День 3

О языке программирования PL/pgSQL

- SQL и процедурная логика
- Один язык или несколько?
- PL/pgSQL проектировался на основе языка Oracle PL/SQL, а тот, в свою очередь, был создан на основе подмножества языка Ада
- T-SQL – другой подход у Microsoft SQL Server
- PL/pgSQL — один из первых процедурных языков в PostgreSQL. Он появился в 1998 году в версии 6.4, а с версии 9.0 стал устанавливаться по умолчанию
- Тем не менее это по-прежнему загружаемый модуль и администраторы, особо заботящиеся о безопасности, могут удалить его при необходимости.

PL/pgSQL

- PL/pgSQL это процедурный язык для СУБД PostgreSQL. Целью проектирования PL/pgSQL было создание загружаемого процедурного языка, который:
  - используется для создания функций, процедур и триггеров,
  - добавляет управляющие структуры к языку SQL,
  - может выполнять сложные вычисления,
  - наследует все пользовательские типы, функции, процедуры и операторы,
  - может быть определён как доверенный язык,
  - прост в использовании.

Структура блока PL/pgSQL

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

- Все секции, кроме операторов, являются необязательными.
- Распространённой ошибкой является добавление точки с запятой сразу после BEGIN. Это неправильно и приведёт к синтаксической ошибке.

Операторы в блоке

- В качестве операторов можно использовать команды PL/pgSQL, и большинство команд SQL
- Нельзя использовать служебные команды SQL (например, VACUUM)
- Команды управления транзакциями (например, COMMIT, ROLLBACK) допускаются только в процедурах
- Как оператор может быть использован вложенный блок

DO – анонимный блок

```
DO $$
DECLARE
   -- однострочный комментарий.
   /*— многострочный.
     После каждого объявления переменной и оператора ставится знак ';'.
        */
   foo text;
   bar text := 'World'; -- также допускается = или DEFAULT
BEGIN
   foo := 'Hello'; -- это присваивание
   RAISE NOTICE '%, %!', foo, bar; -- вывод сообщения
END;
$$;
```

Строки и $$

select 'String constant’;
select 'I''m also a string constant’; -- escape символы
select E'I\'m also a string constant’;

tag<string_constant>tag

Tag может содержать от 0 до N символов – отсюда $$

select $$I'm a string constant that contains a backslash \$$;
SELECT $message$I'm a string constant that contains a backslash \$message$;

Выражения PL/pgSQL

Любые выражения, которые встречаются в PL/pgSQL-коде, вычисляются с помощью SQL-запросов к базе данных.

Интерпретатор сам строит нужный запрос, заменяя переменные PL/pgSQL на параметры, подготавливает оператор (при этом разобранный запрос кешируется) и выполняет его.

Это не способствует производительности PL/pgSQL, но обеспечивает интеграцию с SQL.

В выражениях можно использовать любые возможности SQL без ограничений, включая вызов встроенных и пользовательских функций, выполнение подзапросов и т. п.

Переменные

- Все переменные, используемые в блоке, должны быть определены в секции объявления. (За исключением переменной-счётчика цикла FOR, которая объявляется автоматически.)

имя [CONSTANT] тип [COLLATE имя_правила_сортировки] [NOT NULL] [{ DEFAULT | := | = } выражение];

:= или = --операторы присваивания равноправны, можно использовать любой

quantity integer DEFAULT 32;

url varchar := 'http://mysite.com';

transaction_time CONSTANT timestamp with time zone := now();

Инициализация происходит в момент входа в блок исполнения

ALIAS

Задание псевдонима для переменной или параметра функции (процедуры)

```
DECLARE
  prior ALIAS FOR old;
  updated ALIAS FOR new;

CREATE FUNCTION sales_tax(real) RETURNS real AS $$
DECLARE
    subtotal ALIAS FOR $1;
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

Атрибут %TYPE

Конструкция %TYPE предоставляет тип данных переменной или столбца таблицы. Её можно использовать для объявления переменных, содержащих значения из базы данных. Например, для объявления переменной с таким же типом, как и столбец user_id в таблице users нужно написать:

```
user_id users.user_id%TYPE;
```

Используя %TYPE, не нужно знать тип данных структуры, на которую вы ссылаетесь. И самое главное, если в будущем тип данных изменится (например: тип данных для user_id поменяется с integer на real), то вам может не понадобиться изменять определение функции.

Атрибут %ROWTYPE

Переменная составного типа называется переменной-кортежем (или переменной типа кортежа). Значением такой переменной может быть целый кортеж, полученный в результате выполнения запроса SELECT или FOR, при условии, что набор столбцов запроса соответствует заявленному типу переменной. Доступ к отдельным полям значения кортежа осуществляется как обычно, через точку, например rowvar.field.

Переменная-кортеж может быть объявлена с таким же типом, как и строка в существующей таблице или представлении, используя нотацию имя_таблицы%ROWTYPE; или с именем составного типа.

```
DECLARE
    t2_row table2%ROWTYPE;
```

Тип переменной RECORD (запись)

Переменные типа record похожи на переменные-кортежи, но они не имеют предопределённой структуры. Они приобретают фактическую структуру от строки, которая им присваивается командами SELECT или FOR. Структура переменной типа record может меняться каждый раз при присваивании значения. Следствием этого является то, что пока значение не присвоено первый раз, переменная типа record не имеет структуры и любая попытка получить доступ к отдельному полю приведёт к ошибке во время выполнения.

имя RECORD;

Выполнение команд SQL

В функции на PL/pgSQL можно выполнить любую команду SQL, не возвращающую строк, просто написав эту команду. Например, можно создать таблицу и заполнить её данными, написав

```
CREATE TABLE mytable (id int primary key, data text);
INSERT INTO mytable VALUES (1,'one'), (2,'two');
```

Если команда всё же возвращает строки (например, SELECT или INSERT/UPDATE/DELETE с предложением RETURNING), есть два варианта действий.

Если команда возвращает максимум одну строку или вам интересна только первая строка результата, напишите команду как обычно, но добавьте предложение INTO для захвата вывода

Чтобы обрабатывать все результирующие строки, запишите команду в качестве источника данных для цикла FOR

PERFORM - выполнение запросов, не возвращающих результат

Иногда бывает полезно вычислить значение выражения или запроса SELECT, но отказаться от результата, например, при вызове функции, у которой есть побочные эффекты, но нет полезного результата. Для этого в PL/pgSQL, используется оператор PERFORM:

```
PERFORM create_mv('cs_session_page_requests_mv', my_query);
```

Эта команда выполняет запрос и отбрасывает результат. Запросы пишутся таким же образом, как и в команде SQL SELECT, но ключевое слово SELECT заменяется на PERFORM.

Переменные PL/pgSQL будут подставлены в запрос, план запроса также кешируется.

Специальная переменная FOUND принимает значение true, если запрос возвращает, по крайней мере, одну строку, или false, если не возвращает ни одной строки

Выполнение запросов, возвращающих одну строку

SELECT … INTO

- получение первой строки выборки
- одна переменная record или составного типа или подходящее количество скалярных переменных
- ***Обратите внимание, что данная интерпретация SELECT с INTO полностью отличается от PostgreSQL команды SELECT INTO, где в INTO указывается вновь создаваемая таблица. Если вы хотите в функции на PL/pgSQL создать таблицу, основанную на результате команды SELECT, используйте синтаксис CREATE TABLE … AS SELECT.***

INSERT, UPDATE, DELETE RETURNING … INTO

- получение вставленной (измененной, удаленной) строки
- одна переменная составного типа или подходящее количество скалярных переменных

Особенности применения

Если результат команды присваивается переменной-кортежу или списку переменных, то они должны в точности соответствовать по количеству и типам данных столбцам результата, иначе произойдёт ошибка во время выполнения.

Если используется переменная типа record, то она автоматически приводится к типу строки результата команды.

Предложение INTO может появиться практически в любом месте SQL-команды.

Обычно его записывают непосредственно перед или сразу после списка выражения_select в SELECT или в конце команды для команд других типов.

Рекомендуется следовать этому соглашению на случай, если правила разбора PL/pgSQL ужесточатся в будущих версиях.

Динамически формируемые команды в PL/pgSQL

SQL-команды формируется строкой в момент выполнения

EXECUTE строка-команды [INTO [STRICT] цель] [USING выражение [, ...]];

+

гибкость в приложении

формирование нескольких конкретных запросов вместо одного универсального – может быть удобно для оптимизации

-

операторы не подготавливаются

возрастает риск SQL-инъекции

Сложнее сопровождать

Защита от SQL-инъекций в динамических командах

Подстановка значений параметров

- предложение USING
- гарантируется невозможность внедрения SQL-кода

Экранирование значений

- Имена: format('%I'), quote_ident
- литералы: format('%L'), quote_literal, quote_nullable
- внедрение SQL-кода невозможно при правильном использовании

- https://postgrespro.ru/docs/postgresql/15/functions-string

Обычные строковые функции конкатенация и др.

- возможно внедрение SQL-кода!

Защита от SQL-инъекции

В тексте команды можно использовать значения параметров, ссылки на параметры обозначаются как $1, $2 и т. д. Эти символы указывают на значения, находящиеся в предложении USING.

Такой метод зачастую предпочтительнее, чем вставка значений в команду в виде текста: он позволяет исключить во время выполнения дополнительные расходы на преобразования значений в текст и обратно, и не открывает возможности для SQL-инъекций, не требуя применять экранирование или кавычки для спецсимволов. Пример:

```
EXECUTE 'SELECT count(*) FROM mytable WHERE inserted_by = $1 AND
inserted <= $2'
   INTO c
   USING checked_user, checked_date;
```

Защита от SQL-инъекции

Символы параметров можно использовать только вместо значений данных. Если же требуется динамически формировать имена таблиц или столбцов, их необходимо вставлять в виде текста. Например, если в предыдущем запросе необходимо динамически задавать имя таблицы, можно сделать следующее:

```
EXECUTE 'SELECT count(*) FROM '
    || quote_ident(tabname)
    || ' WHERE inserted_by = $1 AND inserted <= $2'
  INTO c
  USING checked_user, checked_date;
```

Более аккуратным решением будет использование указания %I с функцией format(), позволяющее вставить имя таблицы или столбца, которое будет автоматически заключено в кавычки:

```
EXECUTE format('SELECT count(*) FROM %I '
  'WHERE inserted_by = $1 AND inserted <= $2', tabname)
  INTO c
  USING checked_user, checked_date;
```

Получение статуса выполнения команды

STRICT

- Если указание STRICT отсутствует в предложении INTO, то цели присваивается первая строка, возвращённая командой;
- или NULL, если команда не вернула строки. (Заметим, что понятие «первая строка» определяется неоднозначно без ORDER BY.)
- Все остальные строки результата после первой отбрасываются.
- Если добавлено указание STRICT, то команда должна вернуть ровно одну строку или произойдёт ошибка во время выполнения: либо NO_DATA_FOUND (нет строк), либо TOO_MANY_ROWS (более одной строки).
- Для INSERT/UPDATE/DELETE с RETURNING, PL/pgSQL возвращает ошибку, если выбрано более одной строки, даже в том случае, когда указание STRICT отсутствует. Так происходит потому, что у этих команд нет возможности, типа ORDER BY, указать какая из задействованных строк должна быть возвращена.

STRICT

```
BEGIN
    SELECT * INTO STRICT myrec FROM emp WHERE empname =
myname;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE EXCEPTION 'Сотрудник % не найден', myname;
        WHEN TOO_MANY_ROWS THEN
            RAISE EXCEPTION 'Сотрудник % уже существует', myname;
END;
```

Переменная FOUND

после команды SQL: истина, если команда вернула (обработала) строку

После успешного выполнения команды с указанием STRICT, значение переменной FOUND всегда принимает значение true.

после цикла: выполнилась хотя бы одна итерация - true

GET DIAGNOSTICS

Команда GET DIAGNOSTICS позволяет узнать количество строк, затронутых исполненной командой

row_count

Условный оператор IF

- if then

```
if condition then   statements;
end if;
```

- if then else

```
if condition then  statements;
else  alternative-statements;
END if;
```

- if then elsif

```
if condition_1 then  statement_1;
elsif condition_2 then  statement_2...
elsif condition_n then  statement_n;
else  else-statement;
end if;
```

Условный оператор CASE

```
if found then
        case
            when total_payment > 200 then
          service_level = 'Platinum' ;
        when total_payment > 100 then
            service_level = 'Gold' ;
        else
          service_level = 'Silver' ;
      end case;
          raise notice 'Service Level: %', service_level;
   else
        raise notice 'Customer not found';
      end if;
```

Simple case

```
if found then
            case rate
              when 0.99 then
          price_segment =  'Mass';
              when 2.99 then
          price_segment = 'Mainstream';
              when 4.99 then
          price_segment = 'High End';
              else
             price_segment = 'Unspecified';
              end case;
            raise notice '%', price_segment;
  end if;
```

Условный оператор CASE

- Simple case -- оценивает значение
- Searched case – проверяет логическое условие

- Это оператор, а не выражение!
- Case выражение вычисляет значение, а оператор выполняет действие

EXIT и CONTINUE в циклах

Выход из цикла

```
exit when counter > 10;
```

То же самое через if

```
if counter > 10 then   exit;
end if;
```

CONTINUE – прекращиение текущей итерации цикла и переход к следующей

Цикл LOOP

- Безусловный цикл
- Выход по exit или return

```
<<label>>loop   statements;
 if condition then      exit;
 end if;
end loop;
```

Цикл WHILE

[<<label>>]while condition

loop statements;

end loop;

Проверка условия на входе в цикл. Цикл прекращается при результате false

Цикл FOR (целочисленный)

[<<label>>]for loop_counter in [reverse] from.. to [by step]
loop statements
end loop [label];

Цикл по счетчику
do
$$begin for counter in 1..5
loop raise notice 'counter: %', counter;
end loop;
end; $$;

Цикл FOR (по строкам запроса)

```
for f in select title, length
        from film
        order by length desc, title
        limit 10
    loop
     raise notice '%(% mins)', f.title, f.length;
    end loop;
```

Цикл FOREACH (по элементам массива)

[<<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.

```
FOREACH x IN ARRAY $1
  LOOP
    s := s + x;
  END LOOP;
  RETURN s;
```

Курсоры в PL/pgSQL

- declare
- open
- fetch
- проверить есть ли еще записи
- close

Курсорные переменные

declare my_cursor refcursor;

Или

cursor_name [[no] scroll] cursor [(name datatype, name data type, ...)] for query;

Открытие курсоров - bound

open cursor_variable[(name:=value,name:=value,...)];

Задается ранее определенная переменная курсора

Это открытие привязанного курсора

В этот момент курсор наполняется данными запроса, определенного для него

Открытие курсоров - unbound

OPEN unbound_cursor_variable [[NO] SCROLL] FOR query;

open my_cursor for select * from city where country = p_country;

Использование курсоров - fetch

fetch [direction { from | in }] cursor_variable into target_variable; --получение текущей записи курсора

Direction

- NEXT –по умолчанию
- LAST
- PRIOR
- FIRST
- ABSOLUTE count
- RELATIVE count
- FORWARD --только для SCROLL
- BACKWARD --только для SCROLL

- fetch cur_films into row_film;
- fetch last from row_film into title, release_year;

Использование курсоров - move

Move – перемещение по курсору, не получая текущую запись

move [direction { from | in }] cursor_variable;

move cur_films2;
move last from cur_films;
move relative -1 from cur_films;
move forward 3 from cur_films;

Обновление и удаление через курсор

update table_name set column = value, ... where current of cursor_variable;

delete from table_name where current of cursor_variable;

update film set release_year = p_year where current of cur_films;

Закрытие курсоров

close cursor_variable;

Освобождает ресуры

Позволяет заново использовать переменную курсора через OPEN

Функции, возвращающие таблицу

```
create or replace function function_name (   parameter_list)
returns table ( column_list )
language plpgsql
As
 $$
declare -- variable declaration
Begin
-- body
end; $$;
```