

Задачи администрирования Управление расширениями



Авторские права

© Postgres Professional, 2016–2025

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов, Игорь Гнатюк

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

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

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, 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 расширений и управление ими

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

3

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 дает возможность 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/16/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.

Группа взаимосвязанных объектов БД

- установка всех объектов одной командой
- невозможность удалить отдельный объект
- сохранение связи при выгрузке с помощью `pg_dump`
- инструменты для перехода на новую версию

Источники

- в составе дистрибутива (`contrib`)
- внешние расширения
- возможность создания собственных расширений

Бывают ситуации, когда несколько объектов базы данных логически связаны между собой. Например, несколько типов данных, функции и операторы для работы с ними, классы операторов. Такую связь можно сделать явной с помощью механизма расширений.

Это облегчает управление объектами:

- все объекты устанавливаются одной командой;
- невозможно удалить отдельный объект — расширение можно удалить только полностью;
- связь между объектами сохраняется и при создании резервной копии с помощью утилиты `pg_dump`;
- есть инструменты для управления версиями расширений.

В состав PostgreSQL входит значительное количество полезных расширений, частью из которых мы уже пользовались.

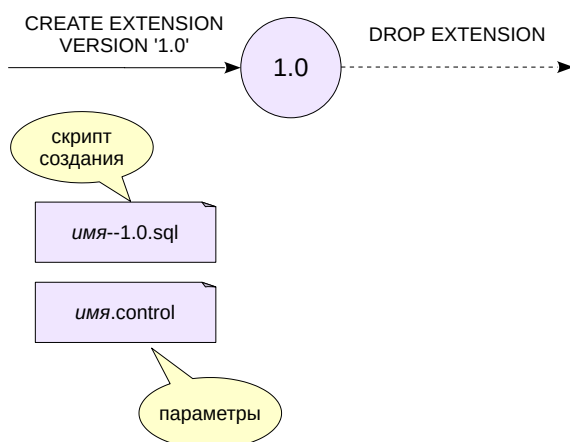
<https://postgrespro.ru/docs/postgresql/16/contrib>

<https://postgrespro.ru/docs/postgresql/16/contrib-prog>

Другой источник — PostgreSQL Extension Network (PGXN) — сеть расширений по аналогии с CPAN для Perl: <https://pgxn.org/>

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

Создание расширения



Расширение устанавливается в базу данных командой `CREATE EXTENSION`. При этом должны существовать два файла:

- **управляющий файл** «*имя.control*» с параметрами расширения;
- **скрипт создания** объектов расширения «*имя--версия.sql*».

Версия традиционно имеет вид «1.0», «1.1» и т. д., но это не обязательно: она может состоять из любых символов (но не должна содержать «--» и начинаться или заканчиваться на «--»).

Обычно версию не указывают в команде `CREATE EXTENSION`, поскольку текущая актуальная версия записана в управляющем файле (параметр `default_version`) и используется по умолчанию.

Другие параметры в управляющем файле указывают на возможность перемещения объектов расширения между схемами (`relocatable`), зависимости от других расширений (`requires`), возможность установки только суперпользователем (`superuser`) и пр.

<https://postgrespro.ru/docs/postgresql/16/extend-extensions>

<https://postgrespro.ru/docs/postgresql/16/sql-createextension>

Расширение удаляется командой `DROP EXTENSION`. Скрипт для удаления объектов не требуется.

<https://postgrespro.ru/docs/postgresql/16/sql-dropextension>

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 июн 23 19:17 Makefile
-rw-rw-r-- 1 student student 1132 июн 23 19:17 README.md
-rw-rw-r-- 1 student student 1644 июн 23 19:17 uom--1.0--1.1.sql
-rw-rw-r-- 1 student student 939 июн 23 19:17 uom--1.0.sql
-rw-rw-r-- 1 student student 755 июн 23 19:17 uom--1.1--1.2.sql
-rw-rw-r-- 1 student student 2794 июн 23 19:17 uom--1.2.sql
-rw-rw-r-- 1 student student 93 июн 23 19:17 uom.control
```

Среди файлов есть управляющий файл с настройками:

```
student$ cat /home/student/uom/uom.control
```

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

- default_version определяет версию по умолчанию, без этого параметра версию придется указывать явно;
- relocatable говорит о том, что расширение можно перемещать из схемы в схему (мы поговорим об этом чуть позже);
- encoding требуется, если используются символы, отличные от ASCII;
- comment определяет комментарий к расширению.

Это не все возможные параметры; полный список можно узнать из документации.

Файлы, общие для всех кластеров одной версии PostgreSQL, находятся в каталоге SHAREDIR:

```
student$ pg_config --sharedir
```

```
/usr/share/postgresql/16
```

В частности, файлы установленных расширений находятся в каталоге SHAREDIR/extension. Например, файлы расширения pg_buffercache:

```
student$ ls $(pg_config --sharedir)/extension/pg_buffercache*
```

```
/usr/share/postgresql/16/extension/pg_buffercache--1.0--1.1.sql
/usr/share/postgresql/16/extension/pg_buffercache--1.1--1.2.sql
/usr/share/postgresql/16/extension/pg_buffercache--1.2--1.3.sql
/usr/share/postgresql/16/extension/pg_buffercache--1.2.sql
/usr/share/postgresql/16/extension/pg_buffercache--1.3--1.4.sql
/usr/share/postgresql/16/extension/pg_buffercache.control
```

Для установки расширений обычно используется утилита make. Если расширение требует компиляции, а PostgreSQL установлен из пакета, дополнительно нужно установить пакет для разработчиков; его имя обычно содержит слово dev, например postgresql-server-dev-16. Пакет включает все необходимое для сборки, в том числе заголовочные файлы PostgreSQL.

Наше расширение содержит только функции на SQL и не требует компиляции и сборки, поэтому Makefile должен обеспечить только копирование файлов расширения в SHAREDIR/extension:

```
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)
```

- Переменная EXTENSION задает имя расширения;
- Переменная DATA определяет список файлов, которые надо скопировать в SHAREDIR (кроме управляющего);
- Последние строки не меняются, они подключают инфраструктуру PGXS для установки расширений. Важно, чтобы утилита pg_config была доступна — иначе неизвестны пути, по которым установлен PostgreSQL.

Установим расширение, выполнив make install в каталоге расширения:

```
student$ sudo make install -C /home/student/uom
```

```
make: Entering directory '/home/student/uom'
```

```

/bin/mkdir -p '/usr/share/postgresql/16/extension'
/bin/mkdir -p '/usr/share/postgresql/16/extension'
/usr/bin/install -c -m 644 ./uom.control '/usr/share/postgresql/16/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/16/extension/'
make: Leaving directory '/home/student/uom'

```

Установка расширения обычно выполняется суперпользователем, в окружении которого путь до исполняемых файлов PostgreSQL может быть не указан. В этом случае в команде установки нужно явно задать расположение утилиты pg_config:

```
student$ sudo make install -C /home/student/uom PG_CONFIG=/usr/lib/postgresql/16/bin/pg_config
```

Список расширений, установленных на сервере и доступных для загрузки в базы данных кластера, виден в системном каталоге:

```

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

```

name	default_version	installed_version
adminpack	2.1	
amcheck	1.3	
autoinc	1.0	
bloom	1.0	
btree_gin	1.3	
btree_gist	1.7	
citext	1.6	
cube	1.5	
dblink	1.2	
dict_int	1.0	
dict_xsyn	1.0	
earthdistance	1.1	
file_fdw	1.0	
fuzzystrmatch	1.2	
hstore	1.8	
insert_username	1.0	
intagg	1.1	
intarray	1.5	
isn	1.2	
lo	1.1	
ltree	1.2	
moddatetime	1.0	
old_snapshot	1.0	
pageinspect	1.12	
pg_buffercache	1.4	
pg_freespacemap	1.2	
pg_prewarm	1.2	
pg_stat_statements	1.10	
pg_surgery	1.0	
pg_trgm	1.6	
pg_visibility	1.2	
pg_wait_sampling	1.1	
pg_walinspect	1.1	
pgcrypto	1.3	
pgrowlocks	1.2	
pgstattuple	1.5	
plpgsql	1.0	1.0
postgres_fdw	1.1	
refint	1.0	
seg	1.4	
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-oss	1.1	
xml2	1.1	
(49 rows)		

В стандартных базах данных установлено только расширение plpgsql.

Список версий нашего 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.

Загрузим в новую базу данных версию 1.0 расширения uom:

```
=> 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
```

```
=> \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 появилось в базе данных, оно содержит таблицу и функцию:

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

```

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

Объекты установлены в схему, в которой они были бы созданы по умолчанию; в данном случае — public. При создании расширения мы могли бы указать схему явно:

```
CREATE EXTENSION uom SCHEMA public;
```

Функция 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)
```

А в таблице хранится список поддерживаемых единиц измерения:

```
=> 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)
```

Объекты расширения были созданы SQL-скриптом для версии 1.0:


```

student$ cat /usr/share/postgresql/16/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
RETURN 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 файла предотвращает случайный запуск скрипта вручную, он должен работать только при выполнении CREATE EXTENSION.
- При создании расширения все команды выполняются в одной транзакции, поэтому команды управления транзакциями (и служебные команды, такие, как VACUUM) здесь не допускаются.
- Путь поиска (параметр search_path) будет установлен на единственную схему — ту, в которой создаются объекты расширения.

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.

```

При этом механизм расширений не отслеживает изменения самих объектов. Например, добавление столбца в таблицу или изменение функции не вызовет ошибки. Вносить изменения в объекты расширения не следует.

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)

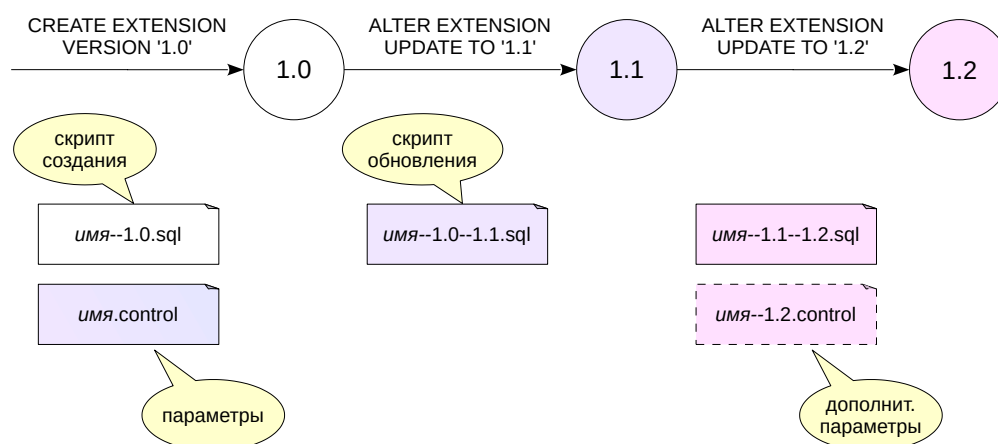
```

Вернем расширение в public:

```
=> ALTER EXTENSION uom SET SCHEMA public;
```

```
ALTER EXTENSION
```

Обновление расширения



7

Обновление версии расширения выполняется командой `ALTER EXTENSION UPDATE`. При этом должен существовать **скрипт обновления** «имя--старая-версия--новая-версия.sql», содержащий необходимые для обновления команды.

Также необходимо изменить управляющий файл «имя.control», обновив актуальную версию и, возможно, другие параметры.

При необходимости может существовать и отдельный управляющий файл, привязанный к версии. Например, если в версии 1.2 появилась зависимость от другого расширения, то эту зависимость лучше указать не в основном управляющем файле, а в дополнительном файле версии 1.2. Параметры, указанные в дополнительном управляющем файле, имеют приоритет над параметрами основного.

В примере, приведенном на слайде, показаны скрипты обновления 1.0 → 1.1 и 1.1 → 1.2. Можно создать и скрипт 1.0 → 1.2, но, как правило, это не требуется: механизм расширений сам берет на себя выбор пути обновления с учетом доступных переходов между версиями. Например, если установлена версия 1.0, то ее можно обновить сразу до 1.2: сначала автоматически применится скрипт 1.0 → 1.1, а затем 1.1 → 1.2.

Как и при создании, при обновлении номер версии обычно не указывают — в этом случае обновление происходит до последней актуальной версии, записанной в основном управляющем файле.

<https://postgrespro.ru/docs/postgresql/16/sql-alterextension>

Версии расширения и обновление

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

Файл с изменениями:

```
student$ cat /usr/share/postgresql/16/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:

- В базе данных есть предыдущая версия, нужно обновить расширение.
- Предыдущей версии нет, 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)
```

Сразу создаем расширение версии 1.1 в отдельной схеме, обеспечив видимость объектов в ней:

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

```
CREATE EXTENSION
```

```
=> SET search_path = public, util;
```

```
SET
```

```
=> \dx uom
```

```

              List of installed extensions
  Name | Version | Schema | Description
-----+-----+-----+-----
  uom  | 1.1     | util   | 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)
```

Можно построить индекс В-дерево:

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

CREATE INDEX

И если запретить планировщику последовательный доступ, новый индекс будет использован при выполнении запроса:

```
=> 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)
```

Утилита pg_dump

Утилита pg_dump выгружает только команду CREATE_EXTENSION, но не команды создания объектов расширения. Поэтому для восстановления из резервной копии нужно, чтобы на сервере были установлены все используемые расширения.

В следующей версии расширения 1.2 в таблицу добавлен столбец predefined. Он нужен для определения строк, которые были вставлены в таблицу не скриптом расширения, а уже после его создания.

Файл обновления:

```
student$ cat /usr/share/postgresql/16/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');
```

Скрипт вызывает функцию pg_extension_config_dump, чтобы утилита выгружала строки таблицы, удовлетворяющие условию.

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	util	Единицы измерения. uom--1.1--1.2.sql
(1 row)			

Теперь в логическую копию будут попадать, кроме CREATE EXTENSION, и данные таблицы uom_ref.

Если выгружать базу данных целиком, то в резервную копию попадают все установленные расширения и такое поведение обычно нас устраивает. Однако если при выгрузке указать схему, то установленные в ней расширения в копию не попадут:

```
student$ pg_dump -d admin_extensions --schema util | grep 'CREATE'
```

```
CREATE SCHEMA util;
```

Восстановить содержимое таблицы uom_ref из такой копии не получится.

Чтобы выгрузить определенные расширения, нужно указать их имена в ключе --extension. Но при этом придется позаботиться о зависимостях — если в нашем случае выгрузить только схему util, то зависящая от расширения таблица public.t в резервную копию не попадет:

```
student$ pg_dump -d admin_extensions --schema util --extension uom | grep 'CREATE'
```

```
CREATE SCHEMA util;
```

```
CREATE EXTENSION IF NOT EXISTS uom WITH SCHEMA util;
```

После создания расширения привилегии на его объекты сохраняются в системном каталоге. Если к моменту выгрузки привилегии изменились, pg_dump добавит в резервную копию команды для формирования новых привилегий.

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.

10

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

3. При добавлении данных важно, чтобы у новых 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
```

```
make: Entering directory '/home/student/uom'
/bin/mkdir -p '/usr/share/postgresql/16/extension'
/bin/mkdir -p '/usr/share/postgresql/16/extension'
/usr/bin/install -c -m 644 ./uom.control '/usr/share/postgresql/16/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/16/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
-----
12.000000000000000
(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 16.2 (Ubuntu 16.2-1ubuntu4)
-- Dumped by pg_dump version 16.2 (Ubuntu 16.2-1ubuntu4)
```

```
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.