Авторы: Егор Рогов, Павел Лузанов

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

Темы

Использование команд SQL

Подстановка переменных

Проверка статуса команды

Табличные функции

Команды без результата

Команды SQL встраиваются в код PL/pgSQL

- как и в выражениях:
  запрос подготавливается,
  переменные PL/pgSQL подставляются как параметры

SELECT → PERFORM

- удобно для вызова функций с побочными эффектами
- запросы, начинающиеся на WITH, надо «оборачивать» в SELECT

INSERT, UPDATE, DELETE и другие команды SQL

- кроме команд управления транзакциями
  и нетранзакционных (служебных) команд

Как мы уже видели, PL/pgSQL очень тесно интегрирован с SQL. В частности, все выражения вычисляются с помощью подготовленных SQL-операторов к базе данных. При этом в выражениях можно использовать переменные PL/pgSQL и параметры функций — они подставляются в запрос в виде параметров.

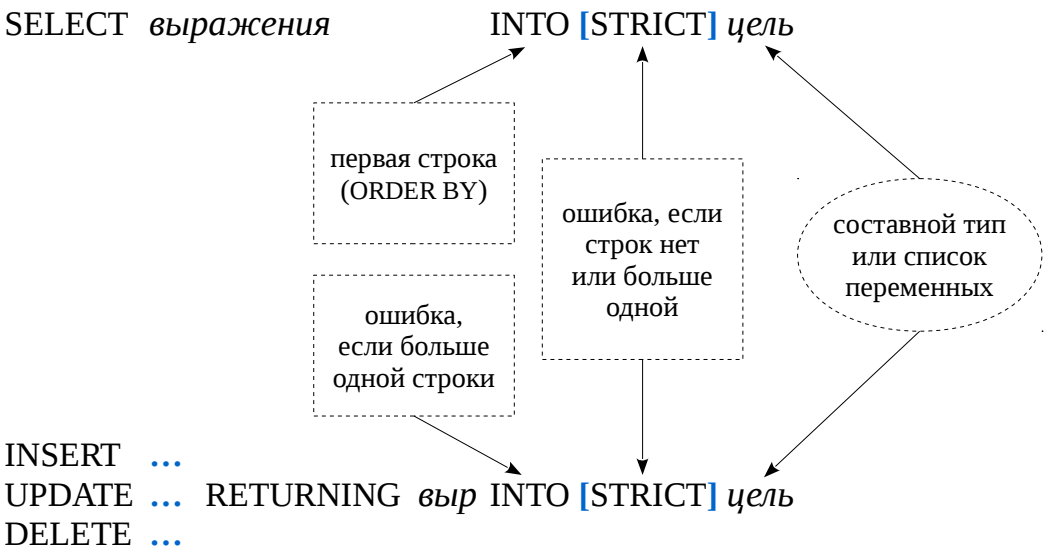
Внутри кода на PL/pgSQL можно выполнить и запрос SQL, не возвращающий результат (такие команды, как INSERT, UPDATE, DELETE, CREATE, DROP и т. п.). Для этого достаточно просто написать команду SQL внутри кода на PL/pgSQL как отдельный оператор.

Команды подготавливаются точно так же, как и выражения. Это позволяет закэшировать разобранный (или спланированный) запрос и не выполнять эту часть работы повторно. Внутри команд так же можно использовать переменные PL/pgSQL, которые будут автоматически заменены на параметры.

Таким же образом можно вызвать и обычный запрос SELECT, заменив это ключевое слово на PERFORM. Это обычно используется для вызова функций с побочным эффектом, потому что иначе в такой команде нет смысла. Если запрос начинается с WITH, его необходимо «обернуть» в SELECT (чтобы в конечном итоге запрос начинался на PERFORM).

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

Одна строка результата



Если результат запроса важен, его можно получить с помощью фразы INTO в переменную составного типа (или в несколько скалярных переменных). Если запрос возвращает несколько строк, в переменную попадет только первая из них (порядок можно гарантировать фразой ORDER BY). Если запрос не возвращает ни одной строки, переменная получит неопределенное значение.

Дополнительная фраза STRICT гарантирует ровно одну строку — иначе будет зафиксирована ошибка.

Аналогично можно использовать команды INSERT, UPDATE, DELETE с фразой RETURNING. Отличие состоит в том, что эти команды не должны возвращать более одной строки — это считается ошибкой даже без указания STRICT, так как нет способа указать, какая из строк считается «первой».

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

Статус команды

Диагностика

`GET DIAGNOSTICS` *`переменная`* `:= ROW_COUNT;`

число строк, возвращенное последней командой SQL

Признак найденных данных

переменная `FOUND`

после команды SQL: истина, если команда вернула строку

Статус только что выполненной команды SQL (если она не завершилась ошибкой) можно узнать двумя способами.

Во-первых, можно получить число строк, затронутых командой, с помощью конструкции GET DIAGNOSTICS.

Во-вторых, специальная логическая переменная FOUND показывает, были ли обработаны командой хотя бы какие-то данные.

Переменную FOUND можно использовать и как индикатор того, что тело цикла выполнилось минимум один раз.

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

Табличные функции

Строки запроса

```
RETURN QUERY запрос;
```

Одна строка

```
RETURN NEXT выражение;
```
если нет выходных параметров

```
RETURN NEXT;
```
если есть выходные параметры

Особенности

строки добавляются к результату,
но выполнение функции не прекращается

команды можно выполнять несколько раз

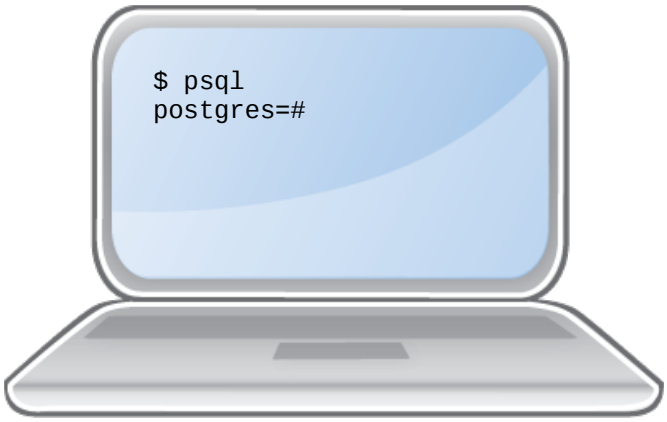
результат не возвращается, пока функция не завершится

Чтобы создать на PL/pgSQL табличную функцию, нужно объявить ее как RETURNS SETOF (точно так же, как и в случае SQL). Но для того, чтобы вернуть из функции множество строк, надо воспользоваться специальной конструкцией RETURNS QUERY *запрос*. Результат будет практически таким же, как в случае SQL-фукции, содержащей этот запрос (за исключением того, что запрос из SQL-функции имеет шансы быть подставленным в объемлющий запрос, а в случае PL/pgSQL-функции это исключено).

Можно возвращать результат и построчно, используя конструкцию RETURN NEXT. Она похожа на обычный RETURN, но не прекращает выполнение функции, а добавляет возвращаемое значение в качестве очередной строки будущего результата. Команду RETURN NEXT (и RETURN QUERY тоже) можно вызывать несколько раз. Окончательный результат будет возвращен только, когда выполнение функции завершится (для этого можно использовать обычную команду RETURN).

https://postgrespro.ru/docs/postgrespro/9.6/plpgsql-control-structures.html#plpgsql-statements-returning

Демонстрация



Итоги

PL/pgSQL тесно интегрирован с SQL

- в процедурном коде можно выполнять запросы (оформленные как выражения или отдельные команды)
- в запросах можно использовать переменные
- можно получать результаты запросов и их статус

Нужно следить за неоднозначностями разрешения имен

Практика

1. Напишите функцию add_author для добавления новых авторов. Функция должна принимать три параметра (фамилия, имя, отчество) и возвращать идентификатор нового автора.
2. Проверьте, что приложение позволяет добавлять авторов.
3. Напишите функцию buy_book для покупки книги. Функция принимает идентификатор книги и уменьшает количество таких книг на складе на единицу. Возвращаемое значение отсутствует.
4. Проверьте, что в «Магазине» появилась возможность покупки книг.

1.

```
FUNCTION add_author(
  last_name text, first_name text, middle_name text
)
RETURNS integer
```

3.

```
FUNCTION buy_book(book_id integer)
RETURNS void
```

Вы можете обратить внимание, что при покупке книг приложение позволяет «уйти в минус». Если бы количество книг хранилось в столбце, простым и хорошим решением было бы сделать ограничение CHECK. Но в нашем случае количество рассчитывается, и мы отложим написание проверки до темы «Триггеры».

Практика

Напишите игру, в которой сервер пытается угадать загаданное пользователем животное, задавая последовательные уточняющие вопросы, на которые можно отвечать «да» или «нет».

Если сервер предложил неправильный вариант, он запрашивает у пользователя имя животного и отличающий вопрос. Эта новая информация запоминается и используется в следующих играх.

1. Создайте таблицу для представления информации.
2. Придумайте интерфейс и реализуйте необходимые функции.
3. Проверьте реализацию.

Пример диалога (между людьми):

— Это млекопитающее?
— Это слон?
— Вы выиграли. Кто это?
— Как отличить кита от слона?

— Да.
— Нет.
— Кит.
— Он живет в воде.

1. Информацию удобно представить в виде двоичного дерева. Внутренние узлы хранят вопросы, листовые узлы — названия животных. Один из дочерних узлов соответствует ответу «да», другой — ответу «нет».

2. Между вызовами функций надо передавать информацию о том, на каком узле дерева мы остановились («контекст» диалога). Функции могут быть, например, такими:

- начать игру (нет входного контекста)

```
FUNCTION start_game(OUT context integer, OUT question text)
```

- продолжение игры (получаем ответ, выдаем следующий вопрос)

```
FUNCTION continue_game(
    INOUT context integer, answer boolean,
    OUT you_win boolean, OUT question text)
```

- завершение игры (внесение информации о новом животном)

```
FUNCTION end_game(context integer, name text, question text)
RETURNS void
```