Table of Contents

[1. SQL wildcards 1](#_Toc391502368)

[2. Numbers and Like- don't do this! 3](#_Toc391502369)

[3. Dates and Like- there is a better way 5](#_Toc391502370)

[4. To escape a wildcard: 6](#_Toc391502371)

1. SQL wildcards

Wildcard characters are used when you want to do a **partial string match**. Suppose you are looking for any product that has the word "mixer" in the product description. We might want to find 'portable mixer', '6-speed mixer'. We can use the Like operator and a wildcard pattern to find all products which contain the pattern 'mixer'- this will also find 'cement mixers' but that is the way that wildcard matching works. We cannot really do a **word** match, but we can do a pattern match.

Wildcards are used for matching strings. It is acceptable to match digits in text values- such as in an ISBN code. MySQL will do wildcard matching with numeric values but this is not a good idea- wildcards are designed for string matching and numbers are not strings. It is common to do wildcard matches with date values- but that relies on date formats. There are safer ways and generally more efficient ways to test dates and numeric values.

The wildcard patterns are interpreted as wildcards only if you use the LIKE operator.

You can also test using the operator NOT LIKE to find rows that do not match your pattern.

Do not use the LIKE operator if you are not using wildcards. The Like operator is more expensive- if you are testing if the customer city is Chicago, that is an equality test, not a wildcard test.

The wildcard patterns used are:   
 % for zero or more characters, matching any character

\_ for a single character, matching any character

For these examples we are using a table named z\_wildcards in the a\_testbed database with the following rows:

CUSTID CUSTPHONE CUSTADDRESS

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

1 510-239-8989 101 Bush Street

2 510-45-78785 One Bush Street

3 415-809-8989 124 High

4 415-124-2398 15 High Road

5 415-239-8523 1554 Rural Highway 12

1. This is the SQL to create and populate the table.

Create table a\_testbed.z\_wildcards

( cust\_id int

, cust\_phone varchar(12)

, cust\_address varchar(30) )

engine=INNODB;

insert into a\_testbed.z\_wildcards

values (1, '510-239-8989', '101 Bush Street');

insert into a\_testbed.z\_wildcards

values (2, '510-45-78785', 'One Bush Street');

insert into a\_testbed.z\_wildcards

values (3, '415-809-8989', '124 High');

insert into a\_testbed.z\_wildcards

values (4, '415-124-2398', '15 High Road');

insert into a\_testbed.z\_wildcards

values (5, '415-239-8523', '1554 Rural Highway 12');

1. We want to locate customers with an address on Bush.

select cust\_id, cust\_addressfrom a\_testbed.z\_wildcards

where cust\_Address LIKE '%Bush%';

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

| cust\_id | cust\_address |

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

| 1 | 101 Bush Street |

| 2 | One Bush Street |

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

1. We want to locate customers whose phone number is in the 415 area code.

select cust\_id, cust\_phonefrom a\_testbed.z\_wildcards

where cust\_phone LIKE '415%';

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

| cust\_id | cust\_phone |

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

| 3 | 415-809-8989 |

| 4 | 415-124-2398 |

| 5 | 415-239-8523 |

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

1. We want to find customers in the 239 exchange (the middle 3 digits of their phone number). This does not work properly.

select cust\_id, cust\_phonefrom a\_testbed.z\_wildcards

where cust\_phone LIKE '%239%';

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

| cust\_id | cust\_phone |

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

| 1 | 510-239-8989 |

| 4 | 415-124-2398 |

| 5 | 415-239-8523 |

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

1. We can look for the pattern '-239-'.

select cust\_id, cust\_phonefrom a\_testbed.z\_wildcards

where cust\_phone LIKE '%-239-%';

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

| cust\_id | cust\_phone |

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

| 1 | 510-239-8989 |

| 5 | 415-239-8523 |

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

1. Text strings can be harder to match. We want to locate customers with an address on High – and we don't care if it is a street, road, etc. If we try this approach, we also get the Cust 5.

select cust\_id, cust\_addressfrom a\_testbed.z\_wildcards

where cust\_address LIKE '%High%';

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

| cust\_id | cust\_address |

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

| 3 | 124 High |

| 4 | 15 High Road |

| 5 | 1554 Rural Highway 12 |

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

1. If we try adding a space after the word High, we miss Cust 3.

select cust\_id, cust\_addressfrom a\_testbed.z\_wildcards

where cust\_address LIKE '%High %';

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

| cust\_id | cust\_address |

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

| 4 | 15 High Road |

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

Do not use the LIKE operator if you are doing an exact match. It makes no sense to test for a specific phone number by using the following query. This should be an equality test.

select cust\_id, cust\_phonefrom a\_testbed.z\_wildcards

where cust\_phone LIKE '510-239-8989';

1. Numbers and Like- don't do this!

Use the Like operator for string comparisons only. There are safer ways to test numbers and dates. The following tables shows some of the issues in testing numbers with wildcards in MySQL

1. The create table statement uses a zerofill for the column col\_test

Create table a\_testbed.z\_wild\_nbr

( col\_id int not null

, col\_test int(5) zerofill

);

Insert into a\_testbed.z\_wild\_nbr values ( 1, 25);

Insert into a\_testbed.z\_wild\_nbr values ( 2, 250);

Insert into a\_testbed.z\_wild\_nbr values ( 3, 25000);

Insert into a\_testbed.z\_wild\_nbr values ( 4, 250);

Insert into a\_testbed.z\_wild\_nbr values ( 5, 250);

1. This is the data we have. The numbers are zero filled to the left. So the field is filled to the left with zero to a width of 5.

select \*

from a\_testbed.z\_wild\_nbr ;

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

| col\_id | col\_test |

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

| 1 | 00025 |

| 2 | 00250 |

| 3 | 25000 |

| 4 | 00250 |

| 5 | 00250 |

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

1. Testing if 25 appears anywhere in the value. You might think this is cool.

Select \*

From a\_testbed.z\_wild\_nbr

Where col\_test like '%25%';

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

| col\_id | col\_test |

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

| 1 | 00025 |

| 2 | 00250 |

| 3 | 25000 |

| 4 | 00250 |

| 5 | 00250 |

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

1. Testing if the number starts with 25. The zero fill is taken into consideration.

Select \*

From a\_testbed.z\_wild\_nbr

Where col\_test like '25%';

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

| col\_id | col\_test |

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

| 3 | 25000 |

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

This is a MySQL feature (not a flaw). But you should consider if it is appropriate in your task to treat a numeric value as a string.

1. You can test for a numeric 25.

Select \*

From a\_testbed.z\_wild\_nbr

Where col\_test = 25;

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

| col\_id | col\_test |

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

| 1 | 00025 |

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

1. You might think that adding 0 to an integer does not change it values; but here it does change the display format.

Select col\_test + 0 as col\_test

From a\_testbed.z\_wild\_nbr ;

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

| col\_test |

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

| 25 |

| 250 |

| 25000 |

| 250 |

| 250 |

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

1. Dates and Like- there is a better way

MySQL has a very strict format for dates, so using the Like operator for testing dates is not as dangerous as with other dbms. We will soon see functions that might be better used for this type of testing.

1. This will match exam dates in the year 2015- but we get a warning

select ex\_id, ex\_date

from a\_vets.vt\_exam\_headers

where ex\_date like '2015%'

limit 5;

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

| ex\_id | ex\_date |

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

| 3104 | 2015-01-09 16:30:00 |

| 3288 | 2015-01-31 09:00:00 |

| 3321 | 2015-02-17 10:45:00 |

| 3322 | 2015-02-10 09:15:00 |

| 3323 | 2015-02-25 14:30:00 |

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

5 rows in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '2015%' for column 'ex\_date' at

row 1

1. This gets rows where the exam date is in October (month 10), or is the 10th of the month or is in the year 2010, 1910, or a value where part of the time is '10'

select ex\_id, ex\_date

from a\_vets.vt\_exam\_headers

where ex\_date like '%10%'

;

Selected rows

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

| ex\_id | ex\_date |

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

| 3105 | 2014-10-10 09:15:00 |

| 3202 | 2014-10-03 14:30:00 |

| 3282 | 2014-11-23 10:30:00 |

| 3304 | 2014-11-06 10:30:00 |

| 3322 | 2015-02-10 09:15:00 |

15 rows in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '%10%' for column 'ex\_date' at row 1

1. This actually uses the format for the date to pick out the month 10

select ex\_id, ex\_date

from a\_vets.vt\_exam\_headers

where ex\_date like '%-10-%';

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

| ex\_id | ex\_date |

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

| 3001 | 2014-10-24 10:45:00 |

| 3010 | 2014-10-22 10:45:00 |

| 3105 | 2014-10-10 09:15:00 |

| 3202 | 2014-10-03 14:30:00 |

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

Warning (Code 1292): Incorrect datetime value: '%-10-%' for column 'ex\_date' at

row 1

These three queries all produced a warning that there is an incorrect datetime value . I don't know if MySQL will ever change the default format it uses for dates when it casts them to string- but if it does all code that uses the Like operator with date values will have to be inspected and possibly changed.

It is important to remember that date values are not strings. We write date value as string because that is the choice we have. We display dates as string since our output results are usually strings. And date values can be expressed in many different formats. The following are all different string formats for the same date value.

September 3, 2012

Labor Day 2012

2012/09/03

2012-Sep-03

September Third in the year Two Thousand Twelve

The first Monday in September 2012

1. To escape a wildcard:

Suppose we had data values that included the % symbol and we wanted to search for those values. The Where clause would start as

WHERE col\_1 LIKE '%%%'

But this is an obvious problem since we need to indicate that the middle % is really just a percent symbol and not a wildcard. You need to define an escape character for this query- I am going to use the \character which is the default escape CHARACTER.

Now my Where clause looks like this:

WHERE col\_1 LIKE '%\%%'

The first % is a wildcard, the second % is the literal character and the third % is again a wild card.

If you are looking for a value which contains two % characters, you could use

WHERE col\_1 LIKE '%\%\%%'

We will discuss regular expressions soon which provides for more precise matching.