Table of Contents

1. Dates

Date values are essential to most systems. But date values can be confusing.

1.1. Date values are not stored as strings.

MySQL accepts a limited range of literals for a date. You can use the pattern 'YYYY-MM-DD' or 'YYYY/MM/DD' or 'YYYYMMDD' as long as "the string makes sense as a date" For example you can insert '20071205' as a date but not '20071305' since we do not have a month 13. Depending on other setting this might be taken as a date value '0000-00-00' or just rejected.

You can format the display of a date- we will discuss that in another unit.

1.2. Date versus DateTime

MySQL has several types that can be used to store date- we use only two types in this class: date and datetime. A column defined as a **datetime** type always contains both a date component and a time component. A column defined as a **date** type contains only a date component

Demo 01: Here is little table for the a_testbed database to experiment with these values and testing. The table has a date column and a datetime column

```
create table  a_testbed.z_tst_dates_1 (
   id int primary key
 , col_date  date not null
 , col_datetime datetime not null
 );

/* we insert a row with date only values and a row with date&time values */
insert into a_testbed.z_tst_dates_1 values
   (1, '2015-06-12', '2015-06-12')
,  (2, '2015-06-12  08:45:00', '2015-06-12 08:45:00')
;
Query OK, 2 rows affected, 1 warning (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 1
```

Always look at the warnings. If you did not set \W, then use the command show warning.

```
Note (code 1292) Incorrect date value: '2015-06-12  08:45:00' for column
'col_date' at row 2
```

If we display the tables, we see that the date type column has just the dates, MySQL did a cast for you. And the datetime type has a time component.

```
select * from a_testbed.z_tst_dates_1;
+----+------------+---------------------+
| id | col_date   | col_datetime        |
+----+------------+---------------------+
|  1 | 2015-06-12 | 2015-06-12 00:00:00 |
|  2 | 2015-06-12 | 2015-06-12 08:45:00 |
+----+------------+---------------------+
```

2. Testing with a Date value

Demo 02: Set up the table with dates

```
create table  a_testbed.z_tst_dates_2 (
   id integer primary key
 , col_date  date not null
 );

insert into a_testbed.z_tst_dates_2 values
   (1, '2015-06-12')
,  (2, '2015-06-10')
,  (3, '2015-06-12')
,  (4, '2014-06-10')
,  (5, '2015-02-12')
,  (6, '2015-12-10')
,  (7, '2015-04-30')
;
```

Demo 03: The following queries filter on the date. The values returned should not be surprising

```
select *
from a_testbed.z_tst_dates_2
where col_date = '2015-06-12';
+----+------------+
| id | col_date   |
+----+------------+
|  1 | 2015-06-12 |
|  3 | 2015-06-12 |
+----+------------+

select *
from a_testbed.z_tst_dates_2
where col_date < '2015-06-12';
+----+------------+
| id | col_date   |
+----+------------+
|  2 | 2015-06-10 |
|  4 | 2014-06-10 |
|  5 | 2015-02-12 |
|  7 | 2015-04-30 |
+----+------------+

select *
from a_testbed.z_tst_dates_2
where col_date BETWEEN '2015-05-01' and  '2015-06-12';
+----+------------+
| id | col_date   |
+----+------------+
|  1 | 2015-06-12 |
|  2 | 2015-06-10 |
|  3 | 2015-06-12 |
+----+------------+
```

3. Testing with a DateTime value

Demo 04: Set up the table

```
create table  a_testbed.z_tst_dates_3 (
      id integer primary key
    , col_datetime  datetime not null
    );

insert into a_testbed.z_tst_dates_3 values
    (1, '2015-06-12')
,   (2, '2015-06-10   08:45:00')
,   (3, '2015-06-12   08:45:00')
,   (4, '2014-06-10   14:45:00')
,   (5, '2015-06-12   23:45:00')
,   (6, '2015-06-12   00:00:00')
;

select * from a_testbed.z_tst_dates_3;
+----+---------------------+
| id | col_datetime        |
+----+---------------------+
|  1 | 2015-06-12 00:00:00 |
|  2 | 2015-06-10 08:45:00 |
|  3 | 2015-06-12 08:45:00 |
|  4 | 2014-06-10 14:45:00 |
|  5 | 2015-06-12 23:45:00 |
|  6 | 2015-06-12 00:00:00 |
+----+---------------------+
```

Demo 05: The following queries filter on the datetime value using a date literal. The testing is done using datetime comparison.

```
select *
from a_testbed.z_tst_dates_3
where col_datetime = '2015-06-12';
-- this gets only the rows with that date and the time set to midnight
+----+---------------------+
| id | col_datetime        |
+----+---------------------+
|  1 | 2015-06-12 00:00:00 |
|  6 | 2015-06-12 00:00:00 |
+----+---------------------+

select *
from a_testbed.z_tst_dates_3
where col_datetime < '2015-06-12';
+----+---------------------+
| id | col_datetime        |
+----+---------------------+
|  2 | 2015-06-10 08:45:00 |
|  4 | 2014-06-10 14:45:00 |
+----+---------------------+

select *
from a_testbed.z_tst_dates_3
where col_datetime BETWEEN '2015-05-01' and  '2015-06-12';
+----+---------------------+
| id | col_datetime        |
+----+---------------------+
|  1 | 2015-06-12 00:00:00 |
|  2 | 2015-06-10 08:45:00 |
|  6 | 2015-06-12 00:00:00 |
+----+---------------------+
```

Suppose you want to get all of the rows with a date between '2015-05-01' and '2015-06-12 including the rows with datetime values any time on 2015-06-12.

Demo 06: You can use the comparison operators. Note the second test uses the next date and a < test. We can use a compound test to get all dates from midnight May 1, 2015 up to but not including midnight June 13, 2015.

```
select *
from a_testbed.z_tst_dates_3
where col_datetime >= '2015-05-01' and col_datetime < '2015-06-13'
order by col_datetime;
+----+---------------------+
| id | col_datetime        |
+----+---------------------+
|  2 | 2015-06-10 08:45:00 |
|  1 | 2015-06-12 00:00:00 |
|  6 | 2015-06-12 00:00:00 |
|  3 | 2015-06-12 08:45:00 |
|  5 | 2015-06-12 23:45:00 |
+----+---------------------+
```

Demo 07: You will see the following approach also.

```
select *
from a_testbed.z_tst_dates_3
where col_datetime BETWEEN '2015-05-01' and '2015-06-12 23:59:59';
```

4. 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.

These are the same demos as in the discussion of Like. These each display a Warning about an incorrect datetime.. When possible, use syntax that does not display warning.

Demo 08: This will match exam dates in the year 2015- but we get a warning

```
select ex_id, ex_date
from  vets.vt_exam_headers
where ex_date like '2015%'
limit 5;
+-------+---------------------+
| ex_id | ex_date             |
+-------+---------------------+
|  2205 | 2015-04-08 10:30:00 |
|  2228 | 2015-04-04 12:30:00 |
|  2289 | 2015-04-11 13:00:00 |
|  2290 | 2015-04-11 17:00:00 |
|  2300 | 2015-05-08 09:15:00 |
+-------+---------------------+
5 rows in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '2015%' for column 'ex_date' at
row 1
```

Demo 09: 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  vets.vt_exam_headers
where ex_date like '%10%';
```

```
+-------+---------------------+
| ex_id | ex_date             |
+-------+---------------------+
|  2205 | 2015-04-08 10:30:00 |
|  3001 | 2015-10-24 10:45:00 |
|  3002 | 2015-10-02 13:00:00 |
|  3003 | 2015-10-02 13:00:00 |
|  3010 | 2015-10-22 10:45:00 |
|  3105 | 2015-10-10 09:15:00 |
|  3202 | 2015-10-03 14:30:00 |
|  3304 | 2015-11-06 10:30:00 |
|  3306 | 2015-11-06 10:45:00 |
|  3321 | 2016-02-17 10:45:00 |
|  3322 | 2016-02-10 09:15:00 |
|  3324 | 2016-02-25 10:45:00 |
|  3325 | 2016-01-15 10:45:00 |
|  3409 | 2015-12-27 10:45:00 |
|  3513 | 2015-11-06 10:30:00 |
|  3552 | 2015-11-10 10:30:00 |
|  4282 | 2015-08-23 10:30:00 |
|  4514 | 2015-08-10 10:45:00 |
+-------+---------------------+
18 rows in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '%10%' for column 'ex_date' at
row 1
```

Demo 10: This actually uses the format for the date to pick out the month 10

```
select ex_id, ex_date
from  vets.vt_exam_headers
where ex_date like '%-10-%';
+-------+---------------------+
| ex_id | ex_date             |
+-------+---------------------+
|  3001 | 2015-10-24 10:45:00 |
|  3002 | 2015-10-02 13:00:00 |
|  3003 | 2015-10-02 13:00:00 |
|  3010 | 2015-10-22 10:45:00 |
|  3105 | 2015-10-10 09:15:00 |
|  3202 | 2015-10-03 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.