Авторы: Егор Рогов, Павел Лузанов

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

Темы

Функции и их особенности в базах данных

Параметры и возвращаемое значение

Способы вызова функции

Перегрузка

Полиморфизм

Категории изменчивости

Подстановка в текст запроса

Ограничения

Функции в базе данных

Основной мотив: упрощение задачи

- интерфейс (параметры) и реализация (тело функции)
- о функции можно думать вне контекста всей задачи

	«Обычные» языки	*PostgreSQL*
побочные эффекты	глобальные переменные	вся база данных (категории изменчивости)
модули	со своим интерфейсом и реализацией	пространства имен, клиент и сервер
сложности	накладные расходы на вызов (подстановка)	скрытие запроса от планировщика (подстановка, подзапросы, представления)

Основная цель появления функций в программировании вообще — упростить решаемую задачу за счет ее декомпозиции на более мелкие подзадачи. Упрощение достигается за счет того, что о функции можно думать, абстрагировавшись от «большой» задачи. Для этого функция определяет четкий интерфейс с внешним миром (параметры и возвращаемое значение). Ее реализация (тело функции) может меняться; вызывающая сторона «не видит» этих изменений и не зависит от них. Этой идеальной ситуации может мешать глобальное состояние (глобальные переменные), и надо учитывать, что в случае БД таким состоянием является вся база данных.

В традиционных языках функции часто объединяются в модули (пакеты, классы для ООП и т. п.), имеющие собственный интерфейс и реализацию. Границы модулей могут проводиться более или менее произвольно. Для PostgreSQL есть жесткая граница между клиентской частью и серверной: серверный код работает с базой, клиентский — управляет транзакциями. Модули (пакеты) отсутствуют, есть только пространства имен.

Для традиционных языков единственный минус широкого использования функций состоит в накладных расходах на ее вызов. Иногда его преодолевают с помощью подстановки (inlining) кода функции в вызывающую программу. Для БД последствия могут быть более серьезные: если в функцию выносится часть запроса, планировщик перестает видеть «общую картину» и не может построить хороший план. В некоторых случаях PostgreSQL умеет выполнять подстановку; альтернативные варианты — использование подзапросов или представлений.

Функции

Функция — объект базы данных

Упрощенный общий вид объявления

```
CREATE [OR REPLACE] FUNCTION имя ( [параметр, параметр...] )
[ RETURNS тип ]
AS строка-тело-функции
LANGUAGE язык;
```

доступны несколько языков, в том числе чистый SQL

Другие операции с функциями

`ALTER FUNCTION` — изменение свойств

`DROP FUNCTION` — удаление

`\df` — просмотр имеющихся

`\ef` — редактирование

Функции являются объектами базы данных, наряду с таблицами и индексами.

В PostgreSQL доступно большое количество стандартных функций. С некоторыми из них можно познакомиться в справочном материале (datatypes.pdf), где они для удобства привязаны к типам данным, с которыми работают.

Кроме этого, можно писать и собственные функции на разных языках программирования.

Материал слайдов этой темы относится к любым функциям, независимо от языка программирования, но примеры в демонстрации будут на языке SQL.

И psql, и визуальные инструменты позволяют редактировать функции «непосредственно» в базе данных. Такой возможностью лучше не пользоваться (или как минимум не злоупотреблять). В нормально построенном процессе разработки весь код должен находиться в файлах под версионным контролем. При необходимости изменить функцию файл редактируется и выполняется (с помощью psql или средствами IDE). Если же менять определение функций сразу в БД, изменения легко потерять. (Вообще же вопрос организации процесса разработки намного сложнее и мы его не затрагиваем.)

Тело функции

Команды на выбранном языке программирования

- синтаксис зависит от языка

Задается в виде строки

- и в таком виде сохраняется в системном каталоге

Интерпретируется при вызове

Удобно заключать в доллары

```
' SELECT ''How''''s it going?''; '
$$ SELECT 'How''s it going?'; $$
$func$ SELECT $$How's it going?$$; $func$
```


Тело функции содержит команды на языке программирования, выбранном в предложении LANGUAGE.

Тело задается в виде строки. Эта строка сохраняется в системном каталоге и интерпретируется каждый раз, когда функция вызывается. В настоящий момент единственный способ избежать интерпретации — написать функцию на языке Си, но это требуется нечасто и в данном курсе эта тема не рассматривается.

Поскольку тело функции с большой вероятностью будет содержать апострофы и кавычки, удобно использовать синтаксис заключения строки в доллары.

Параметры

Общий вид

```
[режим] [имя] тип [ {DEFAULT|=} значение-по-умолчанию ]
```

Тип

любой тип SQL и некоторые псевдотипы

модификатор типа игнорируется

Режим

IN — входной

OUT — выходной

INOUT — входной и выходной

При определении функции указываются *формальные параметры* — они доступны в теле функции.

Имя параметра может опускаться, хотя это и не рекомендуется.

Можно задать режим использования параметра: как входной (по умолчанию), выходной или и как входной, и как выходной.

В качестве типа можно использовать любой тип SQL, а также некоторые *псевдотипы,* с некоторыми из которых мы познакомимся позже. Модификаторы типа игнорируются; таким образом, и varchar, и varchar(100) будут обозначать просто varchar.

Для параметра можно задать значение по умолчанию, которое будет использоваться, если при вызове функции не указать соответствующий *фактический параметр*.

Возвращаемое значение

Эквивалентные формы: RETURNS и выходные параметры

Конструкция RETURNS

`RETURNS` *тип*

`RETURNS void` при отсутствии возвращаемого значения

Выходные параметры

режим OUT или INOUT

Возвращаемое значение можно определить двумя способами.

- использовать отдельную конструкцию RETURNS для указания типа,

- определить выходные параметры (с режимом OUT или INOUT).

Две эти формы записи эквивалентны.

Например, функция с указанием RETURNS integer и функция с параметром OUT integer возвращают целое число. Можно использовать и оба способа одновременно. В этом случае функция также будет возвращать *одно* целое число. Но при этом типы RETURNS и выходных параметров должны быть согласованы друг с другом.

Если функция не должна ничего возвращать (в некоторых языках это называется *процедурой*), ее следует определить (в духе Си) как RETURNS void.

Вызов функции

Позиционный способ

фактические параметры указываются в порядке определения

параметры со значением по умолчанию можно опускать
(с правого конца списка)

По имени

фактические параметры указываются по имени в любом порядке

имя `=>` *значение*

параметры со значением по умолчанию можно опускать

Смешанный способ

позиционные параметры указываются перед именными

При вызове функции ей передаются *фактические параметры,* которые связываются с *формальными параметрами,* заданными при определении.

Поддерживаются три способа указания параметров.

Позиционный способ связывает фактические параметры с формальными по позиции, то есть параметры должны указываться в том же порядке, в котором они определялись.

При *именной передаче* фактические параметры связываются с формальными по имени. Порядок перечисления не играет в этом случае никакой роли.

Смешанный способ позволяет комбинировать два предыдущих, но позиционные параметры должны идти перед именными.

В любом случае можно опускать параметры, определенные со значением по умолчанию. Но в случае позиционной передачи их можно опускать только справа налево, чтобы не нарушать порядок следования.

Какой способ предпочесть? Для функций с несколькими параметрами, где порядок очевиден, удобно использовать позиционный способ. Во многих стандартных функциях не заданы имена параметров, и такой способ — единственно возможный. Но при большом числе параметров именной способ более предпочтителен.

https://postgrespro.ru/docs/postgresql/9.6/sql-syntax-calling-funcs

Перегруженные функции

Перегрузка

- определение нескольких функций с одним и тем же именем
- функции различаются типами входных параметров,
  а тип возвращаемого значения и выходные параметры игнорируются
- подходящая функция выбирается во время выполнения

Переопределение CREATE OR REPLACE

- при несовпадении типов входных параметров создаст новую
  перегруженную функцию
- при совпадении — изменит существующую функцию,
  но поменять тип возвращаемого значения нельзя

Перегрузка функций — это возможность использования одного и того же имени для нескольких функций, отличающихся типами параметров IN и INOUT. Иными словами, *сигнатура функции* — ее имя и типы входных параметров.

При вызове функции PostgreSQL находит ту функцию, которая соответствует переданным фактическим параметрам. Возможны ситуации, когда подходящую функцию невозможно определить однозначно; в таком случае во время выполнения возникнет ошибка.

Перегрузку надо учитывать при использовании команды CREATE OR REPLACE FUNCTION. Дело в том, что при несовпадении типов входных параметров будет создана новая — перегруженная — функция. Кроме того, эта команда не позволяет изменить тип возвращаемого значения (и типы выходных параметров).

Поэтому при необходимости следует удалять функцию и создавать ее заново. Но это будет уже новая функция; при удалении старой потребуется также удалить зависящие от нее представления, триггеры и т. п. (DROP FUNCTION ... CASCADE).

Полиморфные функции

Полиморфизм

способность функции работать с разными типами параметров и возвращать значение разных типов

в описании используются полиморфные псевдотипы (например, `anyelement`)

конкретный тип выбирается во время выполнения по типу фактических параметров

В некоторых случаях удобно не создавать несколько перегруженных функций для разных типов, а написать одну функцию, принимающую параметр любого (или почти любого) типа.

Для этого в качестве типа формального параметра указывается специальный *полиморфный псевдотип*. Пока мы ограничимся одним типом — anyelement, который соответствует любому базовому типу, — но позже познакомимся и с другими.

Конкретный тип, с которым будет работать функция, выбирается во время выполнения по типу фактического параметра.

Если у функции определено несколько полиморфных параметров, то типы соответствующих фактических параметров должны совпадать. Иными словами, anyelement при каждом вызове функции обозначает какой-то один конкретный тип.

Если функция объявлена с возвращаемым значением полиморфного типа, то она должна иметь по крайней мере один входной полиморфный параметр. Конкретный тип возвращаемого значения также определяется исходя из типа фактического входного параметра.

https://postgrespro.ru/docs/postgresql/9.6/extend-type-system.html

Категории изменчивости

Volatile

возвращаемое значение может произвольно меняться
при одинаковых значениях входных параметров

Stable

значение не меняется в пределах одного оператора SQL
функция не может менять состояние базы данных

Immutable

значение не меняется, функция детерминирована
функция не может менять состояние базы данных

Каждой функции сопоставлена категория изменчивости, которая определяет свойства возвращаемого значения при одинаковых значениях входных параметров.

Категория volatile говорит о том, что возвращаемое значение может произвольно меняться. Такие функции будут выполняться каждый раз при каждом вызове.

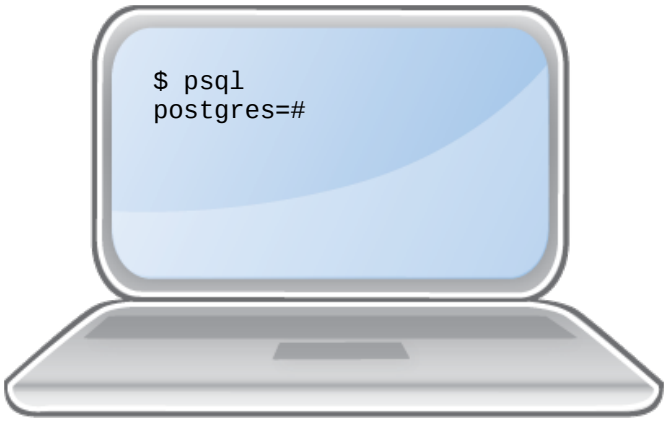
Категория stable используется для функций, возвращаемое значение которых не меняется в пределах одного SQL-оператора. В частности, такие функции не могут менять состояние БД. Такая функция *может* быть выполнена один раз во время выполнения запроса, а затем будет использоваться вычисленное значение.

Категория immutable еще более строгая: возвращаемое значение не меняется никогда. Такую функцию *можно* выполнить на этапе планирования запроса, а не во время выполнения.

Можно — не означает, что всегда происходит именно так, но планировщик вправе выполнить такие оптимизации. В некоторых (простых) случаях планировщик делает собственные выводы об изменчивости функции, невзирая на указанную явно категорию.

https://postgrespro.ru/docs/postgresql/9.6/xfunc-volatility

Демонстрация



Итоги

Можно создавать собственные функции
и использовать их так же, как и встроенные

Функции можно писать на разных языках, в том числе SQL

Поддерживаются перегрузка и полиморфизм

Изменчивость влияет на возможности оптимизации

Иногда функция на SQL может быть подставлена в запрос

Функции не могут управлять транзакциями

Практика

1. Создайте функцию author_name для формирования имени автора.
   Функция принимает три параметра (фамилия, имя, отчество) и возвращает строку с фамилией и инициалами.
2. Используйте эту функцию в представлении authors_v.
3. Создайте функцию book_name для формирования названия книги.
   Функция принимает два параметра (идентификатор книги и заголовок) и возвращает строку, составленную из заголовка и списка авторов в порядке seq_num. Имя каждого автора формируется функцией author_name.
4. Используйте эту функцию в представлении catalog_v.
5. Проверьте изменения в приложении.

1.

```
FUNCTION author_name(
    last_name text, first_name text, middle_name text
)
RETURNS text
```

Например:

```
author_name('Толстой','Лев','Николаевич') → 'Толстой Л. Н.'
```

3.

```
FUNCTION book_name(book_id integer, title text)
RETURNS text
```

Например:

```
book_name(3,'Трудно быть богом') →
→ 'Трудно быть богом. Стругацкий А. Н., Стругацкий Б. Н.'
```

Практика

Обратите внимание на категории изменчивости функций.

1. Напишите функцию, выдающую случайное время, равномерно распределенное в указанном отрезке.
   Начало отрезка задается временной отметкой (timestamptz), конец — либо временной отметкой, либо интервалом (interval).
2. В таблице хранятся номера автомобилей, введенные кое-как: встречаются как латинские, так и русские буквы в любом регистре; между буквами и цифрами могут быть пробелы.
   Считая, что формат номера «*буква три-цифры две-буквы*», напишите функцию, выдающую число уникальных номеров.
   Например, «K 123 XM» и «k123xm» считаются равными.
3. Напишите функцию, находящую корни квадратного уравнения.

2. Сначала напишите функцию «нормализации» номера, то есть приводящую номер к какому-нибудь стандартному виду. Например, без пробелов и только заглавными латинскими буквами.

В номерах используются 12 русских букв, имеющих латинские аналоги аналогичного начертания, а именно: АВЕКМНОРСТУХ.

3. Для уравнения $y = ax^2 + bx + c$:

дискриминант $D = b^2 – 4ac$.

- при $D > 0$ два корня $x_{1,2} = (–b \pm \sqrt{D}) / 2a$
- при $D = 0$ один корень $x = –b / 2a$ (в качестве x_2 можно вернуть null)
- при $D < 0$ корней нет (оба корня null).