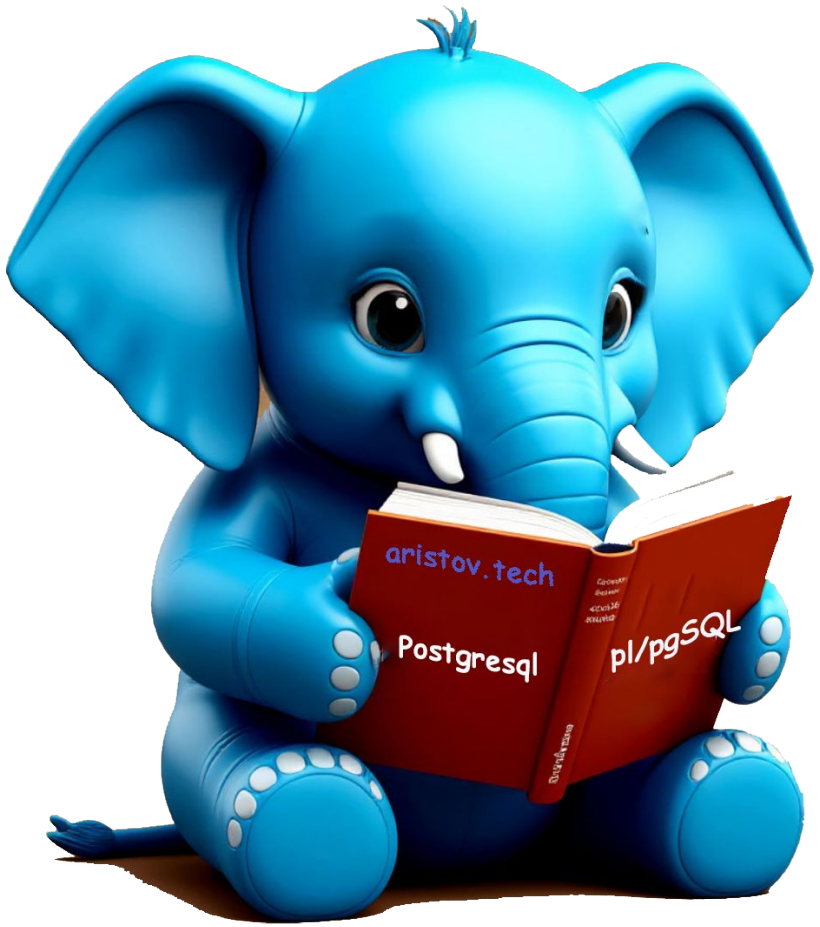
Аристов Евгений

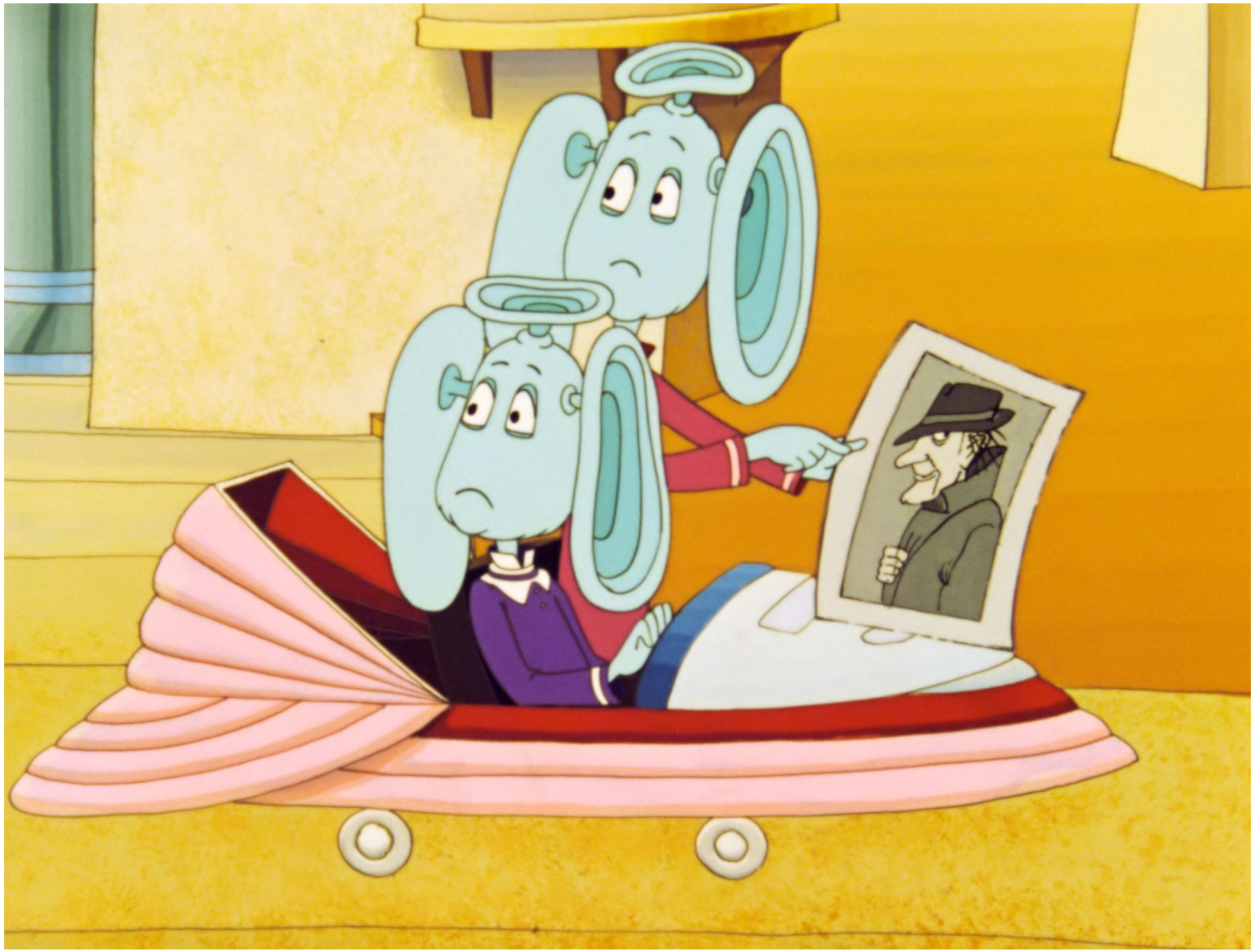
PL/pgSQL в PostgreSQL

за 31 занятие



Структура функции

https://aristov.tech



Аристов Евгений Николаевич

Founder & CEO aristov.tech
25 лет занимаюсь разработкой БД и ПО

Архитектор высоконагруженных баз данных и инфраструктуры

Спроектировал и разработал более ста проектов для финансового сектора, сетевых магазинов, фитнес-центров, отелей.

Сейчас решаю актуальные для бизнеса задачи: аудит и оптимизация БД и инфраструктуры, миграция на PostgreSQL, обучение сотрудников.

Автор более 10 практических курсов по PostgreSQL, MySQL, Mongo и др..

Автор книг по PostgreSQL. Новинка **PostgreSQL 16: лучшие практики оптимизации**

Правила вебинара

Задаем вопрос в чат

Вопросы вижу, отвечу в момент логической паузы

Если есть вопрос голосом - поставьте знак ? в чат

Если остались вопросы, можно их задать на следующем занятии или в комментариях к записи

https://aristov.tech

Маршрут вебинара

Определение функции

Синтаксис

Параметры

Функция

Определение функции

Функции, написанные на PL/pgSQL, определяются на сервере командами CREATE FUNCTION. Такая команда обычно выглядит, например, так:

```
CREATE FUNCTION somefunc(integer, text) RETURNS integer
AS 'тело функции'
LANGUAGE plpgsql;
```

Если рассматривать CREATE FUNCTION, то тело функции представляет собой просто текстовую строку. Часто для написания тела функции удобнее **заключать** эту строку в **двойные доллары**, а не в обычные апострофы.

PostgreSQL: Documentation: 17: 4.1. Lexical Structure

Если не применять заключение в доллары, все апострофы или обратные косые черты в теле функции придётся экранировать, дублируя их.

Синтаксис функции

CREATE [OR REPLACE] **FUNCTION**
  имя ([[*режим_аргумента*] [*имя_аргумента*] *тип_аргумента* [{ DEFAULT | = } *выражение_по_умолчанию*] [, ...]])
  [**RETURNS (не в голом SQL)** *тип_результата*
    | RETURNS TABLE (*имя_столбца тип_столбца* [, ...])]
 { **LANGUAGE** *имя_языка*
      ...
 } ...

https://www.postgresql.org/docs/current/sql-createfunction.html

https://aristov.tech

Параметры функции

Параметры функции:

- **IMMUTABLE**
- **STABLE**
- **VOLATILE (default)**

IMMUTABLE показывает, что функция не может модифицировать базу данных и всегда возвращает один и тот же результат при определённых значениях аргументов.

Она не обращается к базе данных и не использует информацию, не переданную ей явно в списке аргументов. Если функция имеет такую характеристику, любой её вызов с аргументами-константами можно немедленно заменить значением функции.

https://aristov.tech

Параметры функции

STABLE показывает, что функция не может модифицировать базу данных и в рамках одного сканирования таблицы она всегда возвращает один и тот же результат для определённых значений аргументов, но этот результат может быть разным в разных операторах SQL.

Хороший выбор для функций, результаты которых зависят от содержимого базы данных и настраиваемых параметров (например, текущего часового пояса). (Но этот вариант не подходит для триггеров AFTER, желающих прочитать строки, изменённые текущей командой.)

https://aristov.tech

Параметры функции

VOLATILE показывает, что результат функции может меняться даже в рамках одного сканирования таблицы, так что её вызовы нельзя оптимизировать.

Изменчивы в этом смысле относительно немногие функции баз данных, например: random(), currval() и timeofday().

Важно, что любая функция с побочными эффектами должна быть классифицирована как изменчивая, даже если её результат вполне предсказуем, чтобы её вызовы не были соптимизированы; пример такой функции: setval().

Более подробно с примерами на 11 лекции

Еще параметры функции

- **CALLED ON NULL INPUT**
- **RETURNS NULL ON NULL INPUT**
- **STRICT**

CALLED ON NULL INPUT (default) показывает, что функция будет вызвана как обычно, если среди её аргументов оказываются значения NULL. В этом случае ответственность за проверку значений NULL и соответствующую их обработку ложится на разработчика функции

RETURNS NULL ON NULL INPUT показывает, что функция всегда возвращает NULL, получив NULL в одном из аргументов. Такая функция не будет вызываться с аргументами NULL, вместо этого автоматически будет полагаться результат NULL

STRICT аналогичен указанию **RETURNS NULL ON NULL INPUT**, обычно пишут полностью, чтобы не путаться

Еще ВАЖНЫЕ параметры функции

- **SECURITY INVOKER**
- **SECURITY DEFINER**

SECURITY INVOKER - default (безопасность вызывающего) показывает, что функция будет выполняться с правами пользователя, вызвавшего её.

SECURITY DEFINER (безопасность определившего) определяет, что функция выполняется с правами пользователя, владеющего ей.

Интересная особенность при вызове суперюзером функции SECURITY DEFINER с ограниченными правами - доступ мы не получим)

https://aristov.tech

Еще параметры функции

- **PARALLEL**

PARALLEL UNSAFE (default) означает, что эту функцию нельзя выполнять в параллельном режиме и присутствие такой функции в операторе SQL приводит к выбору последовательного плана выполнения.

PARALLEL RESTRICTED означает, что функцию можно выполнять в параллельном режиме, но только в ведущем процессе группы. PARALLEL **SAFE** показывает, что функция безопасна для выполнения в параллельном режиме без ограничений.

SAFE - можно параллелить используя максимальное количество max_parallel_workers. *Нужно понимать, что параллелизация не всегда есть хорошо.*

https://aristov.tech

Выбор языка

- **LANGUAGE**

PL/pgSQL (Chapter 41),

PL/Tcl (Chapter 42)

PL/Perl (Chapter 43)

PL/Python (Chapter 44)

Более подробно на следующей лекции.

список языков:
https://www.postgresql.org/docs/current/xplang.html

почитать:
PostgreSQL: Серверное программирование на «человеческом» языке (PL/Perl, PL/Python, PL/v8) / Хабр

https://aristov.tech

Общие замечания

- Ключевые слова не чувствительны к регистру символов. Как и в обычных SQL-командах, идентификаторы неявно преобразуются к нижнему регистру, если они не взяты в двойные кавычки.
- Комментарии в PL/pgSQL коде работают так же, как и в обычном SQL. Двойное тире (--) начинает комментарий, который завершается в конце строки. Блочный комментарий начинается с /* и завершается */. Блочные комментарии могут быть вложенными.
- Используйте читабельный синтаксис - не забывайте про отступы.

https://aristov.tech

Ограничения

1. Ограничения на размер исходного кода

- **Размер текста функции:** Код функции хранится в системном каталоге pg_proc в столбце prosrc типа text. Теоретически, размер поля text может достигать **1 ГБ**. Однако на практике такой огромный скрипт будет абсолютно непрактичным.

2. Практические ограничения и проблемы

Хотя технически можно создать гигантский скрипт, на пути встают практические ограничения:

- **Производительность компиляции:** Перед первым выполнением код PL/pgSQL **компилируется** в дерево выражений (expression tree). Чем больше скрипт, тем дольше компиляция и тем больше память она consumes.
- **Потребление памяти:** Большие функции, особенно те, что используют много переменных, сильнее нагружают сервер. Каждая сессия, выполняющая такую функцию, будет хранить ее скомпилированное представление в своей памяти.
- **Сложность отладки и поддержки:** Скрипт размером в несколько мегабайт (десятки тысяч строк) будет кошмаром для понимания, отладки и изменения. Это противоречит принципам хорошего стиля программирования.
- **Ограничение на глубину стека:** Очень большие функции с глубокой вложенностью блоков и операторов могут превысить лимит глубины стека (max_stack_depth), что приведет к ошибке.

Лучшие практики

Если ваш скрипт разрастается до больших размеров, это верный признак того, что его нужно разбить на части:

1. **Разделение на несколько функций:** Вынесите логические блоки кода в отдельные, более мелкие функции. Это улучшит читаемость, упростит тестирование и позволит повторно использовать код.
2. **Динамический SQL:** Для очень больших операций, которые генерируются кодом, иногда лучше строить и выполнять динамический SQL, чтобы не хранить его как статический текст.
3. **Внешние скрипты:** возможно есть смысл хранить исходный код функций в файлах системы контроля версий (Git)

https://aristov.tech

Практика

Итоги

Итоги

Остались ли вопросы?

Увидимся на следующем занятии

Спасибо за внимание!

Когда дальше и куда?

Аристов Евгений