Основные типы данных и функции

Логический тип

BOOLEAN — трехзначная логика: истина (true), ложь (false) и «неизвестно» (null) (1 байт)

Логические операции

and, or — логические «И» (конъюнкция) и «ИЛИ» (дизъюнкция)

```
            and  or

f   f       f    f
f   t       f    t
t   f       f    t
t   t       t    t

N   f       f    N
f   N       f    N
N   t       N    t
t   N       N    t
N   N       N    N
```

not — логическое «НЕ» (отрицание)

```
     not

f    t
t    f

N    N
```

bool_and, bool_or — агрегатные варианты логических «И» и «ИЛИ»

```
select bool_and(b) from (values (true), (false)) t(b) → f
select bool_and(b) from (values (true), (null))  t(b) → t  — null не учитывается

select bool_or(b)  from (values (true), (false)) t(b) → t
select bool_or(b)  from (values (false), (null)) t(b) → f  — null не учитывается
```

Операторы сравнения

Возвращают логический тип и определены для всех типов данных, для которых имеют смысл.

<, >, <=, >=, =, <> (!=) — меньше, больше, меньше или равно, больше или равно, не равно

```
'Привет' < 'мир' → f  — сравнение строк зависит от правил сортировки (collation)
1 = 1    → t
1 = null → null  — большинство операторов возвращают null для неопределенных аргументов
```

is [not] distinct from — (не) отличается; неопределенные значения считаются одинаковыми

```
1    is distinct from null → t
1    is distinct from 1    → f
null is distinct from null → f
```

is [not] null — проверка на неопределенность

```
1    is null → f
null is null → t
```

is [not] true, is [not] false — проверка на истинность или ложность

	is true	= true	is not false	is false	= false	is not true
f	f	f	f	**t**	**t**	**t**
t	**t**	**t**	**t**	f	f	f
N	f	N	**t**	f	N	**t**

[not] between — находится между

```
X     between A and B  =  A <= X and X <= B
X not between A and B  =  X < A  or  X > B

2 between 1 and 3 → t
2 between 3 and 1 → f
1 between 2 and 3 → f
```

[not] between symmetric — находится между (без учета порядка)

```
2 between symmetric 3 and 1 → t
```

Обработка null

coalesce — замена неопределенного значения [коулéс]

```
coalesce(null, 'no')  → no
coalesce('yes', 'no') → yes
coalesce(null, null, 'no', f()) → no  — принимает любое число аргументов; f() не будет вычисляться
```

nullif — замена значения на null

```
nullif('no', 'no')  → N
nullif('yes', 'no') → yes
```

Числа

Целые

SMALLINT — от −32 768 до +32 767 (2 байта)

INTEGER, INT — от −2 147 483 648 до +2 147 483 647 (4 байта)

BIGINT — примерно от -10^{19} до $+10^{19}$ (8 байт)

SMALLSERIAL
SERIAL
BIGSERIAL
} с автоувеличением

С плавающей точкой

REAL — около 6 десятичных цифр (4 байта)

FLOAT, DOUBLE PRECISION — около 15 десятичных цифр (8 байт)

```
-Infinity   минус бесконечность
+Infinity   плюс бесконечность
NaN         не число

0.1::real * 10 = 1.0::real → f  — значения хранятся не точно, осторожно при сравнениях!
```

С фиксированной точностью

NUMERIC, DECIMAL — 131 072 цифр до десятичной точки и 16 383 — после (переменная длина)

NUMERIC(N) — целые числа до N цифр (N ≤ 1000)

NUMERIC(N, M) — вещественные числа до N цифр, из них M после десятичной точки

```
NaN         не число
```

Арифметика

+, −, *, / — плюс, минус, умножить, разделить

div — целочисленное деление

```
div(7.0,2.0) → 3 → trunc(7.0/2.0)
```

mod, % — остаток от деления

```
mod(7,2) → 7%2 → 1
```

power, ^ — возведение в степень

```
power(2,3) → 2^3 → 8
```

abs — абсолютное значение

```
abs(-2.7) → 2.7
```

sign — знак

```
        -2.7  0   2.7
sign    -1    0   1
```

trunc, ceil (ceiling), round, floor — округление

```
        -2.7  2.7

trunc   -2    2
ceil    -2    3
round   -3    3
floor   -3    2
```

Приведение типов

to_number — строка к числу (см. коды форматирования)

```
9     цифра                          0     цифра с ведущим нулем
.     десятичная точка               D     точка или запятая (из локали)
,     разделитель разрядов           G     разделитель разрядов (из локали)
MI    минус (<0)                     PL    плюс (>0)                SG    плюс или минус

to_number('3,1416',    '99D00') → 3.14
to_number('3,1416','99D000000') → 3.1416
to_number('123,45',    '99D00') → numeric field overflow
```

Текстовые строки

CHAR(N) — строка фиксированной длины, дополненная пробелами

VARCHAR(N) — строка с ограничением максимальной длины

TEXT, VARCHAR — строка без ограничения длины

```
'What''s up?'
$$What's up?$$
$function$BEGIN; RETURN $$What's up?$$; END;$function$
E'col1\tcol2\nval1\tval2'
```

Выделение части

length, char_length — длина строки в символах

```
length('Привет, мир!') → char_length('Привет, мир!') → 12
```

octet_length — длина строки в байтах

```
octet_length('Привет, мир!') → 21  — зависит от кодировки
```

position, strpos — позиция подстроки

```
position('мир' in 'Привет, мир!') → strpos('Привет, мир!', 'мир') → 9
```

substring — выделение подстроки

```
substring('Привет, мир!' from 9 for 3) → substr('Привет, мир!', 9, 3) → мир
substring('Привет, мир!' from 9)       → substr('Привет, мир!', 9)    → мир!
```

left, right — подстрока слева или справа

```
left( 'Привет, мир!', 6) → Привет
right('Привет, мир!', 4) → мир!
```

Изменение

overlay — замена подстроки

```
overlay('Привет, мир!' placing 'PostgreSQL' from 9 for 3) → Привет, PostgreSQL!
```

replace — замена всех вхождений подстроки

```
replace('Привет, мир!', 'р', 'ppp') → Прррivет, миррр!
```

translate — замена символов по соответствию

```
translate('Привет, мир!', 'Првтмие', 'Prvtm') → Prvt, mr!
```

lower, upper, initcap — преобразование регистра (зависит от CTYPE)

```
lower('Привет, мир!')   → привет, мир!
upper('Привет, мир!')   → ПРИВЕТ, МИР!
initcap('Привет, мир!') → Привет, Мир!
```

trim, ltrim, rtrim, btrim — отрезание символов с концов строки (по умолчанию — пробелы)

```
trim( leading 'Пр!' from 'Привет, мир!') → ltrim('Привет, мир!', 'Пр!') → ивет, мир!
trim(trailing 'Пр!' from 'Привет, мир!') → rtrim('Привет, мир!', 'Пр!') → Привет, ми
trim(    both 'Пр!' from 'Привет, мир!') → btrim('Привет, мир!', 'Пр!') → ивет, ми
```

lpad, rpad — дополнение слева или справа (по умолчанию — пробелами)

```
lpad('Привет, мир!', 17, '. ') → . . .Привет, мир!
rpad('Привет, мир!', 17, '. ') → Привет, мир!. . .
```

reverse — переворачивает строку

```
reverse('Привет, мир!') → !рим ,тевирП
```

Конструирование

concat, concat_ws — склейка строк (произвольное число аргументов)

```
concat('Привет,', ' ', 'мир!') → 'Привет,' || ' ' || 'мир!' → Привет, мир!
concat_ws(', ', 'Привет', 'о', 'мир!') → Привет, о, мир!
```

string_agg — агрегация строк (см. агрегатные функции)

```
string_agg(s, ', ' order by id) from (values (2,'мир!'), (1,'Привет')) v(id,s)
→ Привет, мир!
```

repeat — повторение строки

```
repeat('Привет, 2) → ПриветПривет
```

chr — символ по коду (зависит от кодировки)

```
chr(34) → "
```

Взятие в кавычки и экранирование для динамического SQL

quote_ident — представление строки в виде идентификатора

```
quote_ident('id') → id
quote_ident('foo bar') → "foo bar"
```

quote_literal, quote_nullable — представление в виде строкового литерала

```
quote_literal('id')           → quote_nullable('id')           → 'id'
quote_literal($$What's up?$$) → quote_nullable($$What's up?$$) → 'What''s up?'
quote_literal(null)  → N
quote_nullable(null) → NULL
```

format — форматированный текст а-ля sprintf

```
format('Привет, %s!', 'мир') → Привет, мир!
format('UPDATE %I SET s = %L', 'tbl', $$What's up?$$)
→ 'UPDATE '||quote_ident('tbl')||' SET s = '||quote_nullable($$What's up?$$)
→ UPDATE tbl SET s = 'What''s up?'
```

Приведение типов

to_char — число к строке (см. коды форматирования)

```
9     цифра                          0     цифра с ведущим нулем
.     десятичная точка               D     точка или запятая (из локали)
,     разделитель разрядов           G     разделитель разрядов (из локали)
RN    римскими цифрами
EEEE  экспоненциальная запись
MI    минус (<0)                     PL    плюс (>0)                SG    плюс или минус
FM    без ведущих нулей и пробелов

to_char(3.1416, 'FM99D00')        → 3,14
to_char(3.1416, 'FM99D000000')    → 3,141600
to_char(56789,     '999G999G999') →       56 789
to_char(123456789, '999G999G999') →  123 456 789
to_char(123456789, '999G999G999') → -123 456 789
```

to_char — дата к строке (см. коды форматирования)

```
YYYY  год
MM    месяц (01-12)                  MON   месяц (сокр.)            MONTH месяц полностью
DD    день (01-31)
D     номер дня недели (1-7)         DY    день недели (сокр.)      DAY   день недели
HH    час (01-12)                    HH24  час (00-23)
MI    минуты
SS    секунды
TZ    часовой пояс                   OF    смещение часового пояса
FM    без ведущих пробелов           TM    перевод для дней и месяцев

to_char(now(), 'DD.MM.YYYY HH24:MI:SSOF') → 05.10.2016 10:51:08+03
to_char(now(), 'FMDD TMmonth YYYY, day')  → 5 октября 2016, среда
```

Сравнение по шаблону

like (~~), not like (!~~) — сравнение по шаблону

```
_     1 любой символ
%     ≥0 символов

'Привет, мир!' like 'Привет_ %'            → t
'Привет_мир!'  like 'Привет\_%' escape '\' → t
```

ilike (~~*), not ilike (!~~*) — сравнение по шаблону без учета регистра

```
'Привет, мир!' ilike 'привет, мир!'        → t
```

Регулярные выражения SQL

similar to — сопоставление

```
_     1 любой символ
%     ≥0 символов
*     повтор предыдущего ≥0 раз     {m}   повтор m раз            |     альтернативы
+     повтор ≥1 раза                {m,}  повтор ≥m раз           (…)   группировка
?     повтор 0 или 1 раз            {m,n} повтор от m до n раз    […]   класс символов

'Привет, мир!' similar to 'Привет_ %'                   → t
'-3.14' similar to '(#+|-)?[0-9]+(.[0-9]*)?' escape '#' → t
'24.сен.2016' similar to '_{1,2}._{3}._{4}'             → t
```

substring — выделение подстроки

```
substring('Привет, мир!' from 'Привет, \"%\"!' for '\') → мир
```

split_part — одно из слов по разделителю

```
split_part('Привет, мир!', ', ', 1) → Привет
split_part('Привет, мир!', ', ', 2) → мир!
```

Регулярные выражения POSIX

~, !~ — сопоставление

```
.     1 любой символ
*     повтор предыдущего ≥0 раз     {m}   повтор m раз            |     альтернативы
+     повтор ≥1 раза                {m,}  повтор ≥m раз           (…)   группировка
?     повтор 0 или 1 раз            {m,n} повтор от m до n раз    […]   класс символов
^     начало строки                 $     конец строки

'Привет, мир!' ~ 'Привет'       → t
'Привет, мир!' ~ '^Привет$'     → f
'Привет, мир!' ~ '^Привет. .*$' → t
'-3.14' ~ '(\+|-)?[0-9]+(\.[0-9]*)?' → t
'24.сен.2016' ~ '.{1,2}\..{3}\..{4}' → t

\m    начало слова                  \M    конец слова
\d    цифра                         \D    не цифра
\s    пробельный символ             \S    не пробельный символ
\w    буква или цифра               \W    не буква и не цифра
\n    конец строки                  \t    табуляция

'Привет, мир!' ~ '\mмир\M' → t
'Приветмир!'   ~ '\mмир\M' → f
'-3.14' ~ '[+-]?\d+(\.\d*)?' → t
'24.сен.2016' ~ '\d{1,2}\.\w{3}\.\d{4}' → t
```

~*, !~* — сопоставление без учета регистра

```
'Привет, мир!' ~* 'привет'     → t
'Привет, мир!' ~  '(?i)привет' → t  — другой способ с (?i) в начале шаблона
```

substring — выделение подстроки

```
?     «не жадный» квантор (для *, +, ?, {})

substring('Привет, мир!' from '.*p')  → Привет, мир
substring('Привет, мир!' from '.*?p') → Пр

(?=…) положительный просмотр вперед
(?!…) отрицательный просмотр вперед

substring('Привет, мир!' from '\w+,')     → Привет,
substring('Привет, мир!' from '\w+(?=,)') → Привет
substring('Привет, мир!' from '\w+(?!,)') → мир
```

regexp_matches — выделение подстрок

```
regexp_matches('Привет, мир!', '\w+')           → {Привет}
regexp_matches('Привет, мир!', '(\w+).*?(\w+)') → {Привет,мир}
regexp_matches('Привет, мир!', '\w+', 'g')      → {Привет}
                                                  {мир}     — 2 строки
regexp_matches('Привет, мир!', '\d')            — 0 строк
```

regexp_replace — замена

```
regexp_replace('Привет, мир!', '\w+', 'Hello')            → Hello, мир!
regexp_replace('Привет, мир!', '(\w+).*?(\w+)', '\2, \1') → мир, Привет!
regexp_replace('Привет, мир!', '\w+', '_', 'g')           → _, _!
```

regexp_split_to_array — разбиение в массив

```
regexp_split_to_array('Привет,  мир!', ',\s+') → {Привет,мир!}
```

regexp_split_to_table — разбиение в таблицу

```
regexp_split_to_table('Привет,  мир!', ',\s+') → Привет
                                                 мир!   — 2 строки
```

Даты и времена

DATE — дата без времени суток (4 байта)

```
date '2016-12-31'

make_date(2016,12,31)
```

TIME — время без даты (8 байт)

```
time '23:59:59.999'

make_time(23,59,59.999)
```

TIME WITH TIME ZONE — время с часовым поясом (12 байт)

TIMESTAMP — дата и время (8 байт)

```
timestamp '2016-12-31 23:59:59.999'

make_timestamp(2016,12,31,23,59,59.999)
```

TIMESTAMP WITH TIME ZONE, TIMESTAMPTZ — дата и время с часовым поясом (8 байт)

```
timestamptz '2016-12-31 23:59:59.999 MSK'

make_timestamptz(2016,12,31,23,59,59.999,'MSK')
```

INTERVAL — временной интервал (16 байт)

```
interval '1 year 4 months 12 days 03:17:23 ago'

make_interval(-1,-4,0/*weeks*/,-12,-3,-17,-23)
```

Арифметика

+, − — увеличить (уменьшить) дату/время на интервал

```
date '2016-12-31' + 1 → 2017-01-01  — для DATE интервалом является целое число дней
timestamp '2016-12-31 23:50:01' + interval '9 minutes 59 seconds' → 2017-01-01 00:00:00
```

− — интервал между двумя датами/временами

```
date '2016-12-31' - date '2016-12-01' → 30  — для DATE интервалом является целое число дней
timestamp '2016-12-31 23:59:59' - timestamp '2016-12-01 23:50:00' → 30 days 00:09:59
```

***, /** — увеличить (уменьшить) интервал в указанное число раз

```
5 * interval '1 day' → 5 days
interval '5 day' / 2 → 2 days 12:00:00
```

Функции

overlaps — пересекаются ли интервалы

```
(time '12:00', time '14:00')       overlaps (time '13:00', time '15:00')       → t
(time '12:00', interval '2 hours') overlaps (time '13:00', interval '2 hours') → t
```

date_trunc — округление даты/времени или интервала

```
year         hour
month        minute
day          second

date_trunc('month', timestamp '2016-12-31 23:59:59')   → 2016-12-01 00:00:00
date_trunc('minutes', interval '9 minutes 59 seconds') → 00:09:00
```

Текущее время

current_date, localtime, localtimestamp — начало транзакции (без часового пояса)

current_time, current_timestamp = transaction_timestamp() = now() — начало транзакции (с часовым поясом)

statement_timestamp() — начало оператора (с часовым поясом)

clock_timestamp() — фактическое текущее время (с часовым поясом)

Выделение частей

extract, date_part — выделение отдельных полей из даты/времени (результат типа double precision)

```
year         hour         isodow
month        minute       timezone
day          second       timezone_hour

extract(year from   timestamp '2016-12-31 23:59:59')
→ date_part('year', timestamp '2016-12-31 23:59:59') → 2016
extract(isodow from timestamp '2016-12-31 23:59:59') → 6    — 1..7 (пн..вс)
```

Приведение типов

to_date — строка к дате (см. коды форматирования)

```
YYYY  год
MM    месяц (01-12)      MON   месяц (сокр.)            MONTH месяц полностью
DD    день (01-31)       DY    день недели (сокр.)      DAY   день недели

to_date('31.12.2016', 'DD.MM.YYYY') → 2016-12-31
to_date('32.13.2016', 'DD.MM.YYYY') → 2017-02-01  — корректность аргументов не проверяется
```

to_timestamp — строка к дате и времени с часовым поясом (см. коды форматирования)

```
YYYY  год
MM    месяц (01-12)      MON   месяц (сокр.)            MONTH месяц полностью
DD    день (01-31)       DY    день недели (сокр.)      DAY   день недели
HH    час (01-12)        HH24  час (00-23)
MI    минуты
SS    секунды

to_timestamp('31.12.2016 23:59:59', 'DD.MM.YYYY HH24:MI:SS') → 2016-12-31 23:59:59+03
to_timestamp('32.13.2016', 'DD.MM.YYYY') → 2017-02-01 00:00:00+03 — корректность аргументов не проверяется
```

Составные типы

CREATE TYPE AS ( ) — составной тип как объект базы данных

```
CREATE TYPE monetary AS (amount numeric, currency text);
CREATE TABLE uom(units text, value numeric);  — неявно определяет одноименный составной тип

monetary '(25.10,"USD")'
ROW(25.10,'USD')::monetary      (25.10,'USD')::monetary

((25.10,'USD')::monetary).amount   → 25.10
((25.10,'USD')::monetary).currency → USD
```

%ROWTYPE — составной тип, соответствующий строке таблицы (PL/pgSQL)

```
DECLARE
  m monetary%rowtype;
BEGIN
  m := (25.10,'USD')::monetary;
END;
```

RECORD — анонимный составной тип без заранее определенной структуры (PL/pgSQL)

```
DECLARE
  r record;
BEGIN
  r := (25.10,'USD')::monetary;
  SELECT * INTO r FROM uom LIMIT 1;
END;
```

Массивы

ТИП[ ] — массив элементов указанного типа

```
text[]
integer[][]  — двумерный массив

'{"Привет","мир!"}'             ARRAY['Привет','мир!']
'{{1,2,3},{10,20,30}}'          ARRAY[[1,2,3], [10,20,30]]

(ARRAY['Привет','мир!'])[1] → Привет  — по умолчанию индекс начинается с 1
(ARRAY['Привет','мир!'])[2] → мир!
(ARRAY['Привет','мир!'])[3] → N       — не ошибка
(ARRAY[[ 1, 2, 3],
       [10,20,30]])[2][1] → 10

'[-1:1]={10,20,30}'

('[-1:1]={5,6,7}'::int[])[-1]   → 5      — индекс может быть любым
('[-1:1]={5,6,7}'::int[])[-1:0] → {5,6}  — срез массива
(ARRAY[[ 1, 2, 3],
       [10,20,30]])[1:2][2:3] → {{2,3},{20,30}}
```

Размеры

array_ndims — число размерностей

```
\set A '[1:2][-1:1]={{10,20,30},{40,50,60}}'
```

	-1	0	1
1	10	20	30
2	40	50	60

```
array_ndims(:'A'::int[][]) → 2
```

array_length, cardinality — число элементов

```
array_length(:'A'::int[][], 1) → 2
array_length(:'A'::int[][], 2) → 3
array_length(:'A'::int[][])    → 6 = 2*3
```

array_lower, array_upper — индекс первого (последнего) элемента

```
array_lower(:'A'::int[][], 2) → -1
array_upper(:'A'::int[][], 2) → 1
```

Вхождение и поиск

@>, <@ — входит ли один массив в другой (поддержка индексом GIN)

```
ARRAY[1,2,3,4] @> ARRAY[2,3] → t
ARRAY[2,3] <@ ARRAY[1,2,3,4] → t
```

&& — есть ли хотя бы один общий элемент (поддержка индексом GIN)

```
ARRAY[1,3,5] && ARRAY[3,6,9] → t
```

array_position, array_positions — позиция (позиции) элемента

```
array_position (ARRAY[7,8,9,8,7], 8)    → 2
array_position (ARRAY[7,8,9,8,7], 8, 3) → 4
array_positions(ARRAY[7,8,9,8,7], 8)    → {2,4}
array_positions(ARRAY[null,null], null) → {1,2}  — семантика is not distinct from
```

Изменение

array_remove — удаление элемента

```
array_remove(ARRAY[7,8,9,8,7], 8)       → {7,9,7}
```

array_replace — замена элемента

```
array_replace(ARRAY[7,8,9,8,7], 8, 0)   → {7,0,9,0,7}
```

Конструирование

array_fill — инициализация массива значением

```
array_fill(0, ARRAY[2,3])              → {{0,0,0},{0,0,0}}
array_fill(0, ARRAY[2,3], ARRAY[1,-1]) → [1:2][-1:1]={{0,0,0},{0,0,0}}
```

||, array_append, array_prepend, array_cat — конкатенация

```
ARRAY[1,2,3] || 4        → array_append(ARRAY[1,2,3], 4)     → {1,2,3,4}
1 || ARRAY[2,3,4]        → array_prepend(1, ARRAY[2,3,4])    → {1,2,3,4}
ARRAY[1,2] || ARRAY[3,4] → array_cat(ARRAY[1,2], ARRAY[3,4]) → {1,2,3,4}
```

Преобразование типов

array_to_string — конкатенация элементов массива

```
array_to_string(ARRAY[1,2,3], ', ')            → 1, 2, 3
array_to_string(ARRAY[1,2,null,4], ', ', 'N/A) → 1, 2, N/A, 4
```

string_to_array — разбиение строки в массив по разделителю

```
string_to_array('раз два три', ' ')      → {раз,два,три}
string_to_array('Привет', null)          → {П,р,и,в,е,т}
string_to_array('1;2;N/A;4', ';', 'N/A') → {1,2,NULL,4}
```

array_agg — таблица к массиву

```
select array_agg(a) from (values (1),(2)) t(a)                   → {1,2}
select array_agg(a) from (values (ARRAY[1,2]),(ARRAY[3,4])) t(a) → {{1,2},{3,4}}
```

unnest — массив к таблице

```
unnest(ARRAY[1,2]) → 1
                     2 — 2 строки
```