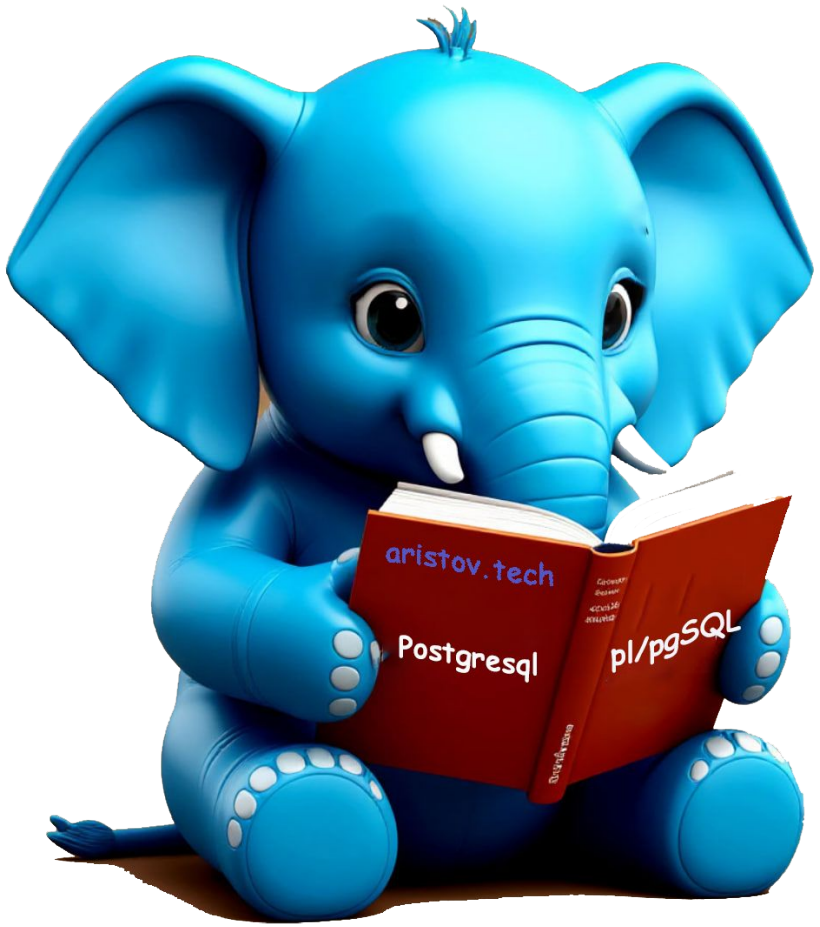
Аристов Евгений

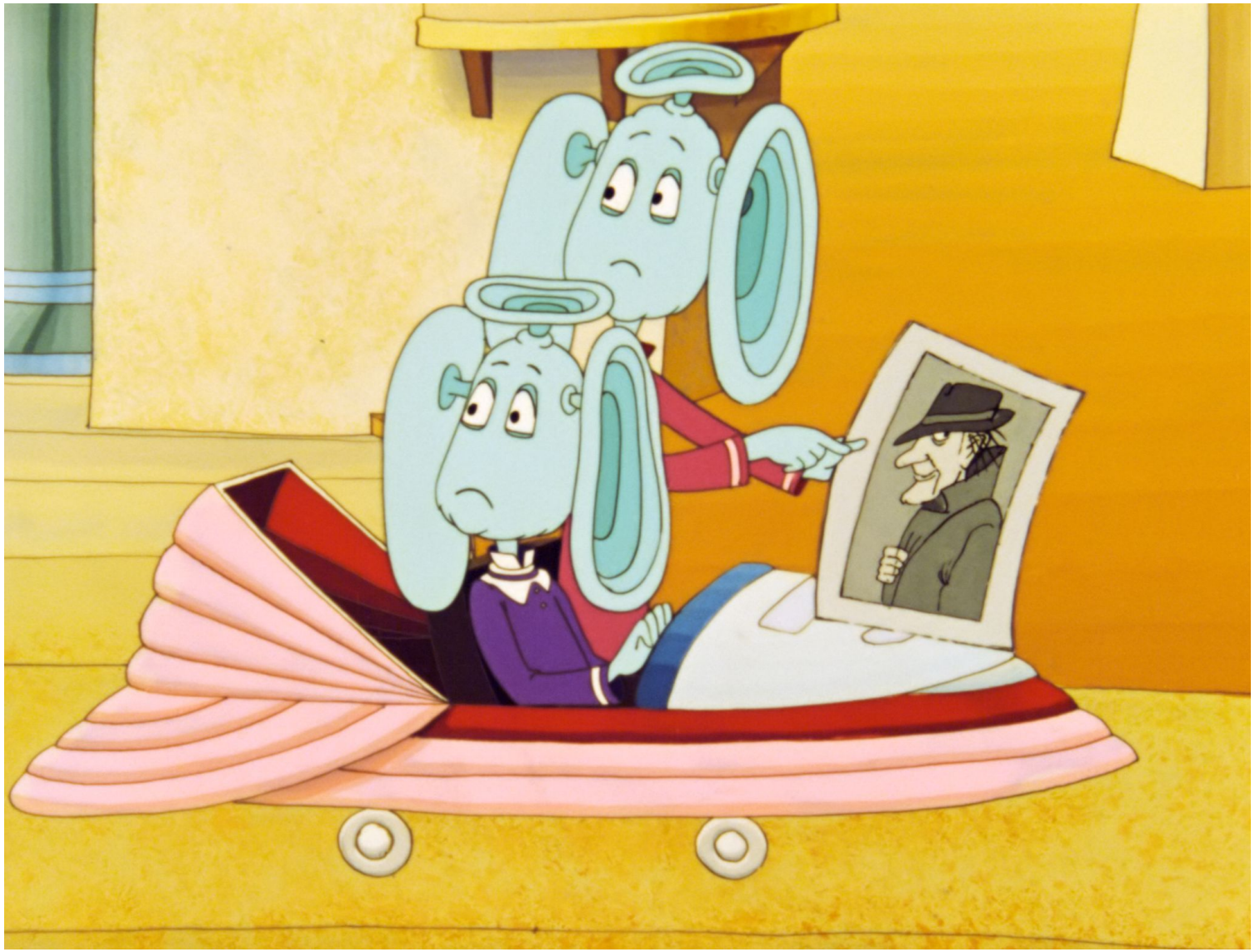
PL/pgSQL в PostgreSQL

за 31 занятие



Циклы

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

Назначение и варианты циклов в PL/pgSQL

Отличия и для каких ситуаций их можно использовать

https://aristov.tech

Циклы

Позволяют повторить серию команд в функции на PL/pgSQL:

- LOOP,
- EXIT,
- CONTINUE,
- WHILE,
- FOR,
- FOREACH

PostgreSQL: Documentation: 16: 43.6. Control Structures

https://aristov.tech

Loop

```
[<<метка>>]
LOOP
   операторы
END LOOP [ метка ];
```

LOOP организует безусловный цикл, который повторяется до бесконечности, пока не будет прекращён операторами EXIT или RETURN. Для вложенных циклов рекомендую использовать ***метку*** (*обсуждали в 6 теме курса*) в операторах EXIT и CONTINUE, чтобы указать, к какому циклу эти операторы относятся.

https://aristov.tech

Выход из цикла

EXIT [*метка*] [WHEN *логическое-выражение*];

Если *метка* не указана, то завершается самый внутренний цикл, далее выполняется оператор, следующий за END LOOP.

Если *метка* указана, то она должна относиться к текущему или внешнему циклу, или это может быть метка блока. При этом в именованном цикле/блоке выполнение прекращается, а управление переходит к следующему оператору после соответствующего END.

При наличии WHEN цикл прекращается, только если *логическое-выражение* истинно. В противном случае управление переходит к оператору, следующему за EXIT.

EXIT можно использовать со всеми типами циклов, не только с безусловным.

Когда EXIT используется для выхода из блока, управление переходит к следующему оператору после окончания блока.

Обратите внимание, что для выхода из блока нужно обязательно указывать метку. EXIT без метки не позволяет прекратить работу блока.

Exit. Пример

```
LOOP
   -- здесь производятся вычисления
   IF count > 0 THEN
      EXIT;  -- выход из цикла
   END IF;
END LOOP;

LOOP
   -- здесь производятся вычисления
   EXIT WHEN count > 0;  -- аналогично предыдущему примеру
END LOOP;

<<new_block>>
BEGIN
   -- здесь производятся вычисления
   IF stocks > 1000 THEN
      EXIT new_block;  -- выход из блока BEGIN
   END IF;
   -- вычисления не будут выполнены, если stocks > 1000
END;
```

Continue

CONTINUE [*метка*] [WHEN *логическое-выражение*];

Если *метка* не указана, то начинается следующая итерация самого внутреннего цикла. То есть все оставшиеся в цикле операторы пропускаются, и управление переходит к управляющему выражению цикла (если есть) для определения, нужна ли ещё одна итерация цикла. Если *метка* присутствует, то она указывает на метку цикла, выполнение которого будет продолжено.

При наличии WHEN следующая итерация цикла начинается только тогда, когда *логическое-выражение* истинно. В противном случае управление переходит к оператору, следующему за CONTINUE.

CONTINUE можно использовать со всеми типами циклов, не только с безусловным.

Continue. Пример

```
LOOP
   -- здесь производятся вычисления
   EXIT WHEN count > 100;
   CONTINUE WHEN count < 50;
   -- вычисления для count в диапазоне 50 .. 100
END LOOP;
```

https://aristov.tech

Цикл WHILE

[<<*метка*>>]
WHILE *логическое-выражение* LOOP
  операторы
END LOOP [*метка*];

WHILE выполняет серию команд до тех пор, пока истинно *логическое-выражение*. Выражение проверяется непосредственно перед каждым входом в тело цикла.

https://aristov.tech

WHILE. Пример

```
WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP
   -- здесь производятся вычисления
END LOOP;
```

Классика:

```
WHILE NOT done LOOP
   -- здесь производятся вычисления
END LOOP;
```

https://aristov.tech

Цикл FOR

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

В цикле FOR итерации выполняются по диапазону целых чисел. Переменная *имя* автоматически определяется с типом integer и существует только внутри цикла (если уже существует переменная с таким именем, то внутри цикла она будет игнорироваться).

Выражения для нижней и верхней границы диапазона чисел вычисляются один раз при входе в цикл. Если не указано BY, то шаг итерации 1, в противном случае используется значение в BY, которое вычисляется, опять же, один раз при входе в цикл. Если указано REVERSE, то после каждой итерации величина шага вычитается, а не добавляется.

https://aristov.tech

FOR. Примеры

```
FOR i IN 1..10 LOOP
   -- внутри цикла переменная i будет иметь значения 1,2,3,4,5,6,7,8,9,10
END LOOP;

FOR i IN REVERSE 10..1 LOOP
   -- внутри цикла переменная i будет иметь значения 10,9,8,7,6,5,4,3,2,1
END LOOP;

FOR i IN REVERSE 10..1 BY 2 LOOP
   -- внутри цикла переменная i будет иметь значения 10,8,6,4,2
END LOOP;
```

Если нижняя граница цикла больше верхней границы (или меньше, в случае REVERSE), то тело цикла не выполняется вообще. При этом ошибка не возникает.

https://aristov.tech

Цикл по результатам запроса FOR

Другой вариант FOR позволяет организовать цикл по результатам запроса.

```
[ <<метка>> ]
FOR цель IN запрос LOOP
   операторы
END LOOP [ метка ];
```

Переменная *цель* может быть строковой переменной, переменной типа record или разделённым запятыми списком скалярных переменных. Переменной *цель* последовательно присваиваются строки результата запроса, и для каждой строки выполняется тело цикла.

Для переменных PL/pgSQL в тексте запроса выполняется подстановка значений, план запроса кешируется для возможного повторного использования, как подробно описано в https://www.postgresql.org/docs/current/plpgsql-implementation.html

https://aristov.tech

Цикл по элементам массива FOR

Цикл FOREACH очень похож на FOR. Отличие в том, что вместо перебора строк SQL-запроса происходит перебор элементов массива. Синтаксис цикла FOREACH:

```
[ <<метка>> ]
FOREACH цель [ SLICE число ] IN ARRAY выражение LOOP
    операторы
END LOOP [ метка ];
```

Без указания SLICE, или если SLICE равен 0, цикл выполняется по всем элементам массива, полученного из *выражения*. Переменной *цель* последовательно присваивается каждый элемент массива и для него выполняется тело цикла.

Цикл по результатам запроса FOR

Ещё одна разновидность этого типа цикла FOR-IN-EXECUTE:

```
[ <<метка>> ]
FOR цель IN EXECUTE выражение_проверки [ USING выражение [, ... ] ] LOOP
    операторы
END LOOP [ метка ];
```

Она похожа на предыдущую форму, за исключением того, что текст запроса указывается в виде строкового выражения. Текст запроса формируется и для него строится план выполнения при каждом входе в цикл. Это даёт программисту выбор между скоростью предварительно разобранного запроса и гибкостью динамического запроса, так же, как и в случае с обычным оператором EXECUTE. Как и в EXECUTE, значения параметров могут быть добавлены в команду с использованием USING.

Итоги

- Выбираем вид цикла в зависимости от бизнес задачи
- Если используются вложенные циклы - желательно использовать метки
- Не допускаем бесконечного цикла - указываем пределы

Практика

Итоги

Итоги

Остались ли вопросы?

Увидимся на следующем занятии

Спасибо за внимание!

Когда дальше и куда?

Аристов Евгений