# Лабораторная работа № 7

**Создание пользовательских функций в PostgreSql**

**Синтаксис команды**

**CREATE**[ OR REPLACE ] **FUNCTION**  
имя\_функции**([ [** метод\_аргумента **][**имя\_аргумента **]** тип\_ аргумента **[,…] ])**  
**RETURNS**тип\_возвращаемого\_значения   
**AS  '**определение**'   
LANGUAGE  '**язык**'  
 [ WITH (**атрибут**[…])]**

* **CREATE FUNCTION** *имя\_функции* ([[ *метод\_аргумента* ] [*имя\_аргумента* ] *тип\_ аргумента* [,…] ]) - после ключевых слов CREATE FUNCTION указывается имя создаваемой функции, после чего в круглых скобках перечисляются аргументы, разделенные запятыми. Для каждого аргумента достаточно указать только тип, но при желании можно задать метод (in, out, inout; по умолчанию in) и  имя.  Если список в круглых скобках пуст, функция вызывается без аргументов (хотя сами круглые скобки обязательно должны присутствовать как в определении функции, так и при ее использовании). Ключевые слова  OR REPLACE  используются для изменения уже существующей функции.

* **RETURNS** тип\_возвращаемого\_значения   -  тип данных, возвращаемый функцией.

* **AS** '*определение*'  -  программное определение функции. В процедурных языках (таких, как PL/pgSQL) оно состоит из кода функции. Для откомпилированных функций С указывается абсолютный системный путь к файлу, содержащему объектный код.
* **LANGUAGE** '*язык*'. Название языка, на котором написана функция. В аргументе может передаваться имя любого процедурного языка (такого, как plpgsql или plperl, если соответствующая поддержка была установлена при компиляции),  С или SQL.   
  **[**

**WITH** ( *атрибут* [. ...] ) ] - *атрибут* может принимать два значения: iscachable  и  isstrict.  
      **iscachable**. Оптимизатор может использовать предыдущие вызовы функций для ускоренной обработки будущих вызовов с тем же набором аргументов. Кэширование обычно применяется при работе с функциями, сопряженны­ми с большими затратами ресурсов, но возвращающими один и тот же ре­зультат при одинаковых значениях аргументов.  
       **isstrict**. Функция всегда возвращает NULL в случае, если хотя бы один из ее аргументов равен NULL. При передаче атрибута isstrict результат возвраща­ется сразу, без фактического выполнения функции.

**1).**

--Создание функции, возвращающей Hello World

CREATE FUNCTION fun\_hello\_world()

RETURNS text

LANGUAGE 'sql'

AS $BODY$

SELECT 'Hello World!';

$BODY$;

-- ВЫЗОВ ФУНКЦИИ

select fun\_hello\_world();

**2).**

CREATE FUNCTION sum\_n\_product(x int, y int, OUT sum int, OUT prod int) AS $$

BEGIN

sum := x + y;

prod := x \* y;

END;

$$ LANGUAGE plpgsql;

-- ВЫЗОВ ФУНКЦИИ

select sum\_n\_product(10,20);

### **3). Объявление параметров функции**

Переданные в функцию параметры именуются идентификаторами $1, $2 и т. д. Либо псевдоним, либо цифровой идентификатор используются для обозначения параметра.

Создать псевдоним можно двумя способами. Предпочтительный способ это дать имя параметру в команде CREATE FUNCTION, например:

CREATE FUNCTION sales\_tax(subtotal real) RETURNS real AS $$

BEGIN

RETURN subtotal \* 0.06;

END;

$$ LANGUAGE plpgsql;

select sales\_tax(100);

Другой способ - это явное объявление псевдонима при помощи синтаксиса:

***имя*** ALIAS FOR $***n***;

Предыдущий пример для этого стиля выглядит так:

DROP FUNCTION sales\_tax;

CREATE OR REPLACE FUNCTION sales\_tax(real) RETURNS real AS $$

DECLARE

subtotal ALIAS FOR $1;

BEGIN

RETURN subtotal \* 0.06;

END;

$$ LANGUAGE plpgsql;

select sales\_tax(100);

CREATE OR REPLACE FUNCTION concat\_selected\_fields(in\_t sometablename) RETURNS text AS $$

BEGIN

RETURN in\_t.f1 || in\_t.f3 || in\_t.f5 || in\_t.f7;

END;

$$ LANGUAGE plpgsql;

**4).**Структура программного блока PL/pgSQL.

CREATE FUNCTION a\_function () RETURNS int4 AS $$

DECLARE

an\_integer int4;

BEGIN

an\_integer:=10\*10;

return an\_integer;

END;

$$ LANGUAGE plpgsql;

Вызов функции

SELECT a\_function() AS output;

5).Создание функции, возвращающей  целое значение (использование select into)

CREATE OR REPLACE FUNCTION apf (integer) RETURNS integer AS '

DECLARE

i integer;

-- Функция вычисляет количество работников в отделе $1

BEGIN

select count(\*) from emp into i where deptno = $1;

-- возвращение результата

return i;

END;

' LANGUAGE plpgsql;

Вызов функции

SELECT apf(10) AS output;

select \* from emp where deptno = 10;

6).Создание функции, возвращающей  вещественное значение (использование select into)

CREATE OR REPLACE FUNCTION avgves (integer) RETURNS real AS '

DECLARE

aves real;

-- Функция вычисляет среднюю з/п в отделе $1

BEGIN

select avg(sal) from emp into aves where deptno=$1;

-- возвращение результата

return aves;

END;

' LANGUAGE plpgsql;

SELECT distinct deptno, apf(10) AS kol, avgves(10) as sves

FROM emp where deptno = 10 ;

7). Создание функции, возвращающей столбец текстовых значений

CREATE OR REPLACE FUNCTION onef (integer) RETURNS SETOF character AS '  
--  Функция возвращает имена работников с зарплатой больше  $1  
select ename from emp where sal>$1;  
' LANGUAGE sql;

Использование функции, возвращающей столбец текстовых значений

SELECT  onef(3000) AS sname ;

8). Создание функции, возвращающей столбец записей

CREATE OR REPLACE FUNCTION manyf (integer) RETURNS SETOF record AS '

-- Функция возвращает информацию о работниках с з.п. больше $1

select ename, deptno, sal from emp where sal>$1 order by deptno;

' LANGUAGE sql;

SELECT manyf(3000) AS sname ;

9). Создание функции, возвращающей курсор

CREATE OR REPLACE FUNCTION fun1 (integer) RETURNS refcursor AS '

-- если входной параметр - 0 - функция возвращает список сотрудников отдела

-- если входной параметр - 1 - функция возвращает список отделов

DECLARE

ref refcursor;

BEGIN

if $1=0 then

OPEN ref FOR SELECT \* FROM emp ;

else

OPEN ref FOR SELECT \* FROM dept ;

end if;

RETURN ref;

END;

' LANGUAGE plpgsql;

select \* from fun1(0) ;

FETCH ALL From "<unnamed portal 1>"

select \* from fun1(1) ;

FETCH ALL From "<unnamed portal 2>"

# Массивы

Массив – это тип данных, в котором хранится упорядоченный набор однотипных элементов. Обращаться к элементу массива можно по целочисленному индексу в квадратных скобках.

*массив*[*индекс*]

Можно извлечь несколько соседних элементов («срез массива»).

*массив*[*нижний\_индекс*:*верхний\_индекс*]

По умолчанию, нумерация элементов массива начинается с 1. Массивы могут быть многомерными, но они должны прямоугольными.

Для того, чтобы объявить массив нужно добавить квадратные скобки к типу.

names text[];

dates date[];

a int[][];

Указывать размер массива в квадратных скобках не надо, т.к. он игнорируется. PostgreSQL не накладывает ограничения на размер массива.

Инициализировать массив можно несколькими способами.

10). По-элементная инициализация.

DO $$

DECLARE

arr int[];

BEGIN

arr[3] = 3;

arr[5] = 5;

arr[6] = 6;

FOR i IN 1..6 LOOP

RAISE NOTICE 'arr[%]=%', i, arr[i];

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: arr[1]=<NULL>

ЗАМЕЧАНИЕ: arr[2]=<NULL>

ЗАМЕЧАНИЕ: arr[3]=3

ЗАМЕЧАНИЕ: arr[4]=<NULL>

ЗАМЕЧАНИЕ: arr[5]=5

ЗАМЕЧАНИЕ: arr[6]=6

Как видим, не инициализированные элементы массива не определены. В связи с этим возникает вопрос, а какой реальный размер массива и с какого индекса начинается отсчет элементов. Для этого воспользуемся встроенными функциями:

* array\_length(), определяющую длину массива;
* array\_lower(), нижняя граница индекса массива;
* array\_upper(), верхняя граница индекса массива.

11).

DO $$

DECLARE

arr int[];

BEGIN

arr[3] = 3;

arr[5] = 5;

arr[6] = 6;

RAISE NOTICE 'длина arr[]=%', array\_length(arr, 1 );

FOR i IN array\_lower(arr, 1)..array\_upper(arr, 1) LOOP

RAISE NOTICE 'arr[%]=%', i, arr[i];

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: длина arr[]=4

ЗАМЕЧАНИЕ: arr[3]=3

ЗАМЕЧАНИЕ: arr[4]=<NULL>

ЗАМЕЧАНИЕ: arr[5]=5

ЗАМЕЧАНИЕ: arr[6]=6

При желании, мы можем заполнять массив, начиная с любого индекса.

Инициализация с помощью конструктора массива. Конструктор простого массива состоит из ключевого слова ARRAY.

names text[] = ARRAY[‘Николай’, ‘Андрей’, ‘Иван’, ‘Сергей’];

dates date[] = ARRAY['15.01.2023', '20.02.2023','30.03.2023'];

a int[][] = ARRAY[ARRAY[10, 11, 12], ARRAY[13, 14, 15]];

b int[][] = ARRAY[[10, 11, 12], [13, 14, 15]];

Применим.

12).

DO $$

DECLARE

arr varchar(10)[] = ARRAY['один', 'два','три'];

BEGIN

FOR i IN 1..3 LOOP

RAISE NOTICE 'arr[%]=%', i, arr[i];

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: arr[1]=один

ЗАМЕЧАНИЕ: arr[2]=два

ЗАМЕЧАНИЕ: arr[3]=три

Также возможно создать массив из результатов подзапроса. В этом случае конструктор массива записывается так же с ключевым словом ARRAY, за которым в круглых скобках следует подзапрос. Подзапрос должен возвращать один столбец.

DO $$

DECLARE

codes char(3)[] = ARRAY(SELECT aircraft\_code FROM aircrafts);

BEGIN

FOR i IN 1..array\_upper(codes, 1) LOOP

RAISE NOTICE 'codes[%]=%', i, codes[i];

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: codes[1]=773

ЗАМЕЧАНИЕ: codes[2]=763

ЗАМЕЧАНИЕ: codes[3]=SU9

ЗАМЕЧАНИЕ: codes[4]=321

ЗАМЕЧАНИЕ: codes[5]=733

ЗАМЕЧАНИЕ: codes[6]=CN1

ЗАМЕЧАНИЕ: codes[7]=CR2

ЗАМЕЧАНИЕ: codes[8]=320

ЗАМЕЧАНИЕ: codes[9]=319

Приведем пример использования двумерного массива.

DO $$

DECLARE

arr text[][] = '{{"один","два"}, {"десять", "одиннадцать"}}';

BEGIN

FOR i IN 1..array\_upper(arr, 1) LOOP

FOR j IN 1..array\_upper(arr, 2) LOOP

RAISE NOTICE 'arr[%][%]=%', i, j, arr[i][j];

END LOOP;

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: arr[1][1]=один

ЗАМЕЧАНИЕ: arr[1][2]=два

ЗАМЕЧАНИЕ: arr[2][1]=десять

ЗАМЕЧАНИЕ: arr[2][2]=одиннадцать

Надо отметь, что после инициализации многомерного массива нельзя изменять его размер.

Для массива есть специальный цикл FOREACH, в котором 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.

Пример цикла по элементам двумерного массива.

DO $$

DECLARE

arr text[][] = ARRAY[ARRAY['один','два'],

ARRAY['десять', 'одиннадцать'],

ARRAY['двадцать', 'двадцать один']];

item text;

BEGIN

FOREACH item IN ARRAY arr LOOP

RAISE NOTICE 'item=%', item;

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: item=один

ЗАМЕЧАНИЕ: item=два

ЗАМЕЧАНИЕ: item=десять

ЗАМЕЧАНИЕ: item=одиннадцать

ЗАМЕЧАНИЕ: item=двадцать

ЗАМЕЧАНИЕ: item=двадцать один

При положительном значении SLICE выполняет итерации по срезам массива, а не по отдельным элементам. Значение SLICE должно быть целым числом, не превышающим размерности массива. Переменная *цель* должна быть массивом, который получает последовательные срезы исходного массива, где размерность каждого среза задаётся значением SLICE.

DO $$

DECLARE

arr text[][] = ARRAY[ARRAY['один','два'],

ARRAY['десять', 'одиннадцать'],

ARRAY['двадцать', 'двадцать один']];

row text[];

BEGIN

FOREACH row SLICE 1 IN ARRAY arr LOOP

RAISE NOTICE 'row=%', row;

END LOOP;

END;

$$;

Результат.

ЗАМЕЧАНИЕ: row={один,два}

ЗАМЕЧАНИЕ: row={десять,одиннадцать}

ЗАМЕЧАНИЕ: row={двадцать,"двадцать один"}

PostgreSQL позволяет создавать в таблицах столбцы, в которых содержатся массивы. Эти массивы могут быть многомерными и могут содержать значения любого из встроенных типов, а также типов данных, определенных пользователем.

Предположим, что нам необходимо сформировать и сохранить в базе данных в удобной форме графики работы пилотов авиакомпании, т. е. номера дней недели, когда они совершают полеты.

Создадим таблицу, в которой эти графики будут храниться в виде одномерных массивов.

|  |
| --- |
| CREATE TABLE pilots  (  pilot\_name text,  schedule integer[]  ); |

Для указания на то, что это массив, нужно добавить квадратные скобки к наименованию типа данных. При этом задавать число элементов не обязательно. Заполним таблицу четырьмя записями.

Массив в команде вставки представлен в виде строкового литерала с указанием типа данных и квадратных скобок, означающих массив. Обратите внимание, что все массивы имеют различное число элементов.

|  |
| --- |
| INSERT INTO pilots  VALUES  (‘Ivan', ‘{1, 3, 5, 6, 7}'::integer[] ),  (‘Petr', ‘{1, 2, 5, 7}'::integer[] ),  (‘Pavel', ‘{2, 5}'::integer[] ),  (‘Boris', ‘{3, 5, 6}'::integer[] ); |

Предположим, что руководство компании решило, что каждый пилот должен летать 4 раза в неделю. Значит, нам придется обновить значения в таблице. Пилоту по имени Boris добавим один день с помощью операции конкатенации:

UPDATE pilots SET schedule = schedule || 7 WHERE pilot\_name = 'Boris';

Пилоту по имени Pavel добавим один день в конец массива с помощью функции array\_append:

UPDATE pilots

SET schedule = array\_append( schedule, 6 )

WHERE pilot\_name = 'Pavel';

Ему же добавим один день в начало списка с помощью функции array\_prepend (обратите внимание, что параметры функции поменялись местами):

UPDATE pilots

SET schedule = array\_prepend( 1, schedule )

WHERE pilot\_name = 'Pavel';

У пилота по имени Ivan имеется лишний день в графике. С помощью функции array\_remove удалим из графика пятницу (второй параметр функции указывает значение элемента массива, а не индекс):

UPDATE pilots

SET schedule = array\_remove( schedule, 5 )

WHERE pilot\_name = 'Ivan';

У пилота по имени Petr изменим дни полетов, не изменяя их общего количества. Воспользуемся индексами для работы на уровне отдельных элементов массива. По умолчанию нумерация индексов начинается с единицы, а не с нуля. При необходимости ее можно изменить. К элементам одного и того же массива можно обращаться в предложении SET по отдельности, как будто это разные столбцы.

UPDATE pilots

SET schedule[1] = 2, schedule[2] = 3

WHERE pilot\_name = 'Petr';

А можно было бы, используя срез (slice) массива, сделать и так:

UPDATE pilots

SET schedule[ 1:2 ] = ARRAY[ 2, 3 ]

WHERE pilot\_name = 'Petr';

Теперь продемонстрируем основные операции, которые можно применять к массивам, выполняя выборки из таблиц. Получим список пилотов, летающих по средам:

SELECT \*

FROM pilots

WHERE array\_position( schedule, 3 ) IS NOT NULL;

Функция array\_position возвращает индекс первого вхождения элемента с указанным значением в массив. Если же такого элемента нет, она возвратит NULL. Выберем пилотов, летающих по понедельникам и воскресеньям:

SELECT \*

FROM pilots

WHERE schedule @> '{ 1, 7 }'::integer[];

Оператор @> означает проверку того факта, что в левом массиве содержатся все элементы правого массива. Конечно, при этом в левом массиве могут находиться и другие элементы, что мы и видим в графике этого пилота. Еще аналогичный вопрос: кто летает по вторникам и/или по пятницам? Для получения ответа воспользуемся оператором &&, который проверяет наличие общих элементов у массивов, т. е. пересекаются ли их множества значений. В нашем примере число общих элементов, если они есть, может быть равно одному или двум. Здесь мы использовали нотацию с ключевым словом ARRAY, а не '{2, 5}'::integer[]. Так же можно применять ту, которая принята в рамках выполнения вашего проекта.

SELECT \*

FROM pilots

WHERE schedule && ARRAY[ 2, 5 ];

Сформулируем вопрос в форме отрицания: кто не летает ни во вторник, ни в пятницу? Для получения ответа добавим в предыдущую SQL-команду отрицание NOT:

SELECT \*

FROM pilots

WHERE NOT ( schedule && ARRAY[ 2, 5 ] );

Иногда требуется развернуть массив в виде столбца таблицы. В таком случае поможет функция unnest:

SELECT unnest( schedule ) AS days\_of\_week

FROM pilots

WHERE pilot\_name = 'Ivan';