

# Основы SQL



## Поиск

*К.э.н., доцент Андрей Анатольевич Черноусов*


# Регулярные выражения

Поиск в соответствии с регулярными выражениями позволяет осуществлять оператор **RLIKE** (или его синоним **REGEXP**). Такой поиск предоставляет значительно более гибкие средства для поиска по сравнению с оператором **LIKE**. Обратной стороной медали является более медленное выполнение операции поиска с использованием регулярных выражений по сравнению с оператором **LIKE**. Операторы **RLIKE** и **REGEXP** имеют следующий синтаксис:

```
expr REGEXP pat  
expr RLIKE pat
```

Данные операторы часто используются в выражении **WHERE** и возвращают 1, если выражение **expr** соответствует выражению **pat**, и 0 в противном случае.

Помимо операторов **REGEXP** и **RLIKE** существуют обратные формы: **NOT REGEXP** и **NOT RLIKE**, которые возвращают 0, если выражение **expr** соответствует выражению **pat**, и 1 в противном случае.



**Регулярное выражение** — это шаблон, применяемый к заданному тексту слева направо. Большая часть символов сохраняет свое значение в шаблоне и означает совпадение с соответствующим символом. Так, регулярное выражение, содержащее обычный текст, например, "грам", соответствует строке, содержащей подстроку "грам". Например, этому регулярному выражению будут соответствовать строки "программирование", "грамм", "грампластинка" и т. п.

В настоящий момент существует несколько диалектов регулярных выражений. В СУБД MySQL реализована расширенная версия предложенной Генри Спенсером реализации регулярных выражений, которая ориентирована на соответствие стандарту **POSIX**.

**POSIX** (англ. *portable operating system interface for Unix* — переносимый интерфейс операционных систем *Unix*) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный **API**), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных **UNIX**-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-*Unix* систем.

Серия стандартов **POSIX** была разработана комитетом 1003 IEEE. Международная организация по стандартизации (ISO) совместно с Международной электротехнической комиссией (IEC) приняли стандарт **POSIX** под названием ISO/IEC 9945.

Версии стандарта **POSIX** являются основой соответствующих версий стандарта **Single UNIX Specification**. Стандарт **POSIX** определяет интерфейс операционной системы, а соответствие стандарту **Single UNIX Specification** определяет реализацию интерфейса и позволяет операционным системам использовать торговую марку **UNIX**. Название «**POSIX**» было предложено Ричардом Столлманом. Введение в POSIX.1 гласит: «Ожидается произношение „позикс“ как „позитив“, а не „посикс“. Произношение опубликовано в целях обнародования стандартного способа ссылки на стандартный интерфейс операционной системы». «**POSIX**» является зарегистрированным товарным знаком IEEE.

## Пример использования оператора RLIKE

```
SELECT 'грамм' RLIKE 'грам', 'грампластинка' RLIKE 'грам';
```

+-----+-----+	
'грамм' RLIKE 'грам'	'грампластинка' RLIKE 'грам'
+-----+-----+	
1	1
+-----+-----+	

```
SELECT 'программирование' RLIKE 'грам';
```

+-----+	
'программирование' RLIKE 'грам'	
+-----+	
1	
+-----+	

Как видно из приведенного примера, регулярное выражение **грам** осуществляет поиск по всему тексту, независимо от того, находится ли подстрока "грам" в начале, середине или конце слова. Часто необходимо привязать регулярное выражение к началу слова, т. е. чтобы регулярное выражение **грам** соответствовало строке, начинающейся с подстроки "грам", например, "грампластинка", но не соответствовало слову "программирование". Для этого используется символ ^, соответствующий началу строки.

```
SELECT 'программирование' RLIKE '^грам',
       'грампластинка' RLIKE '^грам';
```

	0	1
'программирование' RLIKE '^грам'		
'грампластинка' RLIKE '^грам'		

```
SELECT 'грампластинка' RLIKE '^грампластинка$';
```

```
SELECT 'грампластинка — это вам не программирование'  
RLIKE '^грампластинка$';
```

'грампластинка — это вам не программирование' RLIKE '^грампластинка\$'
0

Регулярное выражение `^$` соответствует пустой строке, но не соответствует **NULL**. Для проверки записей на равенство **NULL** следует использовать операторы **IS NULL** и **IS NOT NULL**.  
Создадим таблицу, в которой хранятся полные имена и отчества:

```
CREATE TABLE tbl (  
  id INT(11) NOT NULL,  
  family TINYTEXT NOT NULL,  
  name TINYTEXT NOT NULL,  
  patronymic TINYTEXT NOT NULL  
);  
INSERT INTO tbl VALUES (1, 'Тимирязев', 'Владимир', 'Константинович');  
INSERT INTO tbl VALUES (2, 'Мальшев', 'Юрий', 'Гаврилович');  
INSERT INTO tbl VALUES (3, 'Абрамов', 'Константин', 'Тимурович');  
INSERT INTO tbl VALUES (4, 'Ганюшкин', 'Валерий', 'Владимирович');
```



Извлечем из таблицы **tbl** записи, поле **family** в которых заканчивается символом "в".

```
SELECT * FROM tbl WHERE family RLIKE 'в$';
```

id	family	name	patronymic
1	Тимирязев	Владимир	Константинович
2	Мальшев	Юрий	Гаврилович
3	Абрамов	Константин	Тимурович

Символы **^** и **\$** соответствуют началу и концу строки, но гораздо чаще встает задача привязки регулярного выражения к началу и концу слова. Для этого предназначены последовательности **[[:<:]]** и **[[:>:]]**, соответствующие началу и концу слова соответственно.

```
SELECT 'a word a' REGEXP '[:<:]word[:>:]';
```

```
+-----+
| 'a word a' REGEXP '[:<:]word[:>:]' |
+-----+
|                                     1 |
+-----+
```

```
SELECT 'a xword a' REGEXP '[:<:]word[:>:]';
```

```
+-----+
| 'a xword a' REGEXP '[:<:]word[:>:]' |
+-----+
|                                     0 |
+-----+
```

Символ вертикальной черты **|** (см. ниже) применяется в регулярном выражении для задания альтернативных масок:

**'abc|абв'**

```
SELECT 'abc' RLIKE 'abc|абв', 'абв' RLIKE 'abc|абв';
```

'abc' RLIKE 'abc абв'		'абв' RLIKE 'abc абв'	
	1		1

Специальный символ **|** можно интерпретировать как оператор **ИЛИ**: при его использовании удовлетворяющими шаблону признаются записи, соответствующие либо правой, либо левой частям.

Если шаблон должен включать символ **|** или любой другой специальный символ, например, рассмотренные выше **^** и **\$**, то их необходимо экранировать при помощи двойного обратного слэша **\\** — в этом случае они теряют свое специальное значение и рассматриваются как обычные символы.

Для задания класса символов применяются квадратные скобки (пример ниже). Они ограничивают поиск теми символами, которые в них заключены, например, **[abc]**.

```
SELECT 'a' RLIKE '[abc]' AS a,  
'b' RLIKE '[abc]' AS b,  
'c' RLIKE '[abc]' AS c;
```

a	b	c	
1	1	1	



Регулярному выражению **[abc]** соответствует подстрока, содержащая один символ: либо **a**, либо **b**, либо **c**.

Так, для создания регулярного выражения, соответствующего всем буквам русского алфавита, можно, конечно, перечислить все буквы в квадратных скобках. Это допустимо, но утомительно и неэлегантно. Более кратко такое регулярное выражение можно записать следующим образом:

**' [a-я] '**

Это выражение соответствует всем буквам русского алфавита, поскольку любые два символа, разделяемые дефисом, задают соответствие диапазону символов, находящихся между ними.

Точно таким же образом задается регулярное выражение, соответствующее любому числу:

**' [0-9] '**

Это выражение эквивалентно

**'[0123456789]'**

В примере ниже приводится пример сравнения русской буквы **Л** и английской **z** с регулярным выражением **[а-я]**.

```
SELECT 'Л' RLIKE '[а-я]' AS a;
```

```
+----+
```

```
| a |
```

```
+----+
```

```
| 1 |
```

```
+----+
```

```
SELECT 'z' RLIKE '[а-я]' AS a;
```

```
+----+
```

```
| a |
```

```
+----+
```

```
| 0 |
```

```
+----+
```

Выражение вида

`'[а-я 0-9 ]'`

соответствует либо букве русского алфавита, либо цифре, либо пробелу.


Если в начале класса помещается символ `^`, то смысл выражения инвертируется (пример ниже). Так, выражение

`'[^0-9]'`

соответствует любому символу, кроме цифры.

```
SELECT '7' RLIKE '[^0-9]' AS number, 'a' RLIKE '[^0-9]' AS str;
```

+-----+-----+	
number	str
+-----+-----+	
0	1
+-----+-----+	



Для определения специальных последовательностей внутри строк в СУБД MySQL используется С-нотация, в которой экранирование обычных символов приводит к их специальной интерпретации:


- `\t` — символ табуляции;
- `\f` — конец файла;
- `\n` — символ перевода строки;
- `\r` — символ возврата каретки;
- `\\` — символ обратного слэша `\`.

Кроме классов, которые могут создать разработчики, путем комбинирования отдельных символов и их диапазонов в стандарте POSIX предусмотрены специальные конструкции классов, представленные в таблице ниже.



# Классы символов POSIX регулярных выражений

Класс	Описание
<code>[:alnum:]</code>	Алфавитно-цифровые символы
<code>[:alpha:]</code>	Алфавитные символы
<code>[:blank:]</code>	Символы пробела или табуляции
<code>[:cntrl:]</code>	Управляющие символы
<code>[:digit:]</code>	Десятичные цифры (0–9)
<code>[:graph:]</code>	Графические (видимые) символы
<code>[:lower:]</code>	Символы алфавита в нижнем регистре
<code>[:print:]</code>	Графические или невидимые символы
<code>[:punct:]</code>	Знаки препинания
<code>[:space:]</code>	Символы пробела, табуляции, новой строки или возврата каретки
<code>[:upper:]</code>	Символы алфавита в верхнем регистре
<code>[:xdigit:]</code>	Шестнадцатеричные цифры



Приведенные выше классы определяют диапазоны символов, так, например, регулярное выражение `[0-9]` эквивалентно `[[:digit:]]`. В примере ниже демонстрируется применение классов символов `[[:digit:]]`, `[[:alpha:]]` и `[[:alnum:]]` с цифрой `1` и символом `"a"`. Важно отметить, что алфавитные символы удовлетворяют всем алфавитным символам, включая русские и английские.



```
SELECT '1' RLIKE '[:digit:]', 'a' RLIKE '[:digit:]';
```

```
+-----+
| '1' RLIKE '[:digit:]' | 'a' RLIKE '[:digit:]' |
+-----+
|                        1 |                        0 |
+-----+
```

```
SELECT '1' RLIKE '[:alpha:]', 'a' RLIKE '[:alpha:]';
```

```
+-----+
| '1' RLIKE '[:alpha:]' | 'a' RLIKE '[:alpha:]' |
+-----+
|                        0 |                        1 |
+-----+
```

```
SELECT '1' RLIKE '[:alnum:]', 'a' RLIKE '[:alnum:]';
```

```
+-----+
| '1' RLIKE '[:alnum:]' | 'a' RLIKE '[:alnum:]' |
+-----+
|                        1 |                        1 |
+-----+
```

Выражение в квадратных скобках соответствует только одному символу и часто применяется совместно с так называемыми квантификаторами. Это символы `?`, `+` и `*`, которые следуют сразу за символом и изменяют число вхождений этого символа в строку:

- `?` — символ либо входит в строку один раз, либо вообще в нее не входит;
- `*` — любое число вхождений символа в строку, в том числе и 0;
- `+` — одно или более число вхождений символа в строку.

Символ `?` позволяет сократить выражения вида

```
' ^СУБД|СУБД MySQL$ '
```

до

```
' ^СУБД( MySQL) ?$ '
```

# Использование квантификатора ?

```
SELECT 'СУБД MySQL' RLIKE '^СУБД( MySQL)?$',
'SУБД' RLIKE '^СУБД( MySQL)?$';
```

+-----+		+-----+	
'СУБД MySQL' RLIKE '^СУБД( MySQL)?\$'		'СУБД' RLIKE '^СУБД( MySQL)?\$'	
1		1	

Символ **+** обозначает один или несколько экземпляров элемента непосредственно предшествующего элемента. Так, если необходимо найти подстроку, содержащую одну цифру или более, можно воспользоваться выражением вида

```
'[[:digit:]]+'
```

Ниже демонстрируется использование квантификатора **+**, который удовлетворяет последовательности из одного и большего числа символов.

```
SELECT '1' RLIKE '^[[[:digit:]]]+$',
'453455234' RLIKE '^[[[:digit:]]]+$';
```

SELECT '1' RLIKE '^[[[:digit:]]]+\$',		'453455234' RLIKE '^[[[:digit:]]]+\$';	
1	1	1	1

```
SELECT '' RLIKE '^[[[:digit:]]]+$',
'45.3455234' RLIKE '^[[[:digit:]]]+$';
```

SELECT '' RLIKE '^[[[:digit:]]]+\$',		'45.3455234' RLIKE '^[[[:digit:]]]+\$';	
0	0	0	0

Символ \* используется для любого числа вхождений строки в подстроку, в том числе и нулевого, т. е. регулярное выражение (пример ниже)

`'^[[:digit:]]*'`

соответствует либо пустой строке, либо строке, содержащей только цифры, причем их количество не ограничено.

```
SELECT '1' RLIKE '^[[:digit:]]*$',
'453455234' RLIKE '^[[:digit:]]*';
```

SELECT '1' RLIKE '^[[:digit:]]*\$',		'453455234' RLIKE '^[[:digit:]]*';	
1	1	1	1

```
SELECT '' RLIKE '^[[:digit:]]*$',
'45.3455234' RLIKE '^[[:digit:]]*';
```

SELECT '' RLIKE '^[[:digit:]]*\$',		'45.3455234' RLIKE '^[[:digit:]]*';	
1	0	0	0

Помимо круглых и квадратных скобок, в регулярных выражениях также применяются фигурные скобки. Они предназначены для указания числа или диапазона повторения элемента:

**$xu\{2\}$**  соответствует строке, в которой за  $x$  следуют два символа  $u$ , т. е.  $xuu$ ;

**$xu\{2, \}$**  соответствует строке, в которой за  $x$  следует не менее двух  $u$  (но может быть и больше);

**$xu\{2, 6\}$**  соответствует строке, в которой за  $x$  следует от двух до шести  $u$ .

Для указания количества вхождений не одного символа, а их последовательности, используются круглые скобки:

**$x(yz)\{2, 6\}$**  соответствует строке, в которой за  $x$  следует от двух до шести последовательностей  $yz$ ;

**$x(yz)^*$**  соответствует строке, в которой за  $x$  следует ноль и более последовательностей  $yz$ .



В примере ниже демонстрируется регулярное выражение, удовлетворяющее числу формата `###.##`. Целая часть может состоять из любого количества цифр, а дробная часть всегда состоит из двух.

```
SELECT '123.90' RLIKE '^[[:digit:]]*\.\[[:digit:]]{2}$';
```

```
+-----+
| '123.90' RLIKE '^([[:digit:]]*\.[[:digit:]]{2})$' |
+-----+
|                                                    1 |
```

```
SELECT '123' RLIKE '^[[[:digit:]]*\.\. [[[:digit:]]]{2}$';
```

[illegible]



Как видно из приведенного примера, элемент регулярного выражения, ответственного за дробную часть числа `\\.[:digit:]{2}`, группируется при помощи круглых скобок, за которыми следует спецсимвол `?`, требующий вхождения подстроки `0` или `1` раз.

# Полнотекстовый поиск

Помимо поиска по регулярным выражениям, в СУБД MySQL предусмотрен так называемый режим полнотекстового поиска, который не требует применения шаблонов. Данный режим предоставляет широкие возможности поиска в тексте и выполняется гораздо быстрее поиска с использованием регулярных выражений и оператора `LIKE` благодаря специальному индексу `FULLTEXT`. Следует помнить, что индексация столбца, в том числе и индексом `FULLTEXT`, требует дополнительного объема памяти для хранения индексов, иногда превышающего объем основных данных в несколько раз, и приводит к замедлению операций вставки и удаления при помощи операторов `INSERT` и `DELETE`.

Полнотекстовый поиск в СУБД MySQL на сегодняшний день поддерживается только для таблиц типа `MyISAM` и столбцов `CHAR`, `VARCHAR` и `TEXT`.

# Индекс FULLTEXT

Для использования возможностей полнотекстового поиска необходимо проиндексировать текстовые столбцы таблицы при помощи индекса **FULLTEXT**. На необходимость индексирования текстового столбца можно указать при создании таблицы в операторе **CREATE TABLE**.

Определение индекса начинается с ключевого слова **FULLTEXT**, после которого следует необязательное ключевое слово **INDEX**. Далее указывается имя индекса (которое может совпадать с именем столбца) и имя индексируемого столбца.

В примере ниже демонстрируется создание таблицы **catalogs**, в которой столбец **name** индексируется индексом **FULLTEXT**.


Если индекс создается только по одному столбцу и его имя совпадает с именем столбца, то имя индекса, размещаемое после ключевого слова **FULLTEXT**, можно опустить.

```
CREATE TABLE catalogs (  
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,  
  name TINYTEXT NOT NULL,  
  PRIMARY KEY (id_catalog),  
  FULLTEXT INDEX name (name)  
) ENGINE=MyISAM;  
DESCRIBE catalogs;
```

Field	Type	Null	Key	Default	Extra
id_catalog	int(11)	NO	PRI	NULL	auto_increment
name	tinytext	NO	MUL		

Индекс **FULLTEXT** можно создать сразу по нескольким столбцам. Так, если поиск следует осуществлять одновременно по нескольким столбцам, то в скобках после имени индекса следует перечислить имена индексируемых столбцов через запятую. В примере ниже приводится оператор **CREATE TABLE**, создающий таблицу **catalogs**, в которой проиндексированы столбцы **name** и **description**.

```
CREATE TABLE catalogs (  
  id_catalog INT(11) NOT NULL AUTO_INCREMENT,  
  name TINYTEXT NOT NULL,  
  description TEXT,  
  PRIMARY KEY (id_catalog),  
  FULLTEXT INDEX search (name, description)  
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```



Имя индекса может быть любым, в примере выше в качестве такого имени выбрано `search`.

Часто задача индексирования столбцов возникает после того, как данные в таблицу уже добавлены. Это может быть вызвано необходимостью добавления поисковых возможностей в уже готовое приложение или переносом таблицы при помощи текстового дампа с ускоренным заполнением таблицы, после которого требуется повторное построение индекса.

Для добавления индекса `FULLTEXT` в уже готовую таблицу предназначен оператор `ALTER TABLE`. Индекс `FULLTEXT`, как и любой другой индекс, создается при помощи спецификации `ADD`, за которой следует определение индекса. В примере ниже приводятся операторы `ALTER TABLE` для создания ранее определенных индексов.





```
ALTER TABLE catalogs ADD FULLTEXT name (name);  
ALTER TABLE catalogs ADD FULLTEXT search (name, description);
```

В таблице не накладывается ограничений на число индексов, допустимо создание нескольких индексов, как показано в примере ниже.

```
ALTER TABLE catalogs ADD FULLTEXT (name),  
ADD FULLTEXT (description),  
ADD FULLTEXT search (name, description);
```

Помимо оператора `ALTER TABLE`, для создания индекса `FULLTEXT` в уже существующей таблице можно использовать оператор `CREATE INDEX`, показанный ниже.

```
CREATE FULLTEXT INDEX name ON catalogs (name);  
CREATE FULLTEXT INDEX search ON catalogs (name, description);
```

## Конструкция MATCH (...) AGAINST (...)

Полнотекстовый поиск выполняется с помощью конструкции `MATCH (...) AGAINST (...)`, которая помещается в условие `WHERE` оператора `SELECT`. В круглых скобках после ключевого слова `MATCH` указываются имена столбцов, по которым производится поиск, а в скобках после `AGAINST` задается фраза, которую необходимо найти.

Полнотекстовый поиск в СУБД MySQL не чувствителен к регистру. Кроме того, при поиске игнорируются так называемые "общеупотребительные" слова. К ним относятся слишком короткие слова (по умолчанию состоящие меньше чем из четырех символов), а также слова, встречающиеся, по крайней мере, в половине записей таблицы. Так, если в таблице имеются только две записи, то полнотекстовый поиск не даст результатов, т. к. каждое слово будет присутствовать, как минимум, в половине записей таблицы.

Для того чтобы осуществлять поиск по столбцу или по комбинации столбцов, столбец и каждая из комбинаций должны быть проиндексированы. То есть создание индекса **FULLTEXT** по двум столбцам **name** и **description** (пример выше) позволит выполнять поиск одновременно по этим двум столбцам.


```
SELECT * FROM products  
WHERE MATCH (name, description) AGAINST ('Celeron')
```

На запросы по отдельным столбцам будет возвращена ошибка 1191 — "Отсутствие проиндексированного столбца". Для осуществления таких запросов необходимо создать дополнительные индексы по столбцам **name** и **description**, помимо уже существующего индекса по обоим столбцам.

В примере ниже представлен дамп таблицы **catalogs**, содержащей два столбца: **name** и **description**. Дальнейшая демонстрация возможностей полнотекстового поиска будет проводиться на примере этой таблицы.

```
CREATE TABLE catalogs (  
id_catalog int(11) NOT NULL auto_increment,  
name tinytext NOT NULL,  
description text NOT NULL,  
PRIMARY KEY (id_catalog),  
FULLTEXT KEY search (name,description)  
);
```

INSERT INTO catalogs VALUES (3, 'Видеоадаптеры', 'Видеоадаптер – это устройство, через которое сигналы изображения выводятся на монитор (или другое устройство). Как правило, имеют собственные видеопроцессоры, чипсет, внутренние шины, специальные BIOS и буферные элементы памяти. При этом видеопамять имеет значительный размер – 64, 128 и 256 Мбайт и больше. Кроме того, память, устанавливаемая на видеоадаптеры работает на больших частотах, нежели ОЗУ компьютера. В каталоге представлены видеоадаптеры нескольких производителей: ASUS, Gigabyte и т. д.');



```
INSERT INTO catalogs VALUES (4, 'Жесткие диски', 'Жесткий диск – это один или несколько дисков из твердого материала с магнитным покрытием. Диски вращаются на общей оси. Кроме собственно дисков, устройство включает в себя магнитные головки для чтения и записи и необходимую электронику. Жесткий диск используется для хранения данных. В каталоге представлены жесткие диски нескольких производителей: Seagate, Maxtor, Western Digital и т.д. ');
```

```
INSERT INTO catalogs VALUES (5, 'Оперативная память', 'Модуль оперативной памяти представляет собой плату для реализации в компьютере памяти с произвольным доступом. Оперативная память отличается более высокой производительностью по сравнению с жестким диском.');
```

В примере ниже демонстрируется поиск слова "Процессор" в таблице `catalogs`, причем из столбца `description` выводятся только первые 20 символов.

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор Gigabyte');
```

name	description
Процессоры	Процессор — это цент...
Материнские платы	Материнская плата — ...

Следует отметить, что ищется целое слово "Процессор" (без учета регистра), слово "видеопроцессоры" из записи с первичным ключом 3 уже не удовлетворяет поиску.

Для каждой строки конструкция **MATCH (...)** **AGAINST (...)** возвращает коэффициент релевантности, т. е. степень сходства между искомой строкой и текстом в таблице. Когда конструкция **MATCH (...)** **AGAINST (...)** используется в выражении **WHERE**, возвращенные строки столбцов автоматически сортируются с помещением в начало списка наиболее релевантных записей. Величина релевантности представляет собой неотрицательное число с плавающей точкой. Релевантность вычисляется на основе количества слов в данной строке столбца, количества уникальных слов в этой строке, общего количества слов в тексте и числа строк, содержащих отдельное слово. Если хоть одно слово из списка **AGAINST()** присутствует в индексированных столбцах, запись появляется в результирующей таблице (пример ниже).

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор Gigabyte');
```

+-----+-----+		
name	description	
+-----+-----+		
Материнские платы	Материнская плата — ...	
Процессоры	Процессор — это цент...	
Видеоадаптеры	Видеоадаптер — это у...	
+-----+-----+		

Слова "Процессор" и "Gigabyte" встречаются вместе в записи, посвященной материнским платам, и по отдельности в записях, посвященных процессорам и видеоадаптерам. Поэтому запись "Материнские платы" получает больший коэффициент релевантности и располагается самой первой в результирующей таблице.



Если поместить конструкцию **MATCH()** в список столбцов, следующих за ключевым словом **SELECT**, можно получить численные значения коэффициента релевантности (пример ниже). Это не приводит к дополнительной нагрузке, поскольку оптимизатор MySQL определяет, что два вызова **MATCH()** идентичны, и выполняет только один поиск.

```
SELECT name,
CONCAT(SUBSTRING(description,1,20),'...') AS description,
MATCH(name, description)
AGAINST ('Процессор Gigabyte') AS coefficient
FROM catalogs
WHERE MATCH (name, description) AGAINST ('Процессор Gigabyte');
```

name	description	coefficient
Материнские платы	Материнская плата — ...	0.56135364417286
Процессоры	Процессор — это цент...	0.32347170307520
Видеоадаптеры	Видеоадаптер — это у...	0.26045211456341

Как видно из приведенных выше примеров, в результат полнотекстового поиска входят записи, содержащие хотя бы одно слово из строки **AGAINST**, и коэффициент релевантности для которых отличен от нуля.

Для разбивки текста на слова СУБД **MySQL** использует очень простой синтаксический анализатор. Словом является любая последовательность символов, состоящая из букв, цифр, одинарных кавычек ' и символов подчеркивания \_. По умолчанию любое слово меньше 4-х символов игнорируется. Если требуется осуществлять поиск по словам, состоящим из трех символов, необходимо изменить значение системной переменной **ft\_min\_word\_len** в конфигурационном файле **my.ini** или **my.cnf** (пример ниже) или передать параметр **--ft\_min\_word\_len=3** при старте сервера **mysqld**.

Если изменение системной переменной **ft\_min\_word\_len** производится после того, как созданы таблицы с индексом **FULLTEXT**, необходимо перестроить индекс (удалить его из таблицы и создать вновь), т. к. структура **MYI-файла** индекса зависит от значения системной переменной **ft\_min\_word\_len**.

```
[mysqld]
```

```
ft_min_word_len=3
```

Каждое слово в искомой фразе оценивается в соответствии с его важностью этом запросе. Таким образом, слово, присутствующее во многих документах, будет иметь меньший вес (и даже, возможно, нулевой, если слово присутствует в более чем половине записей таблицы), как имеющее более низкое смысловое значение в данном конкретном наборе текстов. С другой стороны, редко встречающееся слово получит более высокий вес. Затем полученные значения весов слов объединяются для вычисления релевантности данной строки столбца. Полнотекстовый поиск создавался и настраивался для поиска в большом объеме текста, поэтому он плохо работает с небольшими таблицами, где более эффективно работают регулярные выражения.

Можно составлять достаточно сложные **SQL-запросы**, прибегая к комбинации нескольких конструкций **MATCH (...) AGAINST (...)** при помощи логических операторов **AND**, **OR** и **NOT**. Ниже приводится пример запроса, в котором слова "Процессор" и "Gigabyte" ищутся при помощи двух отдельных конструкций **MATCH (...) AGAINST (...)**, объединенных логическим оператором **AND**.

```
SELECT name,  
CONCAT(SUBSTRING(description,1,20),'...') AS description  
FROM catalogs  
WHERE MATCH (name, description) AGAINST ('Процессор') AND  
MATCH (name, description) AGAINST ('Gigabyte');
```

name	description
Материнские платы	Материнская плата — ...

Как видно из примера возвращается только единственная запись, посвященная материнским платам, которая содержит и слово "Процессор", и слово "Gigabyte".

При использовании комбинации нескольких конструкций **MATCH (...) AGAINST (...)**, объединенных логическими операторами, теряется преимущество сортировки результатов по коэффициенту релевантности, однако в большинстве случаев такая сортировка не нужна, и требуется просто отсортировать результат либо по алфавиту, либо по дате.

# Модификаторы полнотекстового поиска

Конструкция **MATCH (...) AGAINST (...)** позволяет использовать поисковые модификаторы в конструкции **AGAINST**:

**MATCH (col1, col2, ...) AGAINST ('search' [search\_modifier])**

В качестве необязательного модификатора **search\_modifier** могут выступать следующие ключевые слова:

**IN NATURAL LANGUAGE MODE** — естественный режим поиска, данное ключевое слово обозначает обычный режим, рассмотренный в текущем разделе. Ключевое слово **IN NATURAL LANGUAGE MODE** было введено для симметрии, начиная с версии 5.1.7;

**IN BOOLEAN MODE** — логический режим поиска;

**WITH QUERY EXPANSION** — поиск с расширением запроса, введенный, начиная с версии 4.1.1;

**IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION** — данное ключевое слово является синонимом для **WITH QUERY EXPANSION** и введено, начиная с версии 5.1.7.

# Логический режим

Логический режим позволяет более гибко управлять системой поиска, для его активизации в конструкции **AGAINST** после строки с ключевыми словами следует поместить конструкцию **IN BOOLEAN MODE**, как это продемонстрировано в примере ниже.

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
AGAINST ('+Процессор Gigabyte' IN BOOLEAN MODE);
```

+-----+		
name	description	
+-----+		
Процессоры	Процессор — это цент...	
Материнские платы	Материнская плата — ...	
+-----+		

Важно отметить, что запрос, состоящий из одних слов, которые предваряет минус, вернет пустую выборку :

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
AGAINST ('-Процессор -Gigabyte' IN BOOLEAN MODE);
Empty set (0.00 sec)
```

По умолчанию (если ни плюс, ни минус не указаны) данное слово является не обязательным, но содержащие его строки будут оцениваться более высоко. Это имитирует поведение команды **MATCH (...) AGAINST(...)** без модификатора **IN BOOLEAN MODE**. Так, в примере ниже использовать ключевое слово **IN BOOLEAN MODE** не обязательно.

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
AGAINST ('Процессор Gigabyte' IN BOOLEAN MODE);
```

name	description
Процессоры	Процессор — это цент...
Материнские платы	Материнская плата — ...
Видеоадаптеры	Видеоадаптер — это у...

В примере выборка содержит записи, в которых встречается как слово "Процессор", так и слово "Gigabyte".

Помимо **+** и **-**, в логическом режиме предусмотрено еще несколько специальных символов, описание которых приведено в таблице ниже.



## Символ Описание

- +** Предшествующий слову знак + показывает, что это слово должно присутствовать в каждой возвращенной строке
- Предшествующий слову знак - показывает, что это слово не должно присутствовать в какой-либо возвращенной строке
- <** Этот оператор применяется для того, чтобы уменьшить вклад слова в величину релевантности, которая приписывается строке
- >** Этот оператор используется для того, чтобы увеличить вклад слова в величину релевантности, которая приписывается строке
- ( )** Круглые скобки группируют слова в подвыражения
- ~** Предшествующий слову знак ~ воздействует как оператор отрицания, обуславливая негативный вклад данного слова в релевантность строки. Им отмечают нежелательные слова. Строка, содержащая такое слово, будет оценена ниже других, но не будет исключена совершенно, как в случае символа -
- \*** Звездочка является оператором усечения. В отличие от остальных символов, она должна добавляться в конце слова, а не в начале
- "** Фраза, заключенная в двойные кавычки, соответствует только строкам, содержащим эту фразу, написанную буквально

## Режим расширения запроса

Режим расширения запроса применяется, как правило, когда поисковая фраза подразумевает более широкую трактовку запроса. Например, пользователь, выполняя поиск по слову "database" может иметь в виду, что записи, в состав которых входят фразы "MySQL", "Oracle", "DB2" и "MSSQL" тоже должны быть включены в результирующую таблицу. Для включения режима расширения запроса в конструкции **AGAINST** после строки с ключевыми словами следует поместить конструкцию **WITH QUERY EXPANSION** или **IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION**, как это продемонстрировано в примере ниже.

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
AGAINST ('Процессор');
```

# Режим расширения запроса

name	description
Процессоры	Процессор — это цент...
Материнские платы	Материнская плата — ...

```
SELECT name, CONCAT(SUBSTRING(description,1,20),'...') AS description
FROM catalogs
WHERE MATCH (name, description)
AGAINST ('Процессор' WITH QUERY EXPANSION);
```

name	description
Материнские платы	Материнская плата — ...
Процессоры	Процессор — это цент...
Видеоадаптеры	Видеоадаптер — это у...
Оперативная память	Модуль оперативной п...

## Режим расширения запроса

В режиме расширения запроса происходит двойной поиск: сначала ищется искомое слово, а затем слова из результатов поиска. Поэтому при поиске в режиме **WITH QUERY EXPANSION** в последнем примере, помимо записей, содержащих слово "Процессор", найдены также дополнительные записи, в которых это слово не встречается. Следует выбирать как можно более короткие начальные поисковые фразы, т. к. при повторном поиске используются все слова из найденных записей, больше трех символов и встречающиеся в менее чем **50%** записей.