Updated for **MYSQL 8**

Ben Forta

MySQL® CRASH COURSE

SECOND EDITION

GET UP AND RUNNING WITH MYSQL

MASTER MYSQL WORKBENCH

LEARN HOW TO RETRIEVE, SORT, AND FILTER DATA

TAKE ADVANTAGE OF REGULAR EXPRESSION FILTERING AND FULL TEXT SEARCHING

DISCOVER POWERFUL MYSQL FEATURES, INCLUDEING STORED PROCEDURES AND TRIGGERS

USE VIEWS AND CURSORS

MANAGE TRANSACTIONAL PROCESSING

CREATE USER ACCOUNTS AND MANAGE SECURITY VIA ACCESS CONTROL

MySQL Crash Course

Using MySQL Regular Expressions

So what do regular expressions have to do with MySQL? As already explained, regular expressions are used to match text by comparing a pattern (the regular expression) with a string of text. MySQL provides rudimentary support for regular expressions with `WHERE` clauses, allowing you to specify regular expressions that are used to filter data retrieved by using `SELECT`.

Note

Just a Subset of the Regular Expression Language If you are already familiar with regular expressions, take note. MySQL supports only a small subset of what most regular expression implementations support, and this chapter covers most of what is supported.

This will all become much clearer with some examples.

Basic Character Matching

We'll start with a very simple example. The following statement retrieves all rows where the column `prod_name` contains the text `1000`:

Input

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '1000'
ORDER BY prod_name;
```

Output

```
+--------------- +
| prod_name      |
+--------------- +
| JetPack 1000   |
+--------------- +
```

Analysis

This statement looks much like the ones that used `LIKE` in Chapter 8, "Using Wildcard Filtering," except that the keyword `LIKE` has been replaced with `REGEXP`. This tells MySQL that what follows is to be treated as a regular expression (one that just matches the literal text `1000`).

So, why bother using a regular expression? Well, in this example, regular expressions really add no value (and probably hurt performance), but consider this next example:

Input

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '.000'
ORDER BY prod_name;
```

Output

```
+---------------- +
| prod_name       |
+---------------- +
| JetPack 1000    |
| JetPack 2000    |
+---------------- +
```

Analysis

Here the regular expression `.000` is used. The period (`.`) is a special character in regular expression language. It means "match any single character," and so both `1000` and `2000` match and are returned.

Of course, this particular example could also be accomplished by using `LIKE` and wildcards (refer to Chapter 8).

Note

`LIKE` Versus `REGEXP` There is one very important difference between `LIKE` and `REGEXP`. Look at these two statements:

```
SELECT prod_name
FROM products
WHERE prod_name LIKE '1000'
ORDER BY prod_name;
```

and:

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '1000'
ORDER BY prod_name;
```

If you were to try them both, you'd discover that the first one returns no data, and the second one returns one row. Why is this?

As you saw in Chapter 8, `LIKE` matches an entire column. If the text to be matched exists in the middle of a column value, `LIKE` will not find it and will not return the row (unless wildcard characters are used). `REGEXP`, on the other hand, looks for matches within column values, and so if the text to be matched exists in the middle of a column value, `REGEXP` finds it and returns the row. This is a very important distinction.

So can `REGEXP` be used to match entire column values (so that it functions like `LIKE`)? Actually, yes, if you also use the `^` and `$` anchors, as explained later in this chapter.

Tip

Matches Are Not Case-Sensitive Regular expression matching in MySQL is not case-sensitive; that is, either case will be matched. To force case-sensitivity, you can use the `BINARY` keyword, as in this example:

```
WHERE prod_name REGEXP BINARY 'JetPack .000'
```

Performing OR Matches

To search for one of two strings (either one or the other), you use the pipe character (`|`), as shown here:

Input

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '1000|2000'
ORDER BY prod_name;
```

Output

```
+-------------- +
| prod_name     |
+-------------- +
| JetPack 1000  |
| JetPack 2000  |
+-------------- +
```

Analysis

Here the regular expression `1000|2000` is used. `|` is the regular expression `OR` operator. It means "match one or the other," and so both `1000` and `2000` match and are returned.

Using `|` is functionally similar to using `OR` statements in `SELECT` statements, with multiple `OR` conditions being consolidated into a single regular expression.

Tip

More Than Two `OR` Conditions You can specify more than two `OR` conditions. For example, you can use `'1000|2000|3000'` to match `1000` or `2000` or `3000`.

Matching One of Several Characters

The character `.` matches any single character. But what if you want to match only specific characters? You can do this by specifying a set of characters enclosed within `[` and `]`, as shown here:

Input

```
SELECT prod_name
FROM products
```

```
WHERE prod_name REGEXP '[123] Ton'
ORDER BY prod_name;
```

Output

```
+-------------- +
| prod_name     |
+-------------- +
| 1 ton anvil   |
| 2 ton anvil   |
+-------------- +
```

Analysis

Here the regular expression `[123] Ton` is used. `[123]` defines a set of characters, and it says to match `1` or `2` or `3`. In this case, both `1 ton` and `2 ton` match and are returned (and there is no `3 ton`).

As you have just seen, `[]` is another form of an `OR` statement. In fact, the regular expression `[123] Ton` is shorthand for `[1|2|3] Ton`, which would also work. But the `[]` characters are needed to define what the `OR` statement is looking for. To better understand this, look at this example:

Input

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '1|2|3 Ton'
ORDER BY prod_name;
```

Output

```
+---------------- +
| prod_name       |
+---------------- +
| 1 ton anvil     |
| 2 ton anvil     |
| JetPack 1000    |
| JetPack 2000    |
| TNT (1 stick)   |
+---------------- +
```

Analysis

Well, this doesn't work. The two required rows are retrieved, but so are three others. This happens because MySQL assumes that you mean to match `1 ton` or `2 ton` or `3 ton`. The `|` character applies to the entire string unless it is enclosed with a set.

Sets of characters can also be negated. That is, you can tell MySQL to match anything *except* the specified characters. To negate a character set, place a `^` at the start of the set. So, whereas `[123]` matches characters `1`, `2`, or `3`, `[^123]` matches anything except those characters.

Matching Ranges

You can use a set to define one or more characters to be matched. For example, the following set matches digits 0 through 9:

```
[0123456789]
```

To simplify this type of set, you can use `-` to define a range. The following is functionally identical to `[0123456789]`:

```
[0-9]
```

Ranges are not limited to complete sets, so `[1-3]` and `[6-9]` are valid ranges, too. In addition, ranges need not be numeric; for example, `[a-z]` will match any alphabetical character.

Here is an example:

Input

```
SELECT prod_name
FROM products
WHERE prod_name REGEXP '[1-5] Ton'
ORDER BY prod_name;
```

Output

```
+-------------- +
| prod_name     |
+-------------- +
| .5 ton anvil  |
| 1 ton anvil   |
| 2 ton anvil   |
+-------------- +
```

Analysis

Here the regular expression `[1-5] Ton` is used. `[1-5]` defines a range, and this expression means "match 1 through 5." In this case, three matches are returned. `.5 ton` is returned because `5 ton` matches (without the `.` character).

Matching Special Characters

The regular expression language is made up of special characters that have specific meanings. You've already seen `.`, `[]`, `|`, and `-`, and there are others, too. So if you need to match those characters, how can you do it? For example, if you want to find values that contain the `.` character, how can you search for it? Look at this example:

Input

```
SELECT vend_name
FROM vendors
WHERE vend_name REGEXP '.'
ORDER BY vend_name;
```

Output

```
+----------------- +
| vend_name        |
+----------------- +
| ACME             |
| Anvils R Us      |
| Furball Inc.     |
| Jet Set          |
| Jouets Et Ours   |
| LT Supplies      |
+----------------- +
```

Analysis

This did not work. `.` matches any character, and so every row is retrieved.

To match special characters, you need to precede them with `\\`. So, for example, `\\-` means “find -,” and, as shown in this example, `\\.` means “find .”:

Input

```
SELECT vend_name
FROM vendors
WHERE vend_name REGEXP '\\.'
ORDER BY vend_name;
```

Output

```
+--------------- +
| vend_name      |
+--------------- +
| Furball Inc.   |
+--------------- +
```

Analysis

This works. `\\.` matches `.`, and so only a single row is retrieved. This process is known as *escaping*, and all characters that have special significance within regular expressions must be escaped this way—including `.`, `|`, `[]`, and all of the other special characters used thus far.

`\\` can also be used to refer to metacharacters (that is, characters that have specific meanings), such as the ones listed in Table 9.1.

Table 9.1 White Space Metacharacters

Metacharacter	Description
`\\f`	Form feed
`\\n`	Line feed
`\\r`	Carriage return
`\\t`	Tab
`\\v`	Vertical tab