**<대소문자 구분**(?-i)**>**

**SELECT** goods\_nm, REGEXP\_REPLACE(goods\_nm,'(?-i)KG','kg')

**from** tb\_pos\_barcode\_mst

**where** goods\_nm **regexp** '(?-i)KG'

**limit** 10;

**<대소문자 구분하지 않음**(?i)**>**

**SELECT** goods\_nm, REGEXP\_REPLACE(goods\_nm,'(?i)KG','kg')

**from** tb\_pos\_barcode\_mst

**where** goods\_nm **regexp** '(?-i)KG'

**limit** 10;

| Value | Description |
| --- | --- |
| ^ | Matches the beginning of a string. If used with a *match\_parameter* of 'm', it matches the start of a line anywhere within *expression*. |
| $ | Matches the end of a string. If used with a *match\_parameter* of 'm', it matches the end of a line anywhere within *expression*. |
| \* | Matches zero or more occurrences. |
| + | Matches one or more occurrences. |
| ? | Matches zero or one occurrence. |
| . | Matches any character except NULL. |
| | | Used like an "OR" to specify more than one alternative. |
| [ ] | Used to specify a matching list where you are trying to match any one of the characters in the list. |
| [^ ] | Used to specify a nonmatching list where you are trying to match any character except for the ones in the list.  **select** regexp\_replace('8 100','[^0-9]','#'); |
| ( ) | Used to group expressions as a subexpression. |
| {m} | Matches m times. |
| {m,} | Matches at least m times. |
| {m,n} | Matches at least m times, but no more than n times. |
| \n | n is a number between 1 and 9. Matches the nth subexpression found within ( ) before encountering \n. |
| [..] | Matches one collation element that can be more than one character. |
| [::] | Matches character classes. |
| [==] | Matches equivalence classes. |
| \d | Matches a digit character. |
| \D | Matches a nondigit character. |
| \w | Matches a word character. |
| \W | Matches a nonword character. |
| \s | Matches a whitespace character. |
| \S | matches a non-whitespace character. |
| \*? | Matches the preceding pattern zero or more occurrences. |
| +? | Matches the preceding pattern one or more occurrences. |
| ?? | Matches the preceding pattern zero or one occurrence. |
| {n}? | Matches the preceding pattern n times. |
| {n,}? | Matches the preceding pattern at least n times. |
| {n,m}? | Matches the preceding pattern at least n times, but not more than m times. |

**select**

goods\_nm

, REGEXP\_INSTR(goods\_nm,'[0-9]g|[0-9]kg|[0-9]ml|[0-9]l')

, REGEXP\_INSTR(goods\_nm,'[0-9]')

, REGEXP\_substr(goods\_nm,'[0-9]+')

, **substr**(goods\_nm,REGEXP\_INSTR(goods\_nm,'[0-9]'),REGEXP\_INSTR(goods\_nm,'[0-9]g|[0-9]kg|[0-9]ml|[0-9]l')+1-REGEXP\_INSTR(goods\_nm,'[0-9]'))

**from** tb\_pos\_barcode\_mst

**where** REGEXP\_INSTR(goods\_nm,'[0-9]g|[0-9]kg|[0-9]ml|[0-9]l')>0

**limit** 100

|  |  |
| --- | --- |
| **PATTERN** | **WHAT THE PATTERN MATCHES** |
| \* | Zero or more instances of string preceding it |
| + | One or more instances of strings preceding it |
| . | Any single character |
| ? | Match zero or one instances of the strings preceding it. |
| ^ | caret(^) matches Beginning of string |
| $ | End of string |
| [abc] | Any character listed between the square brackets |
| [^abc] | Any character not listed between the square brackets |
| [A-Z] | match any upper case letter. |
| [a-z] | match any lower case letter |
| [0-9] | match any digit from 0 through to 9. |
| [[:<:]] | matches the beginning of words. |
| [[:>:]] | matches the end of words. |
| [:class:] | matches a character class i.e. [:alpha:] to match letters, [:space:] to match white space, [:punct:] is match punctuations and [:upper:] for upper class letters. |
| p1|p2|p3 | Alternation; matches any of the patterns p1, p2, or p3 |
| {n} | n instances of preceding element |
| {m,n} | m through n instances of preceding element |

**Examples with explanation :**

* **Match beginning of string(^):**  
  Gives all the names starting with ‘sa’.Example- sam,samarth.
* SELECT name FROM student\_tbl WHERE name REGEXP '^sa';
* **Match the end of a string($):**  
  Gives all the names ending with ‘on’.Example – norton,merton.

SELECT name FROM student\_tbl WHERE name REGEXP 'on$';

* **Match zero or one instance of the strings preceding it(?):**  
  Gives all the titles containing ‘com’.Example – comedy , romantic comedy.
* SELECT title FROM movies\_tbl WHERE title REGEXP 'com?';
* **matches any of the patterns p1, p2, or p3(p1|p2|p3):**  
  Gives all the names containing ‘be’ or ‘ae’.Example – Abel, Baer.
* SELECT name FROM student\_tbl WHERE name REGEXP 'be|ae' ;
* **Matches any character listed between the square brackets([abc]):**  
  Gives all the names containing ‘j’ or ‘z’.Example – Lorentz, Rajs.
* SELECT name FROM student\_tbl WHERE name REGEXP '[jz]' ;
* **Matches any lower case letter between ‘a’ to ‘z’- ([a-z]) ([a-z] and (.)):**  
  Retrieve all names that contain a letter in the range of ‘b’ and ‘g’, followed by any character, followed by the letter ‘a’.Example – Tobias, sewall.

Matches any single character(.)

SELECT name FROM student\_tbl WHERE name REGEXP '[b-g].[a]' ;

* **Matches any character not listed between the square brackets.([^abc]):**  
  Gives all the names not containing ‘j’ or ‘z’. Example – nerton, sewall.
* SELECT name FROM student\_tbl WHERE name REGEXP '[^jz]' ;
* **Matches the end of words[[:>:]]:**  
  Gives all the titles ending with character “ack”. Example – Black.
* SELECT title FROM movies\_tbl WHERE REGEXP 'ack[[:>:]]';
* **Matches the beginning of words[[:<:]]:**  
  Gives all the titles starting with character “for”. Example – Forgetting Sarah Marshal.
* SELECT title FROM movies\_tbl WHERE title REGEXP '[[:<:]]for';
* **Matches a character class[:class:]:**  
  i.e [:lower:]- lowercase character ,[:digit:] – digit characters etc.  
  Gives all the titles containing alphabetic character only. Example – stranger things, Avengers.
* SELECT title FROM movies\_tbl WHERE REGEXP '[:alpha:]' ;

**Regular Expression Operators**

* ***[expr](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_not-regexp)***[NOT REGEXP](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_not-regexp)***[pat](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_not-regexp)***, ***[expr](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_not-regexp)***[NOT RLIKE](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_not-regexp)***[pat](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_not-regexp)***

This is the same as NOT (***expr*** REGEXP ***pat***).

* ***[expr](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_regexp)***[REGEXP](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_regexp)***[pat](https://dev.mysql.com/doc/refman/5.6/en/regexp.html" \l "operator_regexp)***, [***expr*** RLIKE ***pat***](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp)

Returns 1 if the string ***expr*** matches the regular expression specified by the pattern ***pat***, 0 otherwise. If either ***expr*** or ***pat*** is NULL, the return value is NULL.

[RLIKE](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) is a synonym for [REGEXP](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp), provided for mSQL compatibility.

The pattern can be an extended regular expression, the syntax for which is discussed in [Regular Expression Syntax](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#regexp-syntax). The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

**Note**

Because MySQL uses the C escape syntax in strings (for example, \n to represent the newline character), you must double any \ that you use in your [REGEXP](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) arguments.

Regular expression operations use the character set and collation of the string expression and pattern arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in [Section 10.8.4, “Collation Coercibility in Expressions”](https://dev.mysql.com/doc/refman/5.6/en/charset-collation-coercibility.html). If either argument is a binary string, the arguments are handled in case-sensitive fashion as binary strings.

mysql> SELECT 'Michael!' REGEXP '.\*';

+------------------------+

| 'Michael!' REGEXP '.\*' |

+------------------------+

| 1 |

+------------------------+

mysql> SELECT 'new\*\n\*line' REGEXP 'new\\\*.\\\*line';

+---------------------------------------+

| 'new\*\n\*line' REGEXP 'new\\\*.\\\*line' |

+---------------------------------------+

| 0 |

+---------------------------------------+

mysql> SELECT 'a' REGEXP '^[a-d]';

+---------------------+

| 'a' REGEXP '^[a-d]' |

+---------------------+

| 1 |

+---------------------+

mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';

+----------------+-----------------------+

| 'a' REGEXP 'A' | 'a' REGEXP BINARY 'A' |

+----------------+-----------------------+

| 1 | 0 |

+----------------+-----------------------+

**Warning**

The [REGEXP](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) and [RLIKE](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

**Regular Expression Syntax**

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression hello matches hello and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression hello|world contains the | alternation operator and matches either the hello or world.

As a more complex example, the regular expression B[an]\*s matches any of the strings Bananas, Baaaaas, Bs, and any other string starting with a B, ending with an s, and containing any number of a or n characters in between.

A regular expression for the [REGEXP](https://dev.mysql.com/doc/refman/5.6/en/regexp.html#operator_regexp) operator may use any of the following special characters and constructs:

* ^

Match the beginning of a string.

mysql> SELECT 'fo\nfo' REGEXP '^fo$'; -> 0

mysql> SELECT 'fofo' REGEXP '^fo'; -> 1

* $

Match the end of a string.

mysql> SELECT 'fo\no' REGEXP '^fo\no$'; -> 1

mysql> SELECT 'fo\no' REGEXP '^fo$'; -> 0

* .

Match any character (including carriage return and newline).

mysql> SELECT 'fofo' REGEXP '^f.\*$'; -> 1

mysql> SELECT 'fo\r\nfo' REGEXP '^f.\*$'; -> 1

* a\*

Match any sequence of zero or more a characters.

mysql> SELECT 'Ban' REGEXP '^Ba\*n'; -> 1

mysql> SELECT 'Baaan' REGEXP '^Ba\*n'; -> 1

mysql> SELECT 'Bn' REGEXP '^Ba\*n'; -> 1

* a+

Match any sequence of one or more a characters.

mysql> SELECT 'Ban' REGEXP '^Ba+n'; -> 1

mysql> SELECT 'Bn' REGEXP '^Ba+n'; -> 0

* a?

Match either zero or one a character.

mysql> SELECT 'Bn' REGEXP '^Ba?n'; -> 1

mysql> SELECT 'Ban' REGEXP '^Ba?n'; -> 1

mysql> SELECT 'Baan' REGEXP '^Ba?n'; -> 0

* de|abc

Alternation; match either of the sequences de or abc.

mysql> SELECT 'pi' REGEXP 'pi|apa'; -> 1

mysql> SELECT 'axe' REGEXP 'pi|apa'; -> 0

mysql> SELECT 'apa' REGEXP 'pi|apa'; -> 1

mysql> SELECT 'apa' REGEXP '^(pi|apa)$'; -> 1

mysql> SELECT 'pi' REGEXP '^(pi|apa)$'; -> 1

mysql> SELECT 'pix' REGEXP '^(pi|apa)$'; -> 0

* (abc)\*

Match zero or more instances of the sequence abc.

mysql> SELECT 'pi' REGEXP '^(pi)\*$'; -> 1

mysql> SELECT 'pip' REGEXP '^(pi)\*$'; -> 0

mysql> SELECT 'pipi' REGEXP '^(pi)\*$'; -> 1

* {1}, {2,3}

Repetition; {***n***} and {***m***,***n***} notation provide a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. ***m*** and ***n*** are integers.

* + a\*

Can be written as a{0,}.

* + a+

Can be written as a{1,}.

* + a?

Can be written as a{0,1}.

To be more precise, a{***n***} matches exactly ***n*** instances of a. a{***n***,} matches ***n*** or more instances of a. a{***m***,***n***} matches ***m*** through ***n*** instances of a, inclusive. If both ***m*** and ***n*** are given, ***m*** must be less than or equal to ***n***.

***m*** and ***n*** must be in the range from 0 to RE\_DUP\_MAX (default 255), inclusive.

mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e'; -> 0

mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e'; -> 1

mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e'; -> 1

* [a-dX], [^a-dX]

Matches any character that is (or is not, if ^ is used) either a, b, c, d or X. A - character between two other characters forms a range that matches all characters from the first character to the second. For example, [0-9] matches any decimal digit. To include a literal ] character, it must immediately follow the opening bracket [. To include a literal - character, it must be written first or last. Any character that does not have a defined special meaning inside a [] pair matches only itself.

mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]'; -> 1

mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$'; -> 0

mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$'; -> 1

mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$'; -> 0

mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$'; -> 1

mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$'; -> 0

* [.characters.]

Within a bracket expression (written using [ and ]), matches the sequence of characters of that collating element. characters is either a single character or a character name like newline. The following table lists the permissible character names.

The following table shows the permissible character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

| **Name** | **Character** | **Name** | **Character** |
| --- | --- | --- | --- |
| NUL | 0 | SOH | 001 |
| STX | 002 | ETX | 003 |
| EOT | 004 | ENQ | 005 |
| ACK | 006 | BEL | 007 |
| alert | 007 | BS | 010 |
| backspace | '\b' | HT | 011 |
| tab | '\t' | LF | 012 |
| newline | '\n' | VT | 013 |
| vertical-tab | '\v' | FF | 014 |
| form-feed | '\f' | CR | 015 |
| carriage-return | '\r' | SO | 016 |
| SI | 017 | DLE | 020 |
| DC1 | 021 | DC2 | 022 |
| DC3 | 023 | DC4 | 024 |
| NAK | 025 | SYN | 026 |
| ETB | 027 | CAN | 030 |
| EM | 031 | SUB | 032 |
| ESC | 033 | IS4 | 034 |
| FS | 034 | IS3 | 035 |
| GS | 035 | IS2 | 036 |
| RS | 036 | IS1 | 037 |
| US | 037 | space | ' ' |
| exclamation-mark | '!' | quotation-mark | '"' |
| number-sign | '#' | dollar-sign | '$' |
| percent-sign | '%' | ampersand | '&' |
| apostrophe | '\'' | left-parenthesis | '(' |
| right-parenthesis | ')' | asterisk | '\*' |
| plus-sign | '+' | comma | ',' |
| hyphen | '-' | hyphen-minus | '-' |
| period | '.' | full-stop | '.' |
| slash | '/' | solidus | '/' |
| zero | '0' | one | '1' |
| two | '2' | three | '3' |
| four | '4' | five | '5' |
| six | '6' | seven | '7' |
| eight | '8' | nine | '9' |
| colon | ':' | semicolon | ';' |
| less-than-sign | '<' | equals-sign | '=' |
| greater-than-sign | '>' | question-mark | '?' |
| commercial-at | '@' | left-square-bracket | '[' |
| backslash | '\\' | reverse-solidus | '\\' |
| right-square-bracket | ']' | circumflex | '^' |
| circumflex-accent | '^' | underscore | '\_' |
| low-line | '\_' | grave-accent | '`' |
| left-brace | '{' | left-curly-bracket | '{' |
| vertical-line | '|' | right-brace | '}' |
| right-curly-bracket | '}' | tilde | '~' |
| DEL | 177 |  |  |

mysql> SELECT '~' REGEXP '[[.~.]]'; -> 1

mysql> SELECT '~' REGEXP '[[.tilde.]]'; -> 1

* [=character\_class=]

Within a bracket expression (written using [ and ]), [=character\_class=] represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if o and (+) are the members of an equivalence class, [[=o=]], [[=(+)=]], and [o(+)] are all synonymous. An equivalence class may not be used as an endpoint of a range.

* [:character\_class:]

Within a bracket expression (written using [ and ]), [:character\_class:] represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the ctype(3) manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

| **Character Class Name** | **Meaning** | |
| --- | --- | --- |
| alnum | Alphanumeric characters | |
| alpha | Alphabetic characters | |
| blank | Whitespace characters | |
| cntrl | Control characters | |
| digit | Digit characters | |
| graph | Graphic characters | |
| lower | Lowercase alphabetic characters | |
| print | Graphic or space characters | |
| punct | Punctuation characters | |
| space | Space, tab, newline, and carriage return | |
| upper | Uppercase alphabetic characters | |
| xdigit | Hexadecimal digit characters | |
| **Character Class Name** | | **Meaning** | |

mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+'; -> 1

mysql> SELECT '!!' REGEXP '[[:alnum:]]+'; -> 0

* [[:<:]], [[:>:]]

These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the alnum class or an underscore (\_).

mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]'; -> 1

mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]'; -> 0

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string 1+2 that contains the special + character, only the last of the following regular expressions is the correct one:

mysql> SELECT '1+2' REGEXP '1+2'; -> 0

mysql> SELECT '1+2' REGEXP '1\+2'; -> 0

mysql> SELECT '1+2' REGEXP '1\\+2'; -> 1