



PostgreSQL

Создание программных модулей в БД

Функции и процедуры



Подпрограммы

- **Функции и процедуры** называются **подпрограммами**
 - Подпрограммы, позволяют разбить сложную логику на более простые модули, решающие локальные, повторяющиеся задачи
- **Подпрограмма** – это именованный блок PL/pgSQL, хранящийся в БД и выполняющий одно или несколько действий
 - Код подпрограммы (тело) - строковая константа, содержащая код на выбранном языке программирования
 - Тело подпрограммы сохраняется в системном каталоге БД (в виде строковой константы) - **хранимые**
 - Каждый раз при вызове подпрограммы её код интерпретируется
 - **Единственное исключение - функции на языке Си**
- При создании проводится поверхностная проверка синтаксиса и семантики, но не более
 - Обращение к несуществующей таблице будет распознано
 - Присвоение одной переменной значения другой, несуществующей, до момента выполнения не вызовет ошибку



PostgreSQL

Подпрограммы

- Подпрограммы позволяют:
 - разграничить работу *Frontend developers* и *Server-side developers*
 - обрабатывать кортежи там где они хранятся
 - переиспользовать код разными клиентскими приложениями
 - обеспечить управление безопасностью через регулирование доступа к объектам БД
 - уменьшить сетевой трафик
- Делятся на:
 - SQL-подпрограммы
 - Процедурные (PL/pgSQL – основной диалект)
 - Серверные (написаны на языке C)
 - Собственные C-функции

Функции

- **Функция** – объект БД, принимающий параметры и возвращающий результат (но может и не возвращать)
- Функции вызываются в контексте выражения - как часть запроса или команды DML
- Функции выполняют произвольный список операторов и возвращают результат последнего оператора в списке
 - Могут содержать операторы CRUD – SELECT, INSERT, UPDATE, DELETE
 - **Не могут содержать операторы управления транзакциями (BEGIN, COMMIT, ROLLBACK и т. п.)!**
 - **Не могут содержать служебные команды (VACUUM, CREATE INDEX и т.п.)**

<https://postgrespro.ru/docs/postgresql/13/sql-createfunction>

Процедуры

- **Процедура** — объект БД, который может принимать параметры и не возвращает результат (отсутствует предложение **RETURNS**)
- В отличие от функций, процедуры:
 - Были введены в PostgreSQL 11
 - Вызываются командой **CALL**
 - Могут возвращать данные **только** через параметры (**INOUT** параметр)
 - Могут содержать операторы управления транзакциями (BEGIN, COMMIT, ROLLBACK и т. п.) если:
 - процедура вызывается не из явной транзакции
 - **тело процедуры, написано на языке PL/pgSQL**

<https://postgrespro.ru/docs/postgresql/13/sql-createprocedure>

<https://postgrespro.ru/docs/postgresql/13/sql-call>



Создание подпрограммы

Функция

```
CREATE [OR REPLACE] FUNCTION
имя_функции (список_параметров)
[RETURNS
тип_возвращаемого_значения]
LANGUAGE имя_языка AS $$
[DECLARE ...]
BEGIN
...
[RETURN ...]
[EXCEPTION
...
[RETURN...]]
END
$$
```

Тело
подпрограммы
Блок plpgsql |
Код sql

Процедура

```
CREATE [OR REPLACE] PROCEDURE
имя_процедуры (список_параметров)
LANGUAGE имя_языка AS $$
[DECLARE ...]
BEGIN
...
[RETURN] -- используется для выхода из
-- процедуры до конца блока BEGIN
[EXCEPTION
...]
END
$$
```

Создание подпрограммы, замечания

- Указание ключевых слов **OR REPLACE** позволяет сохранить:
 - владельца подпрограммы
 - права доступа к ней
 - ссылающиеся на неё объекты (например, представления и триггеры)
- Заголовок подпрограммы без учёта описания исходящих параметров задаёт **сигнатуру** подпрограммы
- **Функция может возвращать тип VOID - указание, что функция не возвращает никаких данных, а используется в режиме процедуры**
 - фактически возвращается NULL
 - для совместимости с версиями PL/pgSQL ниже 11
- Использование оператора RETURN в функции обязательно, если в ней нет исходящих параметров, и она не возвращает тип данных VOID
 - В ином случае **RETURN** может быть использован только для выхода из функции



Пример 1

Создание функции

```
CREATE OR REPLACE FUNCTION upd_customer_region ()  
RETURNS void AS  
$$  
--тело функции  
    UPDATE tmp_customers  
    SET region = 'NA'  
    WHERE region IS NULL;  
$$ LANGUAGE SQL;
```

Вызов функции

```
select upd_customer_region () ;
```

Пример 2

Создание процедуры

```
CREATE PROCEDURE "functions".new_category(cat_name varchar(15),  
                                           cat_desc varchar(200))  
  
LANGUAGE SQL  
AS $$  
INSERT INTO "Production"."Categories" (categoryname, description)  
VALUES(cat_name, cat_desc);  
$$;
```

Вызов процедуры

```
CALL "functions".new_category('Обувь', 'Обувь для всех возрастов');
```

Изменение и удаление подпрограммы

- Редактирование тела подпрограммы

CREATE OR REPLACE

- Изменение некоторых атрибутов, например:

```
ALTER { FUNCTION | PROCEDURE | ROUTINE } имя ([тип_входящего_параметра[,...]])  
RENAME TO новое_имя
```

- Описание параметров или тип возвращаемого функцией значения изменить **нельзя** - подпрограмму необходимо пересоздать

- Удаление

```
DROP { FUNCTION | PROCEDURE | ROUTINE } [IF EXISTS] имя  
([тип_входящего_параметра[,...]]), ... [CASCADE]
```

- IF EXISTS - предотвращает получение ошибки при попытке удалить несуществующую подпрограмму
 - CASCADE - удаление вместе с зависимыми объектами
 - по умолчанию выполняется в режиме **RESTRICT**, т.е. при наличии зависимых объектов выдаст ошибку

<https://postgrespro.ru/docs/postgresql/13/sql-alterfunction>

Просмотр кода подпрограммы

- Через графический интерфейс клиента для работы с БД
- Через запрос к представлению системного каталога **pg_proc**

```
SELECT proname, prosrc FROM pg_catalog.pg_proc  
WHERE proname = 'имя_подпрограммы';
```

Параметры подпрограммы

- **Формальные параметры** – имена, объявленные в заголовке блока
- **Фактические параметры** – непосредственно значения и выражения, передаваемые в блок
- Общий вид описания формальных параметров

[режим] [имя] тип_данных [{**DEFAULT** | =} значение_по_умолчанию]

- Если имя параметра не указано:
 - к нему можно обратиться по порядковому номеру (**\$N**)
 - можно назначить псевдоним в секции объявлений: **имя ALIAS FOR \$N**

```
CREATE FUNCTION calc
    (IN a int, IN b int DEFAULT 2, IN c int DEFAULT 3)
RETURNS int
AS $$
    SELECT $1 + $2 + $3;
$$ LANGUAGE SQL;
```

Режимы параметров

- Режимы параметров определяют их поведение:
 - **IN** – входной параметр (по умолчанию)
 - Используется для передачи значения в подпрограмму
 - Действует как константа - не может быть присвоено значение в теле подпрограммы
 - **OUT** – выходной параметр (**не применим для процедур!**)
 - Используется для возврата значения из функции
 - Действует как неинициализированная переменная
 - **Необходимо присвоить значение в теле функции**
 - **INOUT** – входной/выходной параметр
 - Используется как для передачи значения подпрограмме, так и возврата обновленного значения
 - Действует как инициализированная переменная
 - **Необходимо присвоить значение в теле подпрограммы**
 - **VARIADIC** – массив входных параметров
- Параметры OUT и INOUT **нельзя** использовать с предложением **RETURNS TABLE** функции



PostgreSQL

Описание параметров: тип данных и значение по умолчанию

- **Тип данных**
 - длина значения игнорируется
 - можно ссылаться на типы данных столбцов таблиц с помощью **%TYPE**
- **Значение по умолчанию (DEFAULT значение)**
 - может быть задано только для параметров **IN** и **INOUT**
 - используется если при вызове не был указан фактический параметр
 - **NULL** как значение по умолчанию необходимо прописать явно
- **Последовательность аргументов:**
 - Если для входного аргумента определено **DEFAULT-значение**, все последующие входные аргументы также должны иметь значения по умолчанию
 - За единственным параметром **VARIADIC** могут следовать только аргументы **OUT**

Режимы передачи параметров

IN	OUT	INOUT
Используется для передачи значения в подпрограмму	Используется для возврата значения из подпрограммы	Может использоваться как для передачи, так и для возврата значения
Значение нельзя изменить (= константа)	На входе – NULL (= неинициализированная переменная)	На входе: <ul style="list-style-type: none"> переданное значение (может быть константа, так как <u>фактическое</u> значение не изменяется) возможно NULL (обычно = инициализированная переменная)
Может иметь значение по умолчанию	Не может иметь значение по умолчанию	Может иметь значение по Умолчанию
Режим по умолчанию	<ul style="list-style-type: none"> Не требуют наличия раздела RETURNS у функции Могут использоваться для возврата нескольких значений функцией (соответствует RETURNS RECORD) или возврата значения процедурой (INOUT) 	

Нотации передачи параметров

- **Позиционная** – связывание в порядке передачи
 - Фактические параметры необходимо передавать в том же порядке, в каком указаны формальные в заголовке подпрограммы
 - Единственный возможный вариант, если в заголовке подпрограммы не были указаны имена формальных параметров
- **Именованная** – явное связывание формального параметра с фактическим по имени:
 - имя_формального_параметра=>фактический_параметр
 - используется, когда значения первых параметров нужно пропустить
- **Смешанная** – комбинация методов
 - используется, когда нужно пропустить значения параметров в середине



Передача входных параметров

```
CREATE FUNCTION get_product_count
( min_price money DEFAULT 20.00,
  max_price money DEFAULT 80.00)
RETURNS bigint
AS
$$
    SELECT count(*)
    FROM "Production"."Products"
    WHERE unitprice BETWEEN min_price
                        AND max_price;

$$ LANGUAGE sql;
```

--Использование значений по умолчанию

```
select get_product_count();
```

--Позиционная передача параметров

```
select get_product_count(40::money,90::money);
```

--Поименованная передача параметров

```
select get_product_count(min_price => 40::money,
                        max_price => 90::money);
```

--Использование смешанной записи:

--комбинация позиционной и именованной нотации

```
select get_product_count(40::money,
                        max_price => 90::money);
```

--Использовать именованные параметры перед

--позиционными параметрами нельзя!

```
select get_product_count(max_price => 90::money,
                        40::money);
```



SQL Error [42601]: ERROR: positional argument cannot follow named argument

Позиция: 61

Подпрограммы с переменным числом входных параметров

- **VARIADIC** параметр определяется последним в списке входящих параметров и определяет одномерный массив значений указанного типа

VARIADIC имя_параметра тип_данных_массива

```
some_func (p1 int , VARIADIC p2 varchar[])
```

- При вызове в качестве фактического параметра **VARIADIC** передаётся
 - либо список значений с типом данных элемента массива
 - либо массив с указанием VARIADIC

```
some_func (0, 'a', 'b', 'c')           --передача списка значений  
some_func (0, VARIADIC ARRAY['a', 'b', 'c']) --передача массива
```

- Формальные параметры, являющиеся элементами такого массива, не имеют имён – номер элемента в массиве



Пример

```
CREATE OR REPLACE FUNCTION "functions".get_total
    (num1 int, VARIADIC p_arr int[])
RETURNS bigint
AS
$$
    --тело функции
    select sum(t) + num1 from unnest(p_arr) as t;
$$ LANGUAGE SQL;
```

```
select "functions".get_total(5, VARIADIC ARRAY[1, 10, 15]);
```

```
select "functions".get_total(5, 1, 10, 15);
```

select "functions".get_total(5, 1, 10, 15);

123	get_total
1	31

STRICT-функция

- Ключевое слово **STRICT** позволяет объявить функцию **как строгую**:
 - Функция будет возвращать неопределенное значение (NULL) в случае, если хотя бы один из ее входных параметров не определен (NULL)
 - Тело функции при этом вообще не выполняется

```
CREATE OR REPLACE FUNCTION "functions".hello(IN name text, IN title text DEFAULT 'Mr')  
RETURNS text STRICT  
AS $$  
SELECT 'Dear, ' || title || ' ' || name || '!';  
$$ LANGUAGE sql ;
```

```
SELECT "functions".hello('Svetlana', NULL);
```

	hello
1	[NULL]

```
SELECT "functions".hello('Svetlana');
```

	hello
1	Dear, Mr Svetlana!



Возврат значения

- Значение, возвращаемое функцией можно определить:
 - С помощью **RETURNS** с указанием типа возвращаемого значения
 - С помощью **выходных параметров** с режимом **INOUT** или **OUT**
- **В RETURNS можно указать только одно значение, а выходных параметров может быть несколько!**
- Если используется и выходной параметр и оператор RETURNS, типы данных должны быть согласованы друг с другом

```
CREATE or replace FUNCTION "functions".count_orders (in _emp int, out _par bigint)
RETURNS bigint
as
$$
select count(*) from "Sales"."Orders" as o
where o.empid = _emp;
$$
language sql;
```



Пример

```
CREATE or replace FUNCTION "functions".count_orders (_empid int, out _res bigint)
--RETURNS bigint
as
$$
select count(*) from "Sales"."Orders" as o
where o.empid = _empid;
$$
language sql;
```

```
CREATE or replace FUNCTION "functions".count_orders (inout _par bigint)
--RETURNS bigint
as
$$
select count(*) from "Sales"."Orders" as o
where o.empid = _par;
$$
language sql;
```

Мы передаем входное значение, а
выходное возвращается функцией в
качестве результата



Скалярные функции

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

```
CREATE FUNCTION "functions".get_total_orders1
(OUT bigint -- имя можно не указывать
)
--RETURNS bigint
AS
$$
--тело функции
SELECT count(*) FROM "Sales"."Orders";
$$ LANGUAGE SQL;
```

- Скалярная функция возвращает только одно значение указанного типа
- Если SELECT запрос в теле функции, будет возвращать более одного значения (несколько строк/столбцов) – будет возвращена ошибка

⚠ SQL Error [42P13]: ERROR: return type mismatch in function declared to return bigint
Подробности: Final statement must return exactly one column.
Где: SQL function "get_total_orders"



PostgreSQL

Табличные функции

- Возвращают таблицу (набор записей)
- В SQL используются в разделе FROM



Возврат множества строк

- **RETURNS SETOF *data_type*** – возврат N-значений заданного типа
- **RETURNS SETOF *имя_таблицы*** – возврат всех столбцов из таблицы или пользовательского композитного типа
- **RETURNS TABLE(*имя_столбца тип, ...*)** – возврат всех столбцов из таблицы или пользовательского композитного типа, с возможностью явно указать возвращаемые столбцы
- **RETURNS SETOF record** – возврат записей результирующего набора, типы столбцов в которых заранее неизвестны
 - Мы работаем с результирующим набором, как с **анонимным типом**
 - Это снижает удобство использования результирующего набора
- Возврат через **OUT**-аргументы – возврат **1 записи (record)** с несколькими значениями
 - Название столбцов записи – имена **OUT**-аргументов
 - Чтобы возвращались **все** записи, в случае когда результирующий набор содержит несколько строк, необходимо использовать **RETURNS SETOF record**



Пример 1. Возврат 1 записи (record)

```
CREATE OR REPLACE FUNCTION "functions".count_orders  
    (in _emp int, out _count bigint, out _dt timetz)  
as  
$$  
    select count(*), current_time  
    from "Sales"."Orders" as o  
    where o.empid = _emp;  
$$  
language sql;
```

```
select "functions".count_orders (1);
```

	count_orders
1	(123,10:55:11.198666+03)



Пример 2. Возврат нескольких записей

```
CREATE FUNCTION get_avg_price_by_cat()  
RETURNS SETOF numeric  
AS $$  
    SELECT avg(unitprice::numeric)  
    FROM "Production"."Products"  
    GROUP BY categoryid;  
$$ LANGUAGE SQL;
```

```
select *  
from get_avg_price_by_cat() as avg_price;
```

	123 avg_price
1	20,6825
2	32,37
3	37,9791666667
4	20,25
5	28,73
6	24,6653846154
7	54,0066666667
8	25,16

```
CREATE FUNCTION get_customers_by_country  
    (IN customer_country varchar)  
RETURNS Table  
    (comp_id int, comp_name varchar, comp_city varchar)  
AS $$  
    SELECT custid, companyname, city  
    FROM "Sales"."Customers"  
    WHERE country = customer_country;  
$$ LANGUAGE SQL;
```

```
select *  
from get_customers_by_country('Brazil');
```

	123 comp_id	abc comp_name	abc comp_city
1	15	Customer JUW XK	Sao Paulo
2	21	Customer KIDPX	Sao Paulo
3	31	Customer YJCBX	Campinas
4	34	Customer IBVRG	Rio de Janeiro
5	61	Customer WULWD	Rio de Janeiro
6	62	Customer WFIZJ	Sao Paulo
7	67	Customer QVEPD	Rio de Janeiro
8	81	Customer YQQWW	Sao Paulo
9	88	Customer SRQVM	Resende

Получение данных «анонимного» типа

```
DROP FUNCTION get_price_boundaries

CREATE OR REPLACE FUNCTION get_price_boundaries
(IN cat_id int DEFAULT 4)
RETURNS SETOF record
AS $$
    SELECT MAX(unitprice), MIN(unitprice)
    FROM "Production"."Products"
    WHERE categoryid = cat_id;
$$ LANGUAGE SQL;
```

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

```
select *
from get_price_boundaries()
as (max_price money,
    min_price money);
```

	123 max_price ↕	123 min_price ↕
1	\$55.00	\$2.50

```
select get_price_boundaries();
```

	get_price_boundaries ↕
1	(\$55.00,\$2.50)



SQL Error [42601]: ERROR: a column definition list is required for functions returning "record"
Позиция: 17



Категории изменчивости

- **Категория изменчивости** – определяет свойства значения, возвращаемого функцией, при одинаковых значениях входных параметров
- **Volatile** (по умолчанию)
 - функция вычисляется при каждом вызове
 - возвращаемое значение пересчитывается (может произвольно изменяться)
 - **может содержать операторы, изменяющие состояние БД**
- **Stable**
 - функция выполняется один раз во время выполнения запроса, а затем используется вычисленное значение
 - возвращаемое значение не изменяется в пределах выполнения одного оператора SQL
 - **функция не может содержать операторы, изменяющие состояние БД**
- **Immutable**
 - функция может быть вычислена на этапе планирования запроса
 - возвращаемое значение не изменяется (**функция детерминирована**)
 - **функция не может содержать операторы, изменяющие состояние БД**
- **Планировщик может делать собственные выводы об изменчивости функции, не смотря на указанную категорию**

Категории изменчивости и изоляция транзакций

- **Read Committed**

- функции с изменчивостью **VOLATILE** могут приводить к рассогласованию данных внутри одного запроса, если параллельная транзакция меняет данные
- функции с изменчивостью **STABLE** или **IMMUTABLE** – возвращают согласованные данные

- **Любой уровень**

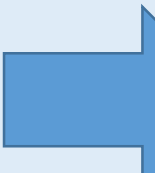
- функции с изменчивостью **VOLATILE** видят все изменения, в том числе сделанные текущим, еще не завершенным оператором SQL
- функции с изменчивостью **STABLE** или **IMMUTABLE** видят изменения только уже завершенных операторов

Пример 1. VOLATILE

```
create table example1."Employees"  
(like "HR"."Employees");
```

Транзакция 1

```
BEGIN ISOLATION LEVEL READ COMMITTED;  
  
SELECT (SELECT  
        count(*) FROM example1."Employees")  
      , "functions".count_rec()  
      , pg_sleep(1)  
FROM generate_series(1,4);  
  
end;
```



```
CREATE FUNCTION "functions".count_rec()  
RETURNS bigint  
VOLATILE  
AS $$  
    SELECT count(*)  
    FROM example1."Employees";  
$$ LANGUAGE sql;
```

Транзакция 2

```
insert into example1."Employees"  
select * from "HR"."Employees"  
where empid =1;
```

	count	count_rec	pg_sleep
1	0	0	
2	0	0	
3	0	0	
4	0	1	



Пример 2. STABLE

```
ALTER FUNCTION "functions".count_rec() STABLE;
```

```
truncate table example1."Employees";
```

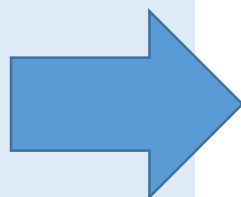
Транзакция 1

```
BEGIN ISOLATION LEVEL READ COMMITTED;
```

```
SELECT (SELECT  
        count(*) FROM example1."Employees")  
      , "functions".count_rec()  
      , pg_sleep(1)
```

```
FROM generate_series(1,4);
```

```
end;
```



Транзакция 2

```
insert into example1."Employees"  
select * from "HR"."Employees"  
where empid =1;
```

	count	count_rec	pg_sleep
1	0	0	
2	0	0	
3	0	0	
4	0	0	

Подстановка тела функции в SQL-запрос

- Тело функции на языке SQL может быть подставлено в основной SQL-оператор на этапе разбора запроса без фактического вызова функции
- Необходимо выполнение следующих условий:
 - тело функции состоит из одного оператора SELECT
 - нет обращений к таблицам
 - не используются подзапросы, группировки и т. п.
 - функция является скалярной (возвращает одно значение)
 - функция не противоречит указанной категории изменчивости
 - Ответственность за неверное указание категории изменчивости – на разработчике



Перегрузка подпрограмм

- Перегрузка подпрограмм позволяет создать одноимённые подпрограммы
 - при вызове подпрограммы сервер определяет, какой вариант использовать по количеству и типу фактических параметров
- Перегружаемые подпрограммы должны отличаться хотя бы одной из следующих характеристик формальных параметров типа **IN** или **INOUT**:
 - количеством
 - порядком
 - типом данных

Перегрузка подпрограмм, замечания

- Функции не могут отличаться только типом возвращаемого значения

```
FUNCTION calc_sal (p_sal IN NUMERIC) RETURN NUMERIC  
FUNCTION calc_sal (p_sal IN NUMERIC) RETURN VARCHAR2 ✗  
FUNCTION calc_sal (p_sal IN NUMERIC, comm IN NUMERIC) RETURNS NUMERIC  
FUNCTION calc_sal() RETURNS NUMERIC
```

- **CREATE OR REPLACE:**

- создаёт новую подпрограмму (перегруженную версию существующей), если изменить характеристики параметров **IN** и **INOUT**
 - **выдаёт ошибку, если сигнатура функции отличается от существующей только типом возвращаемого значения**
- Использование значений по умолчанию в перегруженных подпрограммах может привести к ошибкам из-за неоднозначности