Авторы: Егор Рогов, Павел Лузанов

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

Темы

Процедурные языки

PL/pgSQL

Структура блока

Объявление переменных

Функции PL/pgSQL

Анонимные блоки

Условные операторы и циклы

Вычисление выражений

Процедурные языки

Назначение

дополнить SQL процедурными возможностями

Создание

```
CREATE [ TRUSTED ] LANGUAGE имя
     HANDLER обработчик_вызова
   [ INLINE обработчик_внедренного_кода ]
   [ VALIDATOR функция_проверки ];
```

установка из расширения

Доступные языки

в дистрибутиве: **PL/pgSQL**, PL/Tcl, PL/Perl, PL/Python

дополнительно: PL/Java, PL/V8, PL/R, PL/PHP, …

Процедурные языки дополняют декларативный язык SQL процедурными возможностями. Что позволяет создавать пользовательские функции не только на C или чистом SQL.

Для подключения нового процедурного языка используется команда CREATE LANGUAGE. Сервер базы данных не знает как именно интерпретировать код процедурного языка, вместо этого он передает код *обработчику_вызова*. Обработчик вызова — это специальная функция на C, которая понимает этот язык. При создании языка, вместе с обработчиком вызова можно также указать функцию для выполнения анонимных блоков (*обработчик_внедренного_кода*) и функцию для проверки кода программы (*функция_проверки*).

Обычно добавление нового языка выполняется установкой соответствующего расширения.

Процедурные языки должны устанавливаться в каждую базу данных, в которой планируется их использование. Список установленных в БД языков можно проверить в таблице pg_language.

Общая информация о процедурных языках:
https://postgrespro.ru/docs/postgresql/9.6/xplang.html

Некоторые процедурные языки, не входящие в состав дистрибутива:
https://postgrespro.ru/docs/postgresql/9.6/external-pl.html

Доверенные языки

Доверенные

- ограниченная функциональность в части взаимодействия с ОС
- по умолчанию доступ для всех пользователей

Недоверенные

- полная функциональность языка
- доступ только у суперпользователей
- обычно к названию языка добавляют «u»: plperl → plperlu

Необязательное ключевое слово TRUSTED в команде CREATE LANGUAGE указывает на то, что процедурный язык является доверенным.

Для доверенных языков ограничена функциональность, связанная с взаимодействием с ОС: работа с файлами, процессами и т. д.

Использование такого языка является безопасным, поэтому доступ к нему получают все пользователи (для роли PUBLIC выдается привилегия USAGE).

Недоверенные языки не имеют ограничений на взаимодействие с ОС. Функция на таком языке может выполнять любые операции в ОС с правами пользователя, запустившего сервер базы данных. Это может быть небезопасным, поэтому доступ к такому языку имеют только суперпользователи PostgreSQL.

Отметим, что обычные пользователи могут получить доступ не к самому языку, но к функциям на недоверенном языке. Для этого суперпользователь создает такую функцию с указанием SECURITY_DEFINER и выдает право на ее исполнение нужным пользователям. Подробнее о разграничении доступа к функциям рассказывается в модуле «Разграничение доступа».

Преимущества

Производительность

- меньше команд
- меньше пересылки данных
- меньше разборов команд

API к объектам БД

- гибче управлять согласованностью данных
- гибче управлять доступом к данным

Работа с внешними данными

Использование процедурных языков на сервере дает ряд преимуществ в дополнение к возможностям чистого SQL.

- Можно значительно повысить производительность за счет:
  - Меньше команд: исключаются дополнительные обращения между клиентом и сервером
  - Меньше пересылки данных: промежуточные ненужные результаты не передаются между сервером и клиентом
  - Меньше разборов команд: автоматическое использование подготовленных операторов позволяет избежать многочисленных разборов одного и того же запроса.

  Однако использование процедурных языков в тех случаях, когда достаточно чистого SQL, может существенно снизить производительность работы приложения.
- Хранимые функции позволяют создавать API для работы с объектами БД. Клиентское приложение, вызывая хранимую функцию, не обязано знать о том, какие действия и с какими объектами нужно выполнить. А права доступа можно настроить так, что пользователи смогут выполнять функции, но не иметь доступа к объектам, с которыми эти функции работают.
- Недоверенные языки позволяют получить доступ к внешним данным из хранимых функций.

Обратная сторона в том, что перенос логики приложения на сервер базы данных усиливает зависимость от используемой СУБД.

PL/pgSQL

Появился в версии 6.4 в 1998 году

- устанавливается по умолчанию с версии 9.0

Цели создания языка

- используется для написания функций и триггеров
- добавляет управляющие структуры к языку SQL
- позволяет использовать любые пользовательские типы, функции и операторы
- доверенный язык
- простота использования

На основе Oracle PL/SQL

PL/pgSQL — один из первых процедурных языков в PostgreSQL. Он впервые появился в 1998 году в версии 6.4. А с версии 9.0 стал устанавливаться по умолчанию при создании БД.

PL/pgSQL дополняет SQL, предоставляя такие возможности процедурных языков, как использование переменных и курсоров, условные операторы, циклы, обработка ошибок и т. д.

PL/pgSQL проектировался на основе языка Oracle PL/SQL и имеет с ним похожий синтаксис.

https://postgrespro.ru/docs/postgresql/9.6/plpgsql-overview.html

Структура блока

Блочно-структурированный язык

```
[ <<метка>> ]
[ DECLARE
      объявления ]
  BEGIN
      операторы
[ EXCEPTION
      обработка_ошибок ]
  END [ метка ];
```

Важно:

`BEGIN` — начало исполняемой секции в PL/pgSQL-блоке

`BEGIN;` — команда SQL, открывающая транзакцию

PL/pgSQL — это блочно-структурированный язык.

В структуре блока можно выделить:

- Необязательную секцию, предназначенную для *объявления* локальных переменных и курсоров.
- Основную секцию исполнения, в которой располагаются *операторы*.
- Необязательную секцию *обработки ошибок.*

В качестве операторов можно использовать команды PL/pgSQL, а также большинство команд SQL. Причем важно, что SQL и PL/pgSQL интегрированы бесшовно: команды SQL используются в блоке напрямую. В качестве оператора может выступать и другой (вложенный) PL/pgSQL блок.

Каждому блоку можно присвоить *метку*, которая поможет отличить переменную блока от столбца таблицы с таким же именем. Или во вложенном блоке отличить переменные с одинаковыми именами для внешнего и вложенного блоков.

В качестве операторов нельзя использовать команды управления транзакциями (например, COMMIT, ROLLBACK), а также команды, которые нельзя использовать внутри транзакции (например, VACUUM).

https://postgrespro.ru/docs/postgresql/9.6/plpgsql-structure.html

Объявление переменных

Общий вид

```
имя [CONSTANT] тип [NOT NULL]
    [ {DEFAULT|:=|=} значение-по-умолчанию ];
```

Тип

любой SQL тип и некоторые псевдотипы

наследование типов: `%ROWTYPE, %TYPE`

Ограничения

`CONSTANT` — значение нельзя изменить

`NOT NULL` — значение не может быть `NULL`

Область действия

текущий блок, включая вложенные блоки и секцию обработки ошибок

Переменные для использования в блоке определяются в секции объявления.

В качестве типа можно указать любой тип, определенный в SQL.

Специальная конструкция вида *table*%ROWTYPE может быть использована для объявления переменной составного типа со структурой указанной таблицы.

Конструкция вида *table.column*%TYPE может быть использована для переменной с таким же типом, как и столбец таблицы.

Для переменной с ограничением NOT NULL нужно обязательно указать *значение-по-умолчанию*.

В секции DECLARE могут также объявляться курсоры. Подробнее о работе с курсорами в следующих темах.

Функции PL/pgSQL

Функция с входными параметрами

```
CREATE FUNCTION имя (IN параметр тип [, ...]) RETURNS тип
AS $$
[DECLARE ...]
 BEGIN
     ...
     RETURN значение;
 END;
$$ LANGUAGE plpgsql;
```

RETURN — завершение и возврат значения

Мы уже познакомились с хранимыми функциями в теме о функциях на языке SQL. Большинство рассмотренных вопросов, связанных с созданием и управлением функциями, относится и к функциям на PL/pgSQL:

- создание, изменение, удаление функций;
- расположение в системном каталоге (pg_proc);
- параметры и возвращаемое значений;
- категории изменчивости и т. д.

Для возврата значения из PL/pgSQL-функции используется оператор RETURN.

Выходные параметры

Функция с выходными параметрами

```
CREATE FUNCTION имя (INOUT параметр1 тип, OUT параметр2 тип)
AS $$
[DECLARE ...]
 BEGIN
     ...
     параметр1 := value;
     параметр2 := value;
 END;
$$ LANGUAGE plpgsql;
```

Возврат значения — присвоение значений выходным параметрам

Любой выходной параметр (OUT, INOUT) включается в состав возвращаемого значения функции.

В этом случае, вместо оператора RETURN, нужно присвоить значения выходным параметрам.

Анонимные блоки

Задача

выполнение процедурного кода без создания функции

Реализация

```
DO [ LANGUAGE имя_языка ] код;
```

имя_языка — по умолчанию plpgsql

код — строка с кодом программы

Особенности

не сохраняется в БД

нет прямой возможности передать параметры и вернуть значение

Для использования PL/pgSQL не обязательно создавать функции. Код на PL/pgSQL можно оформить и выполнить как анонимный блок, при помощи команды DO.

Эту команду SQL можно использовать с любым процедурным языком, для которого в CREATE LANGUAGE был определен *обработчик_внедренного_кода* (во фразе INLINE).

Если в команде DO язык не указан, то будет использоваться PL/pgSQL.

Код анонимного блока, в отличие от функций, не сохраняется на сервере. Также, нет возможности передать в анонимный блок параметры или вернуть из него значение. Хотя косвенно это возможно сделать, например, через таблицы.

https://postgrespro.ru/docs/postgresql/9.6/sql-do.html

Условные операторы

Три формы:

```
IF условие THEN
    операторы
[ELSIF условие THEN
    операторы]
[ELSE
    операторы]
END IF;
```

```
CASE
    WHEN условие THEN
        операторы
   [WHEN условие THEN
        операторы]
   [ELSE
        операторы]
END CASE;
```

```
CASE выражение
    WHEN значение [,...] THEN
        операторы
   [WHEN значение [,...] THEN
        операторы]
   [ELSE
        операторы]
END CASE;
```


PL/pgSQL предлагает два условных оператора: IF и CASE.

Если *условие* в IF истинно, то выполняется заданный набор *операторов*. Можно определить несколько *условий* (через ELSIF), которые будут проверяться последовательно, до тех пор, пока какое-либо не даст истину. В этом случае, выполняются *операторы*, связанные с этим *условием*. Через ELSE можно определить *операторы*, которые нужно выполнить, если ни одно из условий не дало истину.

Если ELSE не определено и ни одно из *условий* не является истинным, то управление переходит к следующему после IF оператору.

Нужно помнить о том, что логические выражения могут возвращать три значения: true, false и null. *Условие* срабатывает, только когда оно истинно, и не срабатывает, когда ложно или не определено.

Условный оператор CASE имеет в PL/pgSQL две формы: по *условию* и по *выражению*.

Обе конструкции похожи на CASE в SQL, за исключением:

- Оператор завершается END CASE, а не просто END.
- Вторая форма CASE разрешает перечисление нескольких *значений* после WHEN. Реализация в SQL допускает только одно значение.

В обеих формах CASE, при невыполнении всех условий и отсутствии ELSE будет выдана ошибка (в отличие от IF, где выполнение продолжается со следующего оператора).

Циклы

Цикл FOR по диапазону чисел

```
<<метка>>
FOR имя IN низ .. верх [BY инкремент]
LOOP
    операторы
END LOOP;
```

```
<<метка>>
FOR имя IN REVERSE верх .. низ [BY инкремент]
LOOP
    операторы
END LOOP;
```

Особенности

переменная *имя* создается автоматически

возможно обращение *метка.имя*

с указанием REVERSE нижняя и верхняя границы меняются местами

Для повторного выполнения набора операторов, PL/pgSQL предлагает несколько видов циклов.

Цикл FOR по диапазону чисел.

В начале выполнения, счетчик цикла (*имя*) инициализируется значением нижней границы (*низ*). Переменную-счетчик не нужно объявлять в секции DECLARE, она объявляется автоматически. Циклу можно присвоить метку. Это позволит обращаться к переменной-счетчику с указанием метки: *метка.имя.*

С каждой последующей итерацией, счетчик увеличивается на 1 (если не задано другое значение *инкремента* в BY). *Операторы* цикла выполняются до тех пор, пока счетчик не превысит верхнюю границу (*верх*).

При указании REVERSE значение счетчика с каждой итерацией уменьшается, а цикл выполняется до тех пор, пока значение счетчика не станет меньше нижней границы. Чтобы счетчик инициализировался значением *верх*, нужно нижнюю и верхнюю границы цикла поменять местами.

Цикл FOR может работать не только по диапазону чисел, но и по результатам выполнения запроса. Этот вариант цикла FOR будет рассмотрен в следующих темах.

Циклы

Цикл WHILE

```
<<метка>>
WHILE условие
LOOP
    операторы
END LOOP;
```

Особенности

повторяется, пока истинно *условие*

В цикле WHILE выполнение *операторов* продолжается до тех пор, пока истинно *условие*.

Циклы

Безусловный цикл

```
<<метка>>
LOOP
    ...
    EXIT [метка] [WHEN условие];
    ...
   [CONTINUE [метка] [WHEN условие];]
    ...
END LOOP;
```

Особенности

`EXIT` — выход из цикла

`CONTINUE` — переход на новую итерацию

эти команды работают и для других циклов

Цикл LOOP представляет собой вариант цикла, конструкция которого не определяет условие выхода.

Для выхода из такого цикла используется оператор EXIT. *Условие* выхода можно указать во фразе WHEN (EXIT WHEN) или использовать безусловный EXIT внутри команды IF.

Оператор CONTINUE используется для того, чтобы сразу перейти на следующую итерацию цикла, пропустив выполнение операторов между CONTINUE и END LOOP.

Оба оператора (EXIT и CONTINUE) поддерживают указание метки цикла. Это может быть полезно при использовании вложенных циклов.

Также отметим, что EXIT и CONTINUE можно использовать с любыми типами циклов, не только LOOP.

Вычисление выражений

Любое выражение вычисляется в контексте SQL

- выражение автоматически преобразуется в запрос
- запрос подготавливается
- переменные PL/pgSQL подставляются как параметры

Особенности

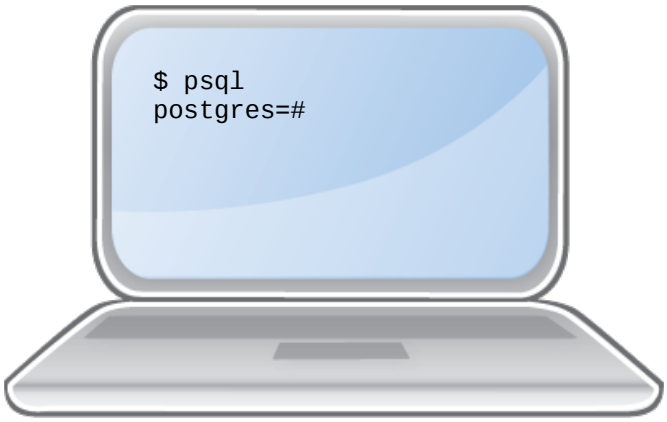
- можно использовать все возможности SQL, включая подзапросы
- невысокая скорость выполнения,
  хотя разобранный запрос (и, возможно, план запроса) кэшируются
- неоднозначности при разрешении имен требуют внимания

Любые выражения, которые встречаются в PL/pgSQL-коде, вычисляются с помощью SQL-запросов к базе данных. Интерпретатор сам строит нужный запрос, подготавливает оператор (при этом разобранный запрос кэшируется, как это обычно и происходит с подготовленными операторами) и выполняет его.

С одной стороны, это не способствует скорости работы PL/pgSQL, зато обеспечивает теснейшую интеграцию с SQL. Фактически, в выражениях можно использовать любые возможности SQL без ограничений, включая вызов встроенных и пользовательских функций, выполнение подзапросов и т. п.

https://postgrespro.ru/docs/postgresql/9.6/plpgsql-expressions.html

Демонстрация



Итоги

PL/pgSQL — доступный по умолчанию, доверенный, простой в использовании язык

Управление функциями на PL/pgSQL не отличается от работы с функциями на других языках

DO — команда SQL для выполнения анонимного блока

Переменные PL/pgSQL могут использовать любые типы SQL

Язык поддерживает условные операторы: IF, CASE и циклы: FOR, WHILE, LOOP

Практика

1. Измените функцию book_name так, чтобы длина возвращаемого значения не превышала 45 символов. Если название книги при этом обрезается, оно должно завершаться на троеточие.
2. Проверьте реализацию в SQL и в приложении; при необходимости добавьте книг с длинными названиями.
3. Снова измените функцию book_name так, чтобы избыточно длинное название уменьшалось на целое слово.
4. Проверьте реализацию.

1. Например:

```
Путешествия в некоторые удалённые страны мира в четырёх частях: сочинение Лемюэля Гулливера, сначала хирурга, а затем капитана нескольких кораблей →

→ Путешествия в некоторые удалённые страны м...
```

Вот некоторые случаи, которые имеет смысл проверить:

- длина названия меньше 45 символов (не должно измениться);

- длина названия ровно 45 символов (не должно измениться);

- длина названия 46 символов (от названия должны быть отрезаны 4 символа, т. к. добавятся еще три точки).

Лучше всего написать и отладить отдельную функцию укорачивания, которую затем использовать в book_name. Это полезно и по другим соображениям:

- такая функция может пригодиться где-то еще;

- каждая функция будут выполнять ровно одну задачу.

3. Например:

```
Путешествия в некоторые удалённые страны мира в четырёх частях: сочинение Лемюэля Гулливера, сначала хирурга, а затем капитана нескольких кораблей →

→ Путешествия в некоторые удалённые страны...
```

Как поведет себя ваша реализация, если название состоит из одного длинного слова без пробелов?

Практика

1. Напишите PL/pgSQL-функцию, которая возвращает произвольную строку заданной длины.
2. Задача про игру в «наперстки».

   В одном из трех наперстков спрятан выигрыш.
   Игрок выбирает один из этих трех. Ведущий убирает один из двух оставшихся наперстков (обязательно пустой) и дает игроку возможность поменять решение, то есть выбрать второй из двух оставшихся.

   Есть ли смысл игроку менять выбор или лучше оставить первоначальный вариант?

   Задание: используя PL/pgSQL, посчитайте вероятность выигрыша и для начального выбора, и для измененного.

0. Предварительно можно создать функцию rnd_integer, которая возвращет случайное целое число в заданном диапазоне. Функция будет полезна для решения обоих заданий.

Например: `rnd_integer(30, 1000) → 616`

1. Помимо длины строки на вход функции можно подавать список допустимых символов. По умолчанию, это могут быть все символы алфавита, числа и некоторые знаки. Для определения случайных символов из списка можно использовать функцию rnd_integer. Объявление функции может быть таким:

```
CREATE FUNCTION rnd_text(
   len int,
   list_of_chars text DEFAULT
'АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэю
яABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz_0123456789'
) RETURNS text AS ...
```

Пример вызова: `rnd_text(10) → 'ЛждфbЁ_OOJ'`

2. Для решения можно использовать анонимный блок.

Сначала нужно реализовать одну игру и посмотреть, какой вариант выиграл: начальный или измененный. Для загадывания и угадывания одного из трех наперстков можно использовать rnd_interger(1,3).

Затем игру поместить в цикл и «сыграть», например, 1000 раз, подсчитывая, какой вариант сколько раз победил. В конце через RAISE NOTICE вывести значения счетчиков и выявить победивший вариант (или отсутствие такового).