CREATE FUNCTION, CREATE VIEW,...). В копию включается только команда CREATE EXTENSION, при выполнении которой создаются объекты расширения.

При восстановлении из такой копии расширение будет создано заново со всеми объектами и связями с помощью установленных в системе скриптов. Поэтому перед восстановлением нужно убедиться, что в системе установлена такая же версия расширения, что и при создании копии.

## pg\_dump: строки таблиц

```
CREATE EXTENSION имя;
```

*имя--версия.sql*

```
CREATE TABLE tab ...  
SELECT pg_extension_config_dump('tab'::regclass);
```

ВЫВОД утилиты pg\_dump

```
CREATE EXTENSION IF NOT EXISTS имя WITH SCHEMA схема;  
COPY tab (...) FROM stdin;  
...  
\.
```

pg\_extension

19

В расширение можно включать и таблицы.

По умолчанию строки таблиц, входящих в расширение, считаются его частью и не выгружаются утилитой pg\_dump.

Если же таблица содержит пользовательские данные, требуется включить ее строки в выгрузку pg\_dump. Для этого в файле SQL расширения нужно вызвать для каждой такой 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.

## Выгрузка pg\_dump

В следующей версии расширения — 1.2 — добавлен вызов функции pg\_extension\_config\_dump.

Теперь pg\_dump будет выгружать содержимое таблицы uom\_ref. Добавленный в таблицу столбец predefined нужен для определения строк, которые были вставлены в таблицу не скриптом расширения, а уже после создания.

Файл обновления:

```
student$ cat /usr/share/postgresql/13/extension/uom--1.1--1.2.sql

\echo Use "ALTER EXTENSION uom UPDATE TO '1.2'" to load this file. \quit

COMMENT ON EXTENSION uom IS 'Единицы измерения. uom--1.1--1.2.sql';

-- Добавим признак, что строка добавлена при установке расширения.
ALTER TABLE uom_ref ADD COLUMN predefined boolean;

UPDATE uom_ref SET predefined = true;

ALTER TABLE uom_ref
  ALTER COLUMN predefined SET NOT NULL,
  ALTER COLUMN predefined SET DEFAULT false;

-- Если справочник будет пополняться при эксплуатации расширения,
-- то pg_dump должен выгружать новые строки.
SELECT pg_extension_config_dump('uom_ref'::regclass, 'WHERE NOT predefined');
```

Для перехода на версию 1.2 мы не можем просто удалить расширение и пересоздать его. Помешает столбец len таблицы t:

```
=> DROP EXTENSION uom;
```

```
ERROR:  cannot drop extension uom because other objects depend on it
DETAIL:  column len of table t depends on type uom
HINT:   Use DROP ... CASCADE to drop the dependent objects too.
```

Если мы не готовы удалять таблицу (или столбец len), то следует выполнить обновление:

```
=> ALTER EXTENSION uom UPDATE;
```

```
ALTER EXTENSION
```

Без указания версии обновление выполняется до версии по умолчанию из управляющего файла:

```
=> \dx uom
```

```

              List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
 uom   | 1.2     | public | Единицы измерения. uom--1.1--1.2.sql
(1 row)
```



```
CREATE EXTENSION имя;
```

*имя--версия.sql*

```
CREATE TABLE tab ...  
GRANT ALL ON tab TO public;
```

*psql*

```
REVOKE ALL ON tab FROM public;  
GRANT ALL ON tab TO admin;
```

вывод утилиты *pg\_dump*

```
CREATE EXTENSION IF NOT EXISTS имя WITH SCHEMA схема;  
REVOKE ALL ON tab FROM public;  
GRANT ALL ON tab TO admin;
```

*pg\_init\_privs*

21

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

Информация о выданных или отозванных расширениями привилегиях сохраняется в системном каталоге *pg\_init\_privs*. Это позволяет *pg\_dump* сформировать «разницу» в виде набора команд GRANT и REVOKE, которые после выполнения CREATE EXTENSION устанавливают права доступа на объекты такими, какими они были на момент создания копии.

В приведенном примере скрипт расширения выдает права на таблицу псевдороди *public*. Затем, в процессе эксплуатации, права на эту 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 псевдороди *public*, которые будут выданы в результате выполнения CREATE EXTENSION.

Расширяемость — важнейшее свойство PostgreSQL

Расширения — упаковка связанных объектов БД

Механизм расширений содержит инструменты для обновления версий, поддержки работы `pg_dump`

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

## 1. Установка расширения

Устанавливаем файлы расширения:

```
student$ sudo make install -C /home/student/uom PG_CONFIG=/usr/lib/postgresql/13/bin/pg_config
```

```
make: Entering directory '/home/student/uom'
/bin/mkdir -p '/usr/share/postgresql/13/extension'
/bin/mkdir -p '/usr/share/postgresql/13/extension'
/usr/bin/install -c -m 644 ./uom.control '/usr/share/postgresql/13/extension/'
/usr/bin/install -c -m 644 ./uom--1.0.sql ./uom--1.2.sql ./uom--1.0--1.1.sql ./uom--1.1--1.2.sql '/usr/share/postgresql/13/extension/'
make: Leaving directory '/home/student/uom'
```

Проверим, что расширение доступно для загрузки в базу данных:

```
=> SELECT *
FROM pg_available_extensions
WHERE name = 'uom';
```

name	default_version	installed_version	comment
uom	1.2		Units of Measurement

(1 row)

## 2. Создание расширения в базе данных

Создаем расширение uom в новой базе:

```
=> CREATE DATABASE admin_extensions;
```

CREATE DATABASE

```
=> \c admin_extensions
```

You are now connected to database "admin\_extensions" as user "student".

```
=> CREATE EXTENSION uom;
```

CREATE EXTENSION

Версия расширения соответствует значению параметра default\_version:

```
=> \dx uom
```

List of installed extensions			
Name	Version	Schema	Description
uom	1.2	public	Единицы измерения. uom--1.2.sql

(1 row)

Поскольку для версии 1.2 есть файл uom--1.2.sql, то он и используется для создания расширения. А для тех, кто хочет перейти с 1.1 или с 1.0, имеются соответствующие пути обновления.

## 3. Расширение справочника

Добавляем футы и дюймы. Столбец predefined заполняется значением по умолчанию (false):

```
=> INSERT INTO uom_ref VALUES ('фут', 0.3048);
```

INSERT 0 1

```
=> INSERT INTO uom_ref VALUES ('дюйм', 0.0254);
```

INSERT 0 1

Сколько же дюймов в футе?

```
=> SELECT uom2uom(1, 'фут', 'дюйм');
```

uom2uom
12.0000000000000000

(1 row)

## 4. Изменение доступа

Создаем отдельную роль для доступа к таблице uom\_ref:

```
=> CREATE ROLE util;
```

CREATE ROLE

Читать из таблицы может только новая роль:

```
=> GRANT SELECT ON uom_ref TO util;
```

GRANT

```
=> REVOKE SELECT ON uom_ref FROM public;
```

REVOKE

## 5. pg\_dump и объекты расширений

Информация о том, содержимое каких таблиц будет выгружать pg\_dump, сохраняется в pg\_extension:

```
=> SELECT extname, extconfig::regclass[], extcondition
FROM pg_extension
WHERE extname = 'uom';

 extname | extconfig |      extcondition
-----+-----+-----
 uom     | {uom_ref} | {"WHERE NOT predefined"}
(1 row)
```

Запускаем pg\_dump:

```
student$ pg_dump -d admin_extensions

--
-- PostgreSQL database dump
--

-- Dumped from database version 13.7 (Ubuntu 13.7-1.pgdg22.04+1)
-- Dumped by pg_dump version 13.7 (Ubuntu 13.7-1.pgdg22.04+1)

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

--
-- Name: uom; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS uom WITH SCHEMA public;

--
-- Name: EXTENSION uom; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION uom IS 'Единицы измерения. uom--1.2.sql';

--
-- Data for Name: uom_ref; Type: TABLE DATA; Schema: public; Owner: student
--

COPY public.uom_ref (name, k, predefined) FROM stdin;
фут      0.3048  f
дюйм     0.0254  f
\.

--
-- Name: TABLE uom_ref; Type: ACL; Schema: public; Owner: student
--

REVOKE SELECT ON TABLE public.uom_ref FROM PUBLIC;
GRANT SELECT ON TABLE public.uom_ref TO util;

--
-- PostgreSQL database dump complete
--
```

Проверим, как pg\_dump выгружает объекты расширения:

- Таблица, тип, функции и операторы не выгружаются. Они будут созданы командой CREATE EXTENSION. В системе, где производится восстановление, должна быть та же версия расширения, что при выгрузке.
- Содержимое вновь добавленных строк таблицы выгружается в виде команды COPY.
- Права доступа к таблице uom\_ref формируются такими, какими они были на момент запуска pg\_dump.

Для того чтобы правильно выгрузить права доступа, в системном каталоге сохраняется информация о правах на объекты, выданные скриптами создания расширений:

```
=> SELECT objoid::regclass, initprivs
FROM pg_init_privs
WHERE privtype = 'e';

 objoid |      initprivs
-----+-----
 uom_ref | {student=arwdDxt/student,=r/student}
(1 row)
```

Без этого было бы невозможно сформировать команду REVOKE.