Процедурный язык PL/pgSQL

Обзор PostgreSQL PL/pgSQL

- PL/pgSQL — процедурный язык программирования для PostgreSQL
  - PL/pgSQL по умолчанию поставляется с PostgreSQL
  - PL/pgSQL наследует все пользовательские типы, функции и операторы
  - PL/pgSQL позволяет расширить функциональность PostgreSQL
- PL/pgSQL был разработан для:
  - Создания серверных объектов со сложной логикой: пользовательских функций, хранимых процедур и триггеров
  - Расширения стандартного SQL, путем добавления в него управляющих структур, таких как операторы **if**, **case** и **loop**
- Начиная с PostgreSQL 9.0, PL/pgSQL устанавливается по умолчанию
- Не может быть переносимым на другие СУБД

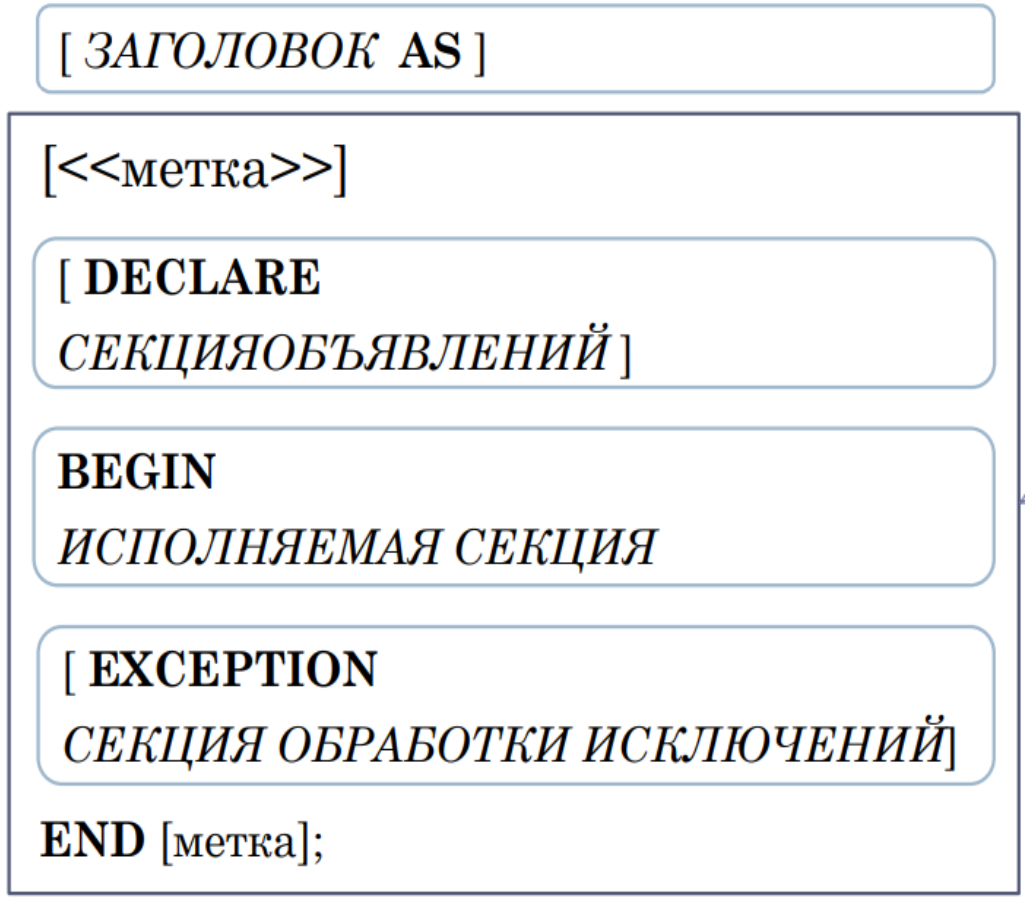
Блок

- Блок является минимальной единицей группировки кода
- Различают:
  - **Анонимный блок** - не хранится в БД
  - **Именованный блок** - хранимая процедура, функция или триггер
- Основные лексические единицы:
  - ***Идентификаторы*** - названия объектов языка – переменных, констант, исключений и т.п.
  - ***Разделители*** - символ завершения команды, простые и сложные операторы
  - ***Литералы*** - значения, которые принимают переменные и константы
  - ***Комментарии*** - однострочные (-- комментарий) и многострочные (/* комментарий */)

Обобщённая структура блока

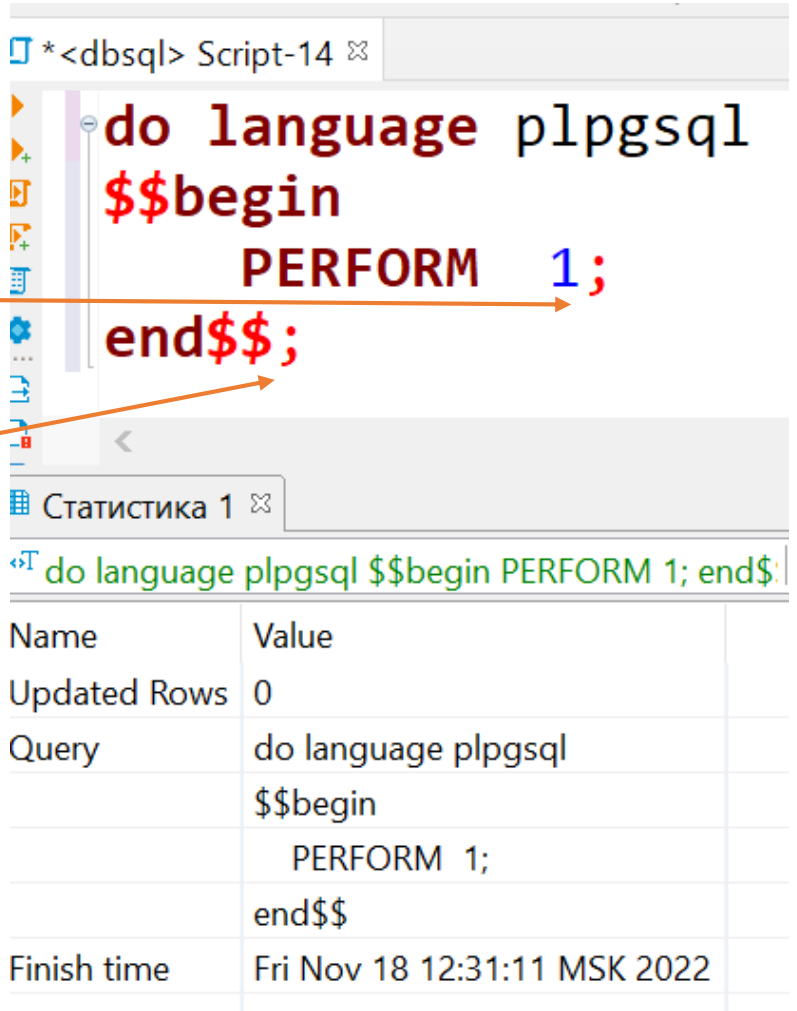
- Код блока оформляется в виде строки:
  - в апострофах
  - в долларах $[метка]$
- **BEGIN** и **END** служат для обозначения начала и конца блока
- Внимание: инструкция **DO** не принадлежит блоку - используется для выполнения анонимного блока

```
DO
LANGUAGE plpgsql
$block_name$ -- метка блока
DECLARE
    Секция_объявления_переменных_констант;
BEGIN
    Секция_исполнения кода;
END
$block_name$  ;
```



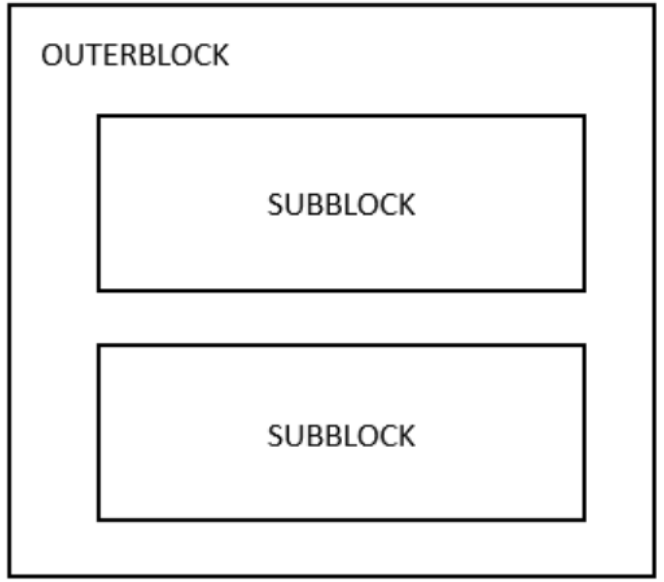
Блок. Замечания

- Каждая инструкция / блок внутри основного блока должна заканчиваться точкой с запятой (;)
- Основной блок также рекомендуется заканчивать точкой с запятой (;)
- Минимально-возможный блок должен содержать ключевые слова **BEGIN** и **END**



Подблоки PL/pgSQL

- PL/pgSQL позволяет размещать блок внутри тела другого блока
  - Блок, вложенный в другой блок, называется **подблоком**
  - Блок, содержащий подблок, называется **внешним блоком**
- Переменные в подблоке могут иметь те же имена, что и во внешнем блоке (но это не рекомендуется):
  - Если имя переменной в подблоке совпадает с именем внешней переменной, внешняя переменная становится недоступна в подблоке
  - Для получения доступа к внешней переменной необходимо использовать метку блока: **имя_блока.имя_переменной**



Функции plpgsql

```
CREATE [OR REPLACE] FUNCTION имя
([[режим_аргумента] [имя_аргумента] тип_аргумента [{DEFAULT|=} выражение_по_умолчанию ]
 [, ...] ])
[ RETURNS {тип_результата|void} |
  RETURNS TABLE ( имя_столбца тип_столбца [, ...] ) ]
$$
BEGIN
--тело функции
END
$$
LANGUAGE plpgsql;
```

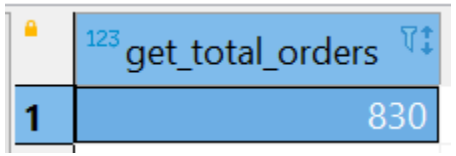
- **Тело функции на языке plpgsql – это блок!**
- Для возврата значения:
  - **RETURN** – возврат скалярной величины
  - **RETURN QUERY –** возврат множества записей

Возврат скалярной величины

```
CREATE OR REPLACE FUNCTION get_total_orders ()
RETURNS bigint
AS
$$
BEGIN
--тело функции
RETURN count(*) FROM "Sales"."Orders";
END
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION get_total_orders ()
RETURNS bigint
AS
$$
--тело функции
   SELECT count(*) FROM "Sales"."Orders";
$$ LANGUAGE SQL;
```

```
select get_total_orders () ;
```



	123 get_total_orders
1	830

Возврат результирующей выборки

```
CREATE OR REPLACE FUNCTION functions.calc_middle_price()
RETURNS SETOF "Production"."Products"
AS
$$
DECLARE
    avg_price numeric;
    low_price numeric;
    high_price numeric;
    k1 constant numeric := 0.75;
BEGIN
    SELECT avg(unitprice::numeric) into avg_price
    FROM "Production"."Products";
    low_price = avg_price * k1;
    high_price = avg_price * 1.25;
    RETURN QUERY
    SELECT *
    FROM "Production"."Products"
    WHERE unitprice BETWEEN low_price::money AND high_price::money;
END
$$ LANGUAGE plpgsql;
```

```
select *
from functions.calc_middle_price();
```

Возврат через OUT-аргументы (2 варианта)

```
CREATE OR replace FUNCTION get_price_boundaries
(OUT max_price money, OUT min_price money)
AS $$
begin
    max_price := MAX(unitprice) FROM "Production"."Products";
    min_price := MIN(unitprice) FROM "Production"."Products";
END
$$ LANGUAGE plpgsql;
```

1

```
CREATE OR replace FUNCTION get_price_boundaries
(OUT max_price money, OUT min_price money)
AS $$
begin
    SELECT MAX(unitprice), MIN(unitprice)
    INTO max_price, min_price
    FROM "Production"."Products";
END
$$ LANGUAGE plpgsql;
```

2

```
select get_price_boundaries();
```

	get_price_boundaries
1	($263.50,$2.50)

```
select *
from  get_price_boundaries();
```

	max_price	min_price
1	$263.50	$2.50

Оператор накопления строк

- Иногда бывает необходимо накапливать записи в результирующем наборе (построчный процессинг)

```
RETURN NEXT выражение;
```

  - Если выражение не указано, подразумевается заполнение ранее объявленных выходных параметров
- Для выхода из функции и возврата накопленного результата (до окончания блока функции) используется оператор **RETURN**
- Обработка большого массива данных в цикле – потенциально плохая производительность
  - Если нет необходимости – рекомендуется использовать обычный SQL
  - Декларативный стиль чистого SQL также более читабелен, чем процедурный стиль

Оператор накопления строк

```
create or replace function get_film (
p_pattern varchar,
p_year int
)
returns table (
              film_title varchar,
              film_release_year int
)
language plpgsql
as $$
declare
          var_r record;
begin
     for var_r in(
          select title, release_year
          from film
          where title ilike p_pattern and release_year = p_year
          ) loop film_title := upper(var_r.title) ;
          film_release_year := var_r.release_year;
          return next;
     end loop;
end; $$
```

Использование условной логики

```
IF выражение THEN
        логика
ELSIF выражение THEN
        логика
ELSEIF выражение THEN
        логика
ELSE
        логика
END IF;
```

```
CREATE OR REPLACE FUNCTION functions.convert_temp
  (temperature real, to_celsius bool default true)
RETURNS real
AS
$$
DECLARE
    result_temp real;
BEGIN
    if to_celsius then --to_celsius = true
        result_temp = (5.0/9.0) * (temperature-32);
    else
        result_temp = (9.0 * temperature + (32 * 5))/5.0;
    end if;

    RETURN result_temp;
END
$$ LANGUAGE plpgsql;
```

Циклы

```
CONTINUE WHEN выражение
```

```
[метка]
WHILE выражение
LOOP
    логика
END LOOP [метка];
```

```
[метка]
LOOP
    EXIT WHEN выражение
    логика
END LOOP [метка];
```

- Цикл **WHILE** выполняет блок кода до тех пор, пока условие не будет оценено как **FALSE**
- Цикл **LOOP** многократно выполняет блок кода до тех пор, пока не будет завершен оператором **EXIT** или **RETURN**

```
[метка]
FOR счетчик IN [REVERSE b..a] a..b [BY шаг]
LOOP
   логика
END LOOP [метка];
```

В случае использования опции **REVERSE** от значения **счетчика** на каждой итерации цикла **FOR LOOP вычитается шаг**


```
CREATE OR REPLACE FUNCTION functions.fib(n int)
  RETURNS int
AS
$$
DECLARE
     counter int = 0;
     i int = 0;
     j int =1;
     f_sum int;
BEGIN
     IF n= 0  THEN
          RETURN 0
     END IF;

     WHILE counter < n
          LOOP
              f_sum = i + j;
              i = j;
              j = f_sum;
              counter = counter + 1;
          END loop;
     RETURN i;
end loop;
RETURN i;
end;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION functions.fib (n int)
  RETURNS int
AS
$$
DECLARE
     counter int = 0;
     i int = 0;
     j int = 1;
BEGIN
     IF n= 0  THEN
          RETURN 0
     END IF;

     WHILE counter < n
          LOOP
              SELECT j, i+j INTO i, j;
              counter = counter + 1;
          END loop;
     RETURN i;
END;
$$ LANGUAGE plpgsql;
```

Цикл по записям динамического запроса

```
[метка]
FOR переменная_record IN запрос
LOOP
   логика
END LOOP [метка];
```

- Запрос любая команда SQL, возвращающая строки: SELECT или INSERT, UPDATE, DELETE с предложением RETURNING
- На каждой итерации в **переменную_record** передается запись
- Для получения данных из конкретного поля используется ссылка вида: **переменная_record.имя_поля**

```
[метка]
FOR ROW IN EXECUTE запрос_выражение
[ USING параметры_запроса [, ... ] ]
LOOP
    логика
END LOOP [метка];
```

- Текст запроса указывается в виде строкового выражения
- Для запроса строится план выполнения **при каждом входе в цикл**
- Предложение **USING** используется для передачи параметров в запрос