**9.7. Поиск по шаблону**

[**9.7.1.**LIKE](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#functions-like)

[**9.7.2. Регулярные выражения**SIMILAR TO](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#functions-similarto-regexp)

[**9.7.3. Регулярные выражения POSIX**](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#functions-posix-regexp)

PostgreSQL предлагает три разных способа поиска текста по шаблону: традиционный оператор LIKE языка SQL, более современный SIMILAR TO (добавленный в SQL:1999) и регулярные выражения в стиле POSIX. Помимо простых операторов, отвечающих на вопрос «соответствует ли строка этому шаблону?», в PostgreSQL есть функции для извлечения или замены соответствующих подстрок и для разделения строки по заданному шаблону.

**Подсказка**

Если этих встроенных возможностей оказывается недостаточно, вы можете написать собственные функции на языке Perl или Tcl.

**Внимание**

Хотя чаще всего поиск по регулярному выражению бывает очень быстрым, регулярные выражения бывают и настолько сложными, что их обработка может занять приличное время и объём памяти. Поэтому опасайтесь шаблонов регулярных выражений, поступающих из недоверенных источников. Если у вас нет другого выхода, рекомендуется ввести тайм-аут для операторов.

Поиск с шаблонами SIMILAR TO несёт те же риски безопасности, так как конструкция SIMILAR TO предоставляет во многом те же возможности, что и регулярные выражения в стиле POSIX.

Поиск с LIKE гораздо проще, чем два другие варианта, поэтому его безопаснее использовать с недоверенными источниками шаблонов поиска.

**9.7.1.**LIKE

***строка*** LIKE ***шаблон*** [ESCAPE ***спецсимвол***]

***строка*** NOT LIKE ***шаблон*** [ESCAPE ***спецсимвол***]

Выражение LIKE возвращает true, если ***строка*** соответствует заданному ***шаблону***. (Как можно было ожидать, выражение NOT LIKE возвращает false, когда LIKE возвращает true, и наоборот. Этому выражению равносильно выражение NOT (***строка*** LIKE ***шаблон***).)

Если ***шаблон*** не содержит знаков процента и подчёркиваний, тогда шаблон представляет в точности строку и LIKE работает как оператор сравнения. Подчёркивание (\_) в ***шаблоне*** подменяет (вместо него подходит) любой символ; а знак процента (%) подменяет любую (в том числе и пустую) последовательность символов.

Несколько примеров:

'abc' LIKE 'abc' *true*

'abc' LIKE 'a%' *true*

'abc' LIKE '\_b\_' *true*

'abc' LIKE 'c' *false*

При проверке по шаблону LIKE всегда рассматривается вся строка. Поэтому, если нужно найти последовательность символов где-то в середине строки, шаблон должен начинаться и заканчиваться знаками процента.

Чтобы найти в строке буквальное вхождение знака процента или подчёркивания, перед соответствующим символом в ***шаблоне*** нужно добавить спецсимвол. По умолчанию в качестве спецсимвола выбрана обратная косая черта, но с помощью предложения ESCAPE можно выбрать и другой. Чтобы включить спецсимвол в шаблон поиска, продублируйте его.

**Примечание**

Если параметр [standard\_conforming\_strings](https://postgrespro.ru/docs/postgresql/9.6/runtime-config-compatible#guc-standard-conforming-strings) выключен, каждый символ обратной косой черты, записываемый в текстовой константе, нужно дублировать. Подробнее это описано в [Подразделе 4.1.2.1](https://postgrespro.ru/docs/postgresql/9.6/sql-syntax-lexical#sql-syntax-strings).

Также можно отказаться от спецсимвола, написав ESCAPE ''. При этом механизм спецпоследовательностей фактически отключается и использовать знаки процента и подчёркивания буквально в шаблоне нельзя.

Вместо LIKE можно использовать ключевое слово ILIKE, чтобы поиск был регистр-независимым с учётом текущей языковой среды. Этот оператор не описан в стандарте SQL; это расширение PostgreSQL.

Кроме того, в PostgreSQL есть оператор ~~, равнозначный LIKE, и ~~\*, соответствующий ILIKE. Есть также два оператора !~~ и !~~\*, представляющие NOT LIKE и NOT ILIKE, соответственно. Все эти операторы относятся к особенностям PostgreSQL. Вы можете увидеть их, например, в выводе команды EXPLAIN, так как при разборе запроса проверка LIKE и подобные заменяются ими.

Фразы LIKE, ILIKE, NOT LIKE и NOT ILIKE в синтаксисе PostgreSQL обычно обрабатываются как операторы; например, их можно использовать в конструкциях ***выражение*** ***оператор*** ANY (***подвыражение***), хотя предложение ESCAPE здесь добавить нельзя. В некоторых особых случаях всё же может потребоваться использовать вместо них нижележащие операторы.

**9.7.2. Регулярные выражения**SIMILAR TO

***строка*** SIMILAR TO ***шаблон*** [ESCAPE ***спецсимвол***]

***строка*** NOT SIMILAR TO ***шаблон*** [ESCAPE ***спецсимвол***]

Оператор SIMILAR TO возвращает true или false в зависимости от того, соответствует ли данная строка шаблону или нет. Он работает подобно оператору LIKE, только его шаблоны соответствуют определению регулярных выражений в стандарте SQL. Регулярные выражения SQL представляют собой любопытный гибрид синтаксиса LIKE с синтаксисом обычных регулярных выражений.

Как и LIKE, условие SIMILAR TO истинно, только если шаблон соответствует всей строке; это отличается от условий с регулярными выражениями, в которых шаблон может соответствовать любой части строки. Также подобно LIKE, SIMILAR TO воспринимает символы \_ и % как знаки подстановки, подменяющие любой один символ или любую подстроку, соответственно (в регулярных выражениях POSIX им аналогичны символы . и .\*).

Помимо средств описания шаблонов, позаимствованных от LIKE, SIMILAR TO поддерживает следующие метасимволы, унаследованные от регулярных выражений POSIX:

* | означает выбор (одного из двух вариантов).
* \* означает повторение предыдущего элемента 0 и более раз.
* + означает повторение предыдущего элемента 1 и более раз.
* ? означает вхождение предыдущего элемента 0 или 1 раз.
* {***m***} означает повторяет предыдущего элемента ровно ***m*** раз.
* {***m***,} означает повторение предыдущего элемента ***m*** или более раз.
* {***m***,***n***} означает повторение предыдущего элемента не менее чем ***m*** и не более чем ***n*** раз.
* Скобки () объединяют несколько элементов в одну логическую группу.
* Квадратные скобки [...] обозначают класс символов так же, как и в регулярных выражениях POSIX.

Обратите внимание, точка (.) не является метасимволом для оператора SIMILAR TO.

Как и с LIKE, обратная косая черта отменяет специальное значение любого из этих метасимволов, а предложение ESCAPE позволяет выбрать другой спецсимвол.

Несколько примеров:

'abc' SIMILAR TO 'abc' *true*

'abc' SIMILAR TO 'a' *false*

'abc' SIMILAR TO '%(b|d)%' *true*

'abc' SIMILAR TO '(b|c)%' *false*

Функция substring с тремя параметрами, substring(***строка*** from ***шаблон*** for ***спецсимвол***) извлекает подстроку, соответствующую шаблону регулярного выражения SQL. Как и с SIMILAR TO, указанному шаблону должна соответствовать вся строка; в противном случае функция не найдёт ничего и вернёт NULL. Для обозначения части шаблона, которая должна быть возвращена в случае успеха, шаблон должен содержать два спецсимвола и кавычки (") после каждого. Эта функция возвращает часть шаблона между двумя такими маркерами.

Несколько примеров с маркерами #", выделяющими возвращаемую строку:

substring('foobar' from '%#"o\_b#"%' for '#') *oob*

substring('foobar' from '#"o\_b#"%' for '#') *NULL*

**9.7.3. Регулярные выражения POSIX**

В [Таблице 9.14](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#functions-posix-table) перечислены все существующие операторы для проверки строк регулярными выражениями POSIX.

**Таблица 9.14. Операторы регулярных выражений**

| **Оператор** | **Описание** | **Пример** |
| --- | --- | --- |
| ~ | Проверяет соответствие регулярному выражению с учётом регистра | 'thomas' ~ '.\*thomas.\*' |
| ~\* | Проверяет соответствие регулярному выражению без учёта регистра | 'thomas' ~\* '.\*Thomas.\*' |
| !~ | Проверяет несоответствие регулярному выражению с учётом регистра | 'thomas' !~ '.\*Thomas.\*' |
| !~\* | Проверяет несоответствие регулярному выражению без учёта регистра | 'thomas' !~\* '.\*vadim.\*' |

Регулярные выражения POSIX предоставляют более мощные средства поиска по шаблонам, чем операторы LIKE и SIMILAR TO. Во многих командах Unix, таких как egrep, sed и awk используется язык шаблонов, похожий на описанный здесь.

Регулярное выражение — это последовательность символов, представляющая собой краткое определение набора строк (*регулярное множество*). Строка считается соответствующей регулярному выражению, если она является членом регулярного множества, описываемого регулярным выражением. Как и для LIKE, символы шаблона непосредственно соответствуют символам строки, за исключением специальных символов языка регулярных выражений. При этом спецсимволы регулярных выражений отличается от спецсимволов LIKE. В отличие от шаблонов LIKE, регулярное выражение может совпадать с любой частью строки, если только оно не привязано явно к началу и/или концу строки.

Несколько примеров:

'abc' ~ 'abc' *true*

'abc' ~ '^a' *true*

'abc' ~ '(b|d)' *true*

'abc' ~ '^(b|c)' *false*

Более подробно язык шаблонов в стиле POSIX описан ниже.

Функция substring с двумя параметрами, substring(***строка*** from ***шаблон***), извлекает подстроку, соответствующую шаблону регулярного выражения POSIX. Она возвращает фрагмент текста, подходящий шаблону, если таковой находится в строке, либо NULL в противном случае. Но если шаблон содержит скобки, она возвращает первое подвыражение, заключённое в скобки (то, которое начинается с самой первой открывающей скобки). Если вы хотите использовать скобки, но не в таком особом режиме, можно просто заключить в них всё выражение. Если же вам нужно включить скобки в шаблон до подвыражения, которое вы хотите извлечь, это можно сделать, используя группы без захвата, которые будут описаны ниже.

Несколько примеров:

substring('foobar' from 'o.b') *oob*

substring('foobar' from 'o(.)b') *o*

Функция regexp\_replace подставляет другой текст вместо подстрок, соответствующих шаблонам регулярных выражений POSIX. Она имеет синтаксис regexp\_replace(***исходная\_строка***, ***шаблон***, ***замена*** [, ***флаги***]). Если ***исходная\_строка*** не содержит фрагмента, подходящего под ***шаблон***, она возвращается неизменной. Если же соответствие находится, возвращается ***исходная\_строка***, в которой вместо соответствующего фрагмента подставляется ***замена***. Строка ***замена*** может содержать \***n***, где ***n*** — число от 1 до 9, указывающее на исходный фрагмент, соответствующий ***n***-ому подвыражению в скобках, и может содержать обозначение \&, указывающее, что будет вставлен фрагмент, соответствующий всему шаблону. Если же в текст замены нужно включить обратную косую черту буквально, следует написать \\. В необязательном параметре ***флаги*** передаётся текстовая строка, содержащая ноль или более однобуквенных флагов, меняющих поведение функции. Флаг i включает поиск без учёта регистра, а флаг g указывает, что заменяться должны все подходящие подстроки, а не только первая из них. Допустимые флаги (кроме g) описаны в [Таблице 9.22](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-embedded-options-table).

Несколько примеров:

regexp\_replace('foobarbaz', 'b..', 'X')

*fooXbaz*

regexp\_replace('foobarbaz', 'b..', 'X', 'g')

*fooXX*

regexp\_replace('foobarbaz', 'b(..)', 'X\1Y', 'g')

*fooXarYXazY*

Функция regexp\_matches возвращает текстовый массив из всех подходящих подстрок, полученных в результате применения регулярного выражения POSIX. Она имеет синтаксис regexp\_matches(***строка***, ***шаблон*** [, ***флаги***]). Эта функция может вовсе не вернуть строк или вернуть одну или несколько строк (см. описание флага g ниже). Если ***шаблон*** не находится в строке, функция не возвращает строк. Если шаблон не содержит подвыражений в скобках, тогда для каждой строки возвращается массив с одним элементом, содержащим подстроку, соответствующую всему шаблону. Если же шаблон содержит подвыражения в скобках, функция возвращает массив элементов, в котором ***n***-й элемент соответствует ***n***-му подвыражению в скобках (не считая скобки «без захвата»; подробнее об этом ниже). В необязательном параметре ***флаги*** передаётся текстовая строка, содержащая ноль или более однобуквенных флагов, меняющих поведение функции. Флаг g указывает, что функция должна находить в строке не только первое, а все соответствия шаблону, и возвращать строку для каждого соответствия. Допустимые флаги (кроме g) описаны в [Таблице 9.22](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-embedded-options-table).

Несколько примеров:

SELECT regexp\_matches('foobarbequebaz', '(bar)(beque)');

regexp\_matches

----------------

{bar,beque}

(1 row)

SELECT regexp\_matches(

'foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');

regexp\_matches

----------------

{bar,beque}

{bazil,barf}

(2 rows)

SELECT regexp\_matches('foobarbequebaz', 'barbeque');

regexp\_matches

----------------

{barbeque}

(1 row)

Добиться, чтобы функция regexp\_matches() всегда возвращала одну строку, можно с помощью вложенного подзапроса; это особенно полезно в списке выборки SELECT, когда нужно получить все строки, в том числе и не соответствующие шаблону:

SELECT col1, (SELECT regexp\_matches(col2, '(bar)(beque)')) FROM tab;

Функция regexp\_split\_to\_table разделяет строку, используя в качестве разделителя шаблон регулярного выражения POSIX. Она имеет синтаксис regexp\_split\_to\_table(***строка***, ***шаблон*** [, ***флаги***]). Если ***шаблон*** не находится в переданной строке, возвращается вся ***строка*** целиком. Если находится минимум одно вхождение, для каждого такого вхождения возвращается текст от конца предыдущего вхождения (или начала строки) до начала вхождения. После последнего найденного вхождения возвращается фрагмент от его конца до конца строки. В необязательном параметре ***флаги*** передаётся текстовая строка, содержащая ноль или более однобуквенных флагов, меняющих поведение функции. Флаги, которые поддерживает regexp\_split\_to\_table, описаны в [Таблице 9.22](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-embedded-options-table).

Функция regexp\_split\_to\_array ведёт себя подобно regexp\_split\_to\_table, за исключением того, что regexp\_split\_to\_array возвращает результат в массиве элементов типа text. Она имеет синтаксис regexp\_split\_to\_array(***строка***, ***шаблон*** [, ***флаги***]). Параметры у этой функции те же, что и у regexp\_split\_to\_table.

Несколько примеров:

SELECT foo FROM regexp\_split\_to\_table('the quick brown fox jumps over the lazy dog', '\s+') AS foo;

foo

-------

the

quick

brown

fox

jumps

over

the

lazy

dog

(9 rows)

SELECT regexp\_split\_to\_array('the quick brown fox jumps over the lazy dog', '\s+');

regexp\_split\_to\_array

-----------------------------------------------

{the,quick,brown,fox,jumps,over,the,lazy,dog}

(1 row)

SELECT foo FROM regexp\_split\_to\_table('the quick brown fox', '\s\*') AS foo;

foo

-----

t

h

e

q

u

i

c

k

b

r

o

w

n

f

o

x

(16 rows)

Как показывает последний пример, функции разделения по регулярным выражениям игнорируют вхождения нулевой длины, идущие в начале и в конце строки, а также непосредственно за предыдущим вхождением. Это поведение противоречит строгому определению поиска по регулярным выражениям, который реализует regexp\_matches, но обычно более удобно на практике. Подобное поведение наблюдается и в других программных средах, например в Perl.

9.7.3.1. Подробное описание регулярных выражений

Регулярные выражения в PostgreSQL реализованы с использованием программного пакета, который разработал Генри Спенсер (Henry Spencer). Практически всё следующее описание регулярных выражений дословно скопировано из его руководства.

Регулярное выражение (Regular expression, RE), согласно определению в POSIX 1003.2, может иметь две формы: *расширенное* RE или ERE (грубо говоря, это выражения, которые понимает egrep) и *простое* RE или BRE (грубо говоря, это выражения для ed). PostgreSQL поддерживает обе формы, а кроме того реализует некоторые расширения, не предусмотренные стандартом POSIX, но широко используемые вследствие их доступности в некоторых языках программирования, например в Perl и Tcl. Регулярные выражения, использующие эти несовместимые с POSIX расширения, здесь называются *усовершенствованными* RE или ARE. ARE практически представляют собой надмножество ERE, тогда как BRE отличаются некоторой несовместимостью в записи (помимо того, что они гораздо более ограничены). Сначала мы опишем формы ARE и ERE, отметив особенности, присущие только ARE, а затем расскажем, чем от них отличаются BRE.

**Примечание**

PostgreSQL изначально всегда предполагает, что регулярное выражение следует правилам ARE. Однако можно переключиться на более ограниченные правила ERE или BRE, добавив в шаблон RE *встроенный параметр*, как описано в [Подразделе 9.7.3.4](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-metasyntax). Это может быть полезно для совместимости с приложениями, ожидающими от СУБД строгого следования правилам POSIX 1003.2.

Регулярное выражение определяется как одна или более *ветвей*, разделённых символами |. Оно считается соответствующим всему, что соответствует одной из этих ветвей.

Ветвь — это ноль или несколько *количественных атомов* или *ограничений*, соединённых вместе. Соответствие ветви в целом образуется из соответствия первой части, за которым следует соответствие второй части и т. д.; пустой ветви соответствует пустая строка.

Количественный атом — это *атом*, за которым может следовать *определитель количества*. Без этого определителя ему соответствует одно вхождение атома. С определителем количества ему может соответствовать некоторое число вхождений этого атома. Все возможные *атомы* перечислены в [Таблице 9.15](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-atoms-table). Варианты определителей количества и их значения перечислены в [Таблице 9.16](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-quantifiers-table).

*Ограничению* соответствует пустая строка, но это соответствие возможно только при выполнении определённых условий. Ограничения могут использоваться там же, где и атомы, за исключением того, что их нельзя дополнять определителями количества. Простые ограничения показаны в [Таблице 9.17](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-constraints-table); некоторые дополнительные ограничения описаны ниже.

**Таблица 9.15. Атомы регулярных выражений**

| **Атом** | **Описание** |
| --- | --- |
| (***re***) | (где ***re*** — любое регулярное выражение) описывает соответствие ***re***, при этом данное соответствие захватывается для последующей обработки |
| (?:***re***) | подобно предыдущему, но соответствие не захватывается (т. е. это набор скобок «без захвата») (применимо только к ARE) |
| . | соответствует любому символу |
| [***символы***] | *выражение в квадратных скобках*, соответствует любому из ***символов*** (подробнее это описано в [Подразделе 9.7.3.2](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-bracket-expressions)) |
| \***k*** | (где ***k*** — не алфавитно-цифровой символ) соответствует обычному символу буквально, т. е. \\ соответствует обратной косой черте |
| \***c*** | где ***c*** — алфавитно-цифровой символ (за которым могут следовать другие символы), это *спецсимвол*, см. [Подраздел 9.7.3.3](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-escape-sequences) (применим только к ARE; в ERE и BRE этому атому соответствует ***c***) |
| { | когда за этим символом следует любой символ, кроме цифры, этот атом соответствует левой фигурной скобке ({), если же за ним следует цифра, это обозначает начало ***границы*** (см. ниже) |
| ***x*** | (где ***x*** — один символ, не имеющий специального значения) соответствует этому символу |

Выражение RE не может заканчиваться обратной косой чертой (\).

**Примечание**

Если параметр [standard\_conforming\_strings](https://postgrespro.ru/docs/postgresql/9.6/runtime-config-compatible#guc-standard-conforming-strings) выключен, каждый символ обратной косой черты, записываемый в текстовой константе, нужно дублировать. Подробнее это описано в [Подразделе 4.1.2.1](https://postgrespro.ru/docs/postgresql/9.6/sql-syntax-lexical#sql-syntax-strings).

**Таблица 9.16. Определители количества в регулярных выражениях**

| **Определитель** | **Соответствует** |
| --- | --- |
| \* | 0 или более вхождений атома |
| + | 1 или более вхождений атома |
| ? | 0 или 1 вхождение атома |
| {***m***} | ровно ***m*** вхождений атома |
| {***m***,} | ***m*** или более вхождений атома |
| {***m***,***n***} | от ***m*** до ***n*** (включая границы) вхождений атома; ***m*** не может быть больше ***n*** |
| \*? | не жадная версия \* |
| +? | не жадная версия + |
| ?? | не жадная версия ? |
| {***m***}? | не жадная версия {***m***} |
| {***m***,}? | не жадная версия {***m***,} |
| {***m***,***n***}? | не жадная версия {***m***,***n***} |

В формах с {***...***} числа ***m*** и ***n*** определяют так называемые *границы* количества. Эти числа должны быть беззнаковыми десятичными целыми в диапазоне от 0 до 255 включительно.

*Не жадные* определители (допустимые только в ARE) описывают те же возможные соответствия, что и аналогичные им обычные («*жадные*»), но предпочитают выбирать наименьшее, а не наибольшее количество вхождений. Подробнее это описано в [Подразделе 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules).

**Примечание**

Определители количества не могут следовать один за другим, например запись \*\* будет ошибочной. Кроме того, определители не могут стоять в начале выражения или подвыражения и идти сразу после ^ или |.

**Таблица 9.17. Ограничения в регулярных выражениях**

| **Ограничение** | **Описание** |
| --- | --- |
| ^ | соответствует началу строки |
| $ | соответствует концу строки |
| (?=***re***) | *позитивный просмотр вперёд* находит соответствие там, где начинается подстрока, соответствующая ***re*** (только для ARE) |
| (?!***re***) | *негативный просмотр вперёд* находит соответствие там, где не начинается подстрока, соответствующая ***re*** (только для ARE) |
| (?<=***re***) | *позитивный просмотр назад* находит соответствие там, где заканчивается подстрока, соответствующая ***re*** (только для ARE) |
| (?<!***re***) | *негативный просмотр назад* находит соответствие там, где не заканчивается подстрока, соответствующая ***re*** (только для ARE) |

Ограничения просмотра вперёд и назад не могут содержать *ссылки назад* (см. [Подраздел 9.7.3.3](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-escape-sequences)), и все скобки в них считаются «скобками без захвата».

9.7.3.2. Выражения в квадратных скобках

*Выражение в квадратных скобках* содержит список символов, заключённый в []. Обычно ему соответствует любой символ из списка (об исключении написано ниже). Если список начинается с ^, ему соответствует любой символ, который ***не*** перечисляется далее в этом списке. Если два символа в списке разделяются знаком -, это воспринимается как краткая запись полного интервала символов между двумя заданными (и включая их) в порядке сортировки; например выражению [0-9] в ASCII соответствует любая десятичная цифра. Два интервала не могут разделять одну границу, т. е. выражение a-c-e недопустимо. Интервалы зависят от порядка сортировки, который может меняться, поэтому в переносимых программах их лучше не использовать.

Чтобы включить в список ], этот символ нужно написать первым (сразу за ^, если он присутствует). Чтобы включить в список символ -, его нужно написать первым или последним, либо как вторую границу интервала. Указать - в качестве первой границы интервал можно, заключив его между [. и .], чтобы он стал элементом сортировки (см. ниже). За исключением этих символов, некоторых комбинаций с [ (см. следующие абзацы) и спецсимволов (в ARE), все остальные специальные символы в квадратных скобках теряют своё особое значение. В частности, символ \ по правилам ERE или BRE воспринимается как обычный, хотя в ARE он экранирует символ, следующий за ним.

Выражения в квадратных скобках могут содержать элемент сортировки (символ или последовательность символов или имя такой последовательности), определение которого заключается между [. и .]. Определяющая его последовательность воспринимается в выражении в скобках как один элемент. Это позволяет включать в такие выражения элементы, соответствующие последовательности нескольких символов. Например, с элементом сортировки ch в квадратных скобках регулярному выражению [[.ch.]]\*c будут соответствовать первые пять символов строки chchcc.

**Примечание**

В настоящее время PostgreSQL не поддерживает элементы сортировки, состоящие из нескольких символов. Эта информация относится к возможному в будущем поведению.

В квадратных скобках могут содержаться элементы сортировки, заключённые между [= и =], обозначающие *классы эквивалентности*, т. е. последовательности символов из всех элементов сортировки, эквивалентных указанному, включая его самого. (Если для этого символа нет эквивалентных, он обрабатывается как заключённый между [. и .].) Например, если е и ё — члены одного класса эквивалентности, выражения [[=е=]], [[=ё=]] и [её] будут равнозначными. Класс эквивалентности нельзя указать в качестве границы интервала.

В квадратных скобках может также содержаться имя класса символов, заключённое между [: и :], и заменяющее список всех символов этого класса. Стандартные имена классов: alnum, alpha, blank, cntrl, digit, graph, lower, print, punct, space, upper и xdigit. Весь этот набор классов определён в ctype и он может меняться в зависимости от локали (языковой среды). Класс символов также нельзя использовать в качестве границы интервала.

Есть два особых вида выражений в квадратных скобках: выражения [[:<:]] и [[:>:]], представляющие собой ограничения, соответствующие пустым строкам в начале и конце слова. Слово в данном контексте определяется как последовательность словосоставляющих символов, перед или после которой нет словосоставляющих символов. Словосоставляющий символ — это символ класса alnum (определённого в ctype) или подчёркивание. Это расширение совместимо со стандартом POSIX 1003.2, но не описано в нём, и поэтому его следует использовать с осторожностью там, где важна совместимость с другими системами. Обычно лучше использовать ограничивающие спецсимволы, описанные ниже; они также не совсем стандартны, но набрать их легче.

9.7.3.3. Спецсимволы регулярных выражений

*Спецсимволы* — это специальные команды, состоящие из \ и последующего алфавитно-цифрового символа. Можно выделить следующие категории спецсимволов: обозначения символов, коды классов, ограничения и ссылки назад. Символ \, за которым идёт алфавитно-цифровой символ, не образующий допустимый спецсимвол, считается ошибочным в ARE. В ERE спецсимволов нет: вне квадратных скобок пара из \ и последующего алфавитно-цифрового символа, воспринимается просто как данный символ, а в квадратных скобках и сам символ \ воспринимается просто как обратная косая черта. (Последнее на самом деле нарушает совместимость между ERE и ARE.)

*Спецобозначения символов* введены для того, чтобы облегчить ввод в RE непечатаемых и других неудобных символов. Они приведены в [Таблице 9.18](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-character-entry-escapes-table).

*Коды классов* представляют собой краткий способ записи имён некоторых распространённых классов символов. Они перечислены в [Таблице 9.19](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-class-shorthand-escapes-table).

*Спецсимволы ограничений* обозначают ограничения, которым при совпадении определённых условий соответствует пустая строка. Они перечислены в [Таблице 9.20](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-constraint-escapes-table).

*Ссылка назад* (\***n***) соответствует той же строке, какой соответствовало предыдущее подвыражение в скобках под номером ***n*** (см. [Таблицу 9.21](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-constraint-backref-table)). Например, ([bc])\1 соответствует bb или cc, но не bc или cb. Это подвыражение должно полностью предшествовать ссылке назад в RE. Нумеруются подвыражения в порядке следования их открывающих скобок. При этом скобки без захвата исключаются из рассмотрения.

**Таблица 9.18. Спецобозначения символов в регулярных выражениях**

| **Спецсимвол** | **Описание** |
| --- | --- |
| \a | символ звонка, как в C |
| \b | символ «забой», как в C |
| \B | синоним для обратной косой черты (\), сокращающий потребность в дублировании этого символа |
| \c***X*** | (где ***X*** — любой символ) символ, младшие 5 бит которого те же, что и у ***X***, а остальные равны 0 |
| \e | символ, определённый в последовательности сортировки с именем ESC, либо, если таковой не определён, символ с восьмеричным значением 033 |
| \f | подача формы, как в C |
| \n | новая строка, как в C |
| \r | возврат каретки, как в C |
| \t | горизонтальная табуляция, как в C |
| \u***wxyz*** | (где ***wxyz*** ровно четыре шестнадцатеричные цифры) символ с шестнадцатеричным кодом 0x***wxyz*** |
| \U***stuvwxyz*** | (где ***stuvwxyz*** ровно восемь шестнадцатеричных цифр) символ с шестнадцатеричным кодом 0x***stuvwxyz*** |
| \v | вертикальная табуляция, как в C |
| \x***hhh*** | (где ***hhh*** — несколько шестнадцатеричных цифр) символ с шестнадцатеричным кодом 0x***hhh*** (символ всегда один вне зависимости от числа шестнадцатеричных цифр) |
| \0 | символ с кодом 0 (нулевой байт) |
| \***xy*** | (где ***xy*** — ровно две восьмеричных цифры, не *ссылка назад*) символ с восьмеричным кодом 0***xy*** |
| \***xyz*** | (где ***xyz*** — ровно три восьмеричных цифры, не *ссылка назад*) символ с восьмеричным кодом 0***xyz*** |

Шестнадцатеричные цифры записываются символами 0-9 и a-f или A-F. Восьмеричные цифры — цифры от 0 до 7.

Спецпоследовательности с числовыми кодами, задающими значения вне диапазона ASCII (0-127), воспринимаются по-разному в зависимости от кодировки базы данных. Когда база данных имеет кодировку UTF-8, спецкод равнозначен позиции символа в Unicode, например, \u1234 обозначает символ U+1234. Для других многобайтных кодировок спецпоследовательности обычно просто задают серию байт, определяющих символ. Если в кодировке базы данных отсутствует символ, заданный спецпоследовательностью, ошибки не будет, но и никакие данные не будут ей соответствовать.

Символы, переданные спецобозначением, всегда воспринимаются как обычные символы. Например, \135 кодирует ] в ASCII, но спецпоследовательность \135 не будет закрывать выражение в квадратных скобках.

**Таблица 9.19. Спецкоды классов в регулярных выражениях**

| **Спецсимвол** | **Описание** |
| --- | --- |
| \d | [[:digit:]] |
| \s | [[:space:]] |
| \w | [[:alnum:]\_] (подчёркивание также включается) |
| \D | [^[:digit:]] |
| \S | [^[:space:]] |
| \W | [^[:alnum:]\_] (подчёркивание также включается) |

В выражениях в квадратных скобках спецсимволы \d, \s и \w теряют свои внешние квадратные скобки, а \D, \S и \W — недопустимы. (Так что, например запись [a-c\d] равнозначна [a-c[:digit:]]. А запись [a-c\D], которая была бы равнозначна [a-c^[:digit:]], — недопустима.)

**Таблица 9.20. Спецсимволы ограничений в регулярных выражений**

| **Спецсимвол** | **Описание** |
| --- | --- |
| \A | соответствует только началу строки (чем это отличается от ^, описано в [Подразделе 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules)) |
| \m | соответствует только началу слова |
| \M | соответствует только концу слова |
| \y | соответствует только началу или концу слова |
| \Y | соответствует только положению не в начале и не в конце слова |
| \Z | соответствует только концу строки (чем это отличается от $, описано в [Подразделе 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules)) |

Определением слова здесь служит то же, что было приведено выше в описании [[:<:]] и [[:>:]]. В квадратных скобках спецсимволы ограничений не допускаются.

**Таблица 9.21. Ссылки назад в регулярных выражениях**

| **Спецсимвол** | **Описание** |
| --- | --- |
| \***m*** | (где ***m*** — цифра, отличная от 0) — ссылка назад на подвыражение под номером ***m*** |
| \***mnn*** | (где ***m*** — цифра, отличная от 0, а ***nn*** — ещё несколько цифр с десятичным значением ***mnn***, не превышающим число закрытых до этого скобок с захватом) ссылка назад на подвыражение под номером ***mnn*** |

**Примечание**

Регулярным выражениям присуща неоднозначность между восьмеричными кодами символов и ссылками назад, которая разрешается следующим образом (это упоминалось выше). Ведущий ноль всегда считается признаком восьмеричной последовательности. Единственная цифра, отличная от 0, за которой не следует ещё одна цифра, всегда воспринимается как ссылка назад. Последовательность из нескольких цифр, которая начинается не с 0, воспринимается как ссылка назад, если она идёт за подходящим подвыражением (т. е. число оказывается в диапазоне, допустимом для ссылки назад), в противном случае она воспринимается как восьмеричное число.

9.7.3.4. Метасинтаксис регулярных выражений

В дополнение к основному синтаксису, описанному выше, можно использовать также несколько особых форм и разнообразные синтаксические удобства.

Регулярное выражение может начинаться с одного из двух специальных префиксов режима. Если RE начинается с \*\*\*:, его продолжение рассматривается как ARE. (В PostgreSQL это обычно не имеет значения, так как регулярные выражения воспринимаются как ARE по умолчанию; но это может быть полезно, когда параметр ***флаги*** функций regex включает режим ERE или BRE.) Если RE начинается с \*\*\*=, его продолжение воспринимается как обычная текстовая строка, все его символы воспринимаются буквально.

ARE может начинаться со *встроенных параметров*: последовательности (?***xyz***) (где ***xyz*** — один или несколько алфавитно-цифровых символов), определяющих параметры остального регулярного выражения. Эти параметры переопределяют любые ранее определённые параметры, в частности они могут переопределить режим чувствительности к регистру, подразумеваемый для оператора regex, или параметр ***флаги*** функции regex. Допустимые буквы параметров показаны в [Таблице 9.22](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-embedded-options-table). Заметьте, что те же буквы используются в параметре ***флаги*** функций regex.

**Таблица 9.22. Буквы встроенных параметров ARE**

| **Параметр** | **Описание** |
| --- | --- |
| b | продолжение регулярного выражения — BRE |
| c | поиск соответствий с учётом регистра (переопределяет тип оператора) |
| e | продолжение RE — ERE |
| i | поиск соответствий без учёта регистра (см. [Подраздел 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules)) (переопределяет тип оператора) |
| m | исторически сложившийся синоним n |
| n | поиск соответствий с учётом перевода строк (см. [Подраздел 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules)) |
| p | переводы строк учитываются частично (см. [Подраздел 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules)) |
| q | продолжение регулярного выражения — обычная строка («в кавычках»), содержимое которой воспринимается буквально |
| s | поиск соответствий без учёта перевода строк (по умолчанию) |
| t | компактный синтаксис (по умолчанию; см. ниже) |
| w | переводы строк учитываются частично, но в другом, «странном» режиме (см. [Подраздел 9.7.3.5](https://postgrespro.ru/docs/postgresql/9.6/functions-matching#posix-matching-rules)) |
| x | развёрнутый синтаксис (см. ниже) |

Внедрённые параметры начинают действовать сразу после скобки ), завершающей их последовательность. Они могут находиться только в начале ARE (после указания \*\*\*:, если оно присутствует).

Помимо обычного (*компактного*) синтаксиса RE, в котором имеют значение все символы, поддерживается также *развёрнутый* синтаксис, включить который можно с помощью встроенного параметра x. В развёрнутом синтаксисе игнорируются пробельные символы, а также все символы от # до конца строки (или конца RE). Это позволяет разделять RE на строки и добавлять в него комментарии. Но есть три исключения:

* пробельный символ или #, за которым следует \, сохраняется
* пробельный символ или # внутри выражения в квадратных скобках сохраняется
* пробельные символы и комментарии не могут присутствовать в составных символах, например, в (?:

В данном контексте пробельными символами считаются пробел, табуляция, перевод строки и любой другой символ, относящийся к классу символов ***space***.

И наконец, в ARE последовательность (?#***ttt***) (где ***ttt*** — любой текст, не содержащий )) вне квадратных скобок также считается комментарием и полностью игнорируется. При этом она так же не может находиться внутри составных символов, таких как (?:. Эти комментарии в большей степени историческое наследие, чем полезное средство; они считаются устаревшими, а вместо них рекомендуется использовать развёрнутый синтаксис.

***Ни одно*** из этих расширений метасинтаксиса не будет работать, если выражение начинается с префикса \*\*\*=, после которого строка воспринимается буквально, а не как RE.

9.7.3.5. Правила соответствия регулярным выражениям

В случае, когда RE может соответствовать более чем одной подстроке в заданной строке, соответствующей RE считается подстрока, которая начинается в ней первой. Если к данной позиции подобных соответствующих подстрок оказывается несколько, из них выбирается либо самая длинная, либо самая короткая из возможных, в зависимости от того, какой режим выбран в RE: *жадный* или *не жадный*.

Где жадный или не жадный характер RE определяется по следующим правилам:

* Большинство атомов и все ограничения не имеют признака жадности (так как они всё равно не могут соответствовать подстрокам разного состава).
* Скобки, окружающие RE, не влияют на его «жадность».
* Атом с определителем фиксированного количества ({***m***} или {***m***}?) имеет ту же характеристику жадности (или может не иметь её), как и сам атом.
* Атом с другими обычными определителями количества (включая {***m***,***n***}, где ***m*** равняется ***n***) считается жадным (предпочитает соответствие максимальной длины).
* Атом с не жадным определителем количества (включая {***m***,***n***}?, где ***m*** равно ***n***) считается не жадным (предпочитает соответствие минимальной длины).
* Ветвь (RE без оператора | на верхнем уровне) имеет ту же характеристику жадности, что и первый количественный атом в нём, имеющий атрибут жадности.
* RE, образованное из двух или более ветвей, соединённых оператором |, всегда считается жадным.

Эти правила связывают характеристики жадности не только с отдельными количественными атомами, но и с ветвями и целыми RE, содержащими количественные атомы. Это означает, что при сопоставлении ветвь или целое RE может соответствовать максимально длинной или короткой подстроке ***в целом***. Когда определена длина всего соответствия, часть его, соответствующая конкретному подвыражению, определяется с учётом характеристики жадности для этого подвыражения, при этом подвыражения, начинающиеся в RE раньше, имеют больший приоритет, чем следующие за ними.

Это иллюстрирует следующий пример:

SELECT SUBSTRING('XY1234Z', 'Y\*([0-9]{1,3})');

*Результат:* 123

SELECT SUBSTRING('XY1234Z', 'Y\*?([0-9]{1,3})');

*Результат:* 1

В первом случае RE в целом жадное, так как жадным является атом Y\*. Соответствие ему начинается с буквы Y и оно включает подстроку максимальной длины с этого места, т. е. подстроку Y123. Результат выражения — её часть, соответствующая подвыражению в скобках, т. е. 123. Во втором случае, RE в целом наследует не жадный характер от атома Y\*?. Соответствие ему так же начинается с Y, но включает оно подстроку минимальной длины с этого места, т. е. Y1. И хотя подвыражение [0-9]{1,3} имеет жадный характер, оно не может повлиять на выбор длины соответствия в целом, поэтому ему остаётся только подстрока 1.

Другими словами, когда RE содержит и жадные, и не жадные подвыражения, всё соответствие будет максимально длинным или коротким в зависимости от характеристики всего RE. Характеристики, связанные с подвыражениями, влияют только на то, какую часть подстроки может «поглотить» одно подвыражение относительно другого.

Чтобы явно придать характеристику «жадности» или «нежадности» подвыражению или всему RE, можно использовать определители {1,1} и {1,1}?, соответственно. Это полезно, когда вам нужно, чтобы общая характеристика жадности RE отличалась от той, что вытекает из его элементов. Например, предположим, что вы пытаетесь выделить из строки, содержащей несколько цифр, эти цифры и части до и после них. Можно попытаться сделать это так:

SELECT regexp\_matches('abc01234xyz', '(.\*)(\d+)(.\*)');

*Результат:* {abc0123,4,xyz}

Но это не будет работать: первая группа .\* — «жадная», она «съест» всё, что сможет, оставляя для соответствия \d+ только последнюю возможность, то есть последнюю цифру. Можно попытаться сделать запрос «нежадным»:

SELECT regexp\_matches('abc01234xyz', '(.\*?)(\d+)(.\*)');

*Результат:* {abc,0,""}

И это не будет работать, так теперь весь RE в целом стал нежадным, и все соответствия завершаются как можно раньше. Но мы можем получить нужный результат, явно сделав жадным всё RE:

SELECT regexp\_matches('abc01234xyz', '(?:(.\*?)(\d+)(.\*)){1,1}');

*Результат:* {abc,01234,xyz}

Управление общей характеристикой «жадности» RE независимо от «жадности» его компонентов даёт большую гибкость в описании шаблонов переменной длины.

При определении более длинного или более короткого соответствия длины соответствий определяются в символах, а не в элементах сортировки. Пустая строка считается длиннее, чем отсутствие соответствия. Например, выражению bb\* соответствуют три символа в середине строки abbbc, выражению (week|wee)(night|knights) — все десять символов weeknights; когда выражение (.\*).\* сопоставляется со строкой abc, подвыражению в скобках соответствуют все три символа; а когда (a\*)\* сопоставляется со строкой bc, то и RE в целом, и подстроке в скобках соответствует пустая строка.

Игнорирование регистра символов даёт практически тот же эффект, как если бы в алфавите исчезли различия прописных и строчных букв. Если буква, существующая и в верхнем, и в нижнем регистре, фигурирует вне квадратных скобок как обычный символ, она по сути преобразуется в выражение в квадратных скобках, содержащее оба варианта, например x становится [xX]. Если же она фигурирует в выражении в квадратных скобках, в это выражение добавляются все её варианты, например [x] становится [xX], а [^x] — [^xX].

Когда включён режим учёта перевода строк, атом . и выражения в квадратных скобках с ^ никогда не будут соответствовать символам конца строки (так что соответствия никогда не будут пересекать границы строк, если в RE нет явных указаний на эти символы), а ^ и $ будут соответствовать пустой подстроке не только в начале и конце всего текста, но и в начале и конце каждой отдельной его строки. Однако спецсимволы ARE \A и \Z по-прежнему будут соответствовать ***только*** началу и концу всего текста.

В режиме, когда переводы строк учитываются частично, особый смысл перевод строк имеет для атома . и выражений в квадратных скобках, но не для ^ и $.

В обратном частичном режиме, перевод строк имеет особый смысл для ^ и $, как и в режиме с учётом перевода строк, но не для . и выражений в квадратных скобках. Данный режим не очень полезен, но существует для симметрии.

9.7.3.6. Пределы и совместимость

В текущей реализации отсутствует какой-либо явно заданный предел длины RE. Однако, разрабатывая программы высокой степени переносимости, не следует применять RE длиннее 256 байт, так как другая POSIX-совместимая реализация может отказаться обрабатывать такие регулярные выражения.

Единственная особенность ARE, действительно несовместимая с ERE стандарта POSIX проявляется в том, что в ARE знак \ не теряет своё специальное значение в квадратных скобках. Все другие расширения ARE используют синтаксические возможности, которые не определены, не допустимы или не поддерживаются в ERE; синтаксис переключения режимов (\*\*\*) также выходит за рамки синтаксиса POSIX как для BRE, так и для ERE.

Многие расширения ARE заимствованы из языка Perl, но некоторые были изменены, оптимизированы, а несколько расширений Perl были исключены. В результате имеют место следующие несовместимости: атомы \b и \B, отсутствие специальной обработки завершающего перевода строки, добавление исключений в квадратных скобках в число случаев, когда учитывается перевод строк, особые условия для скобок и ссылок назад в ограничениях просмотра вперёд/назад и семантика «наиболее длинное/короткое соответствие» (вместо «первое соответствие»).

Важно отметить две несовместимости синтаксиса ARE и регулярных выражений ERE, которые воспринимал PostgreSQL до версии 7.4:

* В ARE \ с последующим алфавитно-цифровым символом представляет либо спецсимвол, либо ошибочную последовательность, тогда как в предыдущих версиях так можно было записывать алфавитно-цифровые символы. Это не должно быть большой проблемой, так как раньше не было причин использовать такие последовательности.
* В ARE знак \ сохраняет своё специальное значение в [], поэтому, чтобы передать \ в квадратных скобках буквально, его нужно записать как \\.

9.7.3.7. Простые регулярные выражения

BRE имеют ряд отличий от ERE. В BRE знаки |, + и ? теряют специальное значение, а замены им нет. Границы количества окружаются символами \{ и \}, тогда как { и } рассматриваются как обычные символы. Вложенные подвыражения помещаются между \( и \), а ( и ) представляют обычные символы. Символ ^ воспринимается как обычный, если только он не находится в начале RE или подвыражения в скобках, $ — тоже обычный символ, если он находится не в конце RE или в конце подвыражения в скобках, и \* — обычный символ, когда он находится в начале RE или подвыражения в скобках (возможно, после начального ^). И, наконец, в BRE работают ссылки назад с одной цифрой, \< и \> — синонимы для [[:<:]] и [[:>:]], соответственно; никакие другие спецсимволы в BRE не поддерживаются.