Table of Contents

1. What is a date?

Dates are always a rather complex piece of data. How do you think of time and time values? Do you think of time as a point of time (July 15, 1902 3:20 p.m.) – if so how much precision do you use when you think of time. Is hours and minutes enough precision- do you need fractions of a second? Or would you just prefer to have the day (July 15, 1902)? Is this Pacific Standard Time? Did we have daylight saving in place in 1902? If so, was it in effect at the location where this data value was entered? If you are asking for the current date and time is this in reference to the computer that runs the server or the computer which runs the client- which could be in different time zones.

Or do you think of time as duration- 5 days and 10 minutes from now?

Do you think of time as a stream- as an analog value that continuously changes? Or is your picture of time digital- clicks of a clock?

Businesses need to record both point in time values for time and duration values and manipulate these values. In this section we focus on point of time values with the understanding that a computer cannot actually store a point of time- there is always a precision involved.

Please note that it is *not* an established business rule that a month is 30 days- a month can be 28, 29, 30 or 31 days. If you are not certain if there is a standard number of days in a month for a particular business situation you should ask the business expert.

Many people find temporal data, the variety of functions for working with temporal data, and the differences in how different dbms handle temporal data to be very confusing. I agree, but think about the variety of places in the past week (a "week" is a temporal concept) where someone stored data about you. How many of these places recorded or tested a temporal component.? Probably most of them. So you cannot just give up and say this is too hard- you need to keep working with it. (and if you are working with more than one dbms, prepare to be even more confused.)

The MySQL manual lists 60 functions in this area- we will break these down into groups- and we will not cover them all. Another thing you may notice with MySQL, particularly with the temporal functions is that sometimes there is more than one function you can use to get a result. For example, suppose I want to get the month from the an_dob column in the table vt_animals. In SQL Server, I can use the expression `month(an_dob)` ; in Oracle I use `extract(month from an_dob)`. In MySQL, either of these will work. Which one is better to use in MySQL? the one you remember.

2. Data types for temporal data

MySQL supports the following data types for temporal data: Date, DateTime. Year, Time, TimeStamp

In class we will work mostly with the Date and DateTime types. You may ignore the other types mentioned here.

Date values have the format CCYY-MM-DD with a range of 1000-01-01 to 9999-12-31 such as 2010-06-15

(You can insert earlier date values - such as '0005-11-26' but the MySQL manual states that these dates are not supported and might not work as expected.)

You can also use the CCYYMMDD format such as 20100615.

DateTime values have the format CCYY-MM-DD hh:mm:ss. The time component defaults to midnight. In the DateTime type, the time component is time of day.

Year We will not use this type in class. Year values have the format CCYY or YY . The 4 digit year has a range of 1901-2155; the 2 digit year has a range of 1970-2069. Note the reduced range for a year- 2155 is not that far away!

Time values have the format hh:mm:ss with a range of +- 839:59:59. A TIME value is elapsed time, not time of day.

TimeStamp values have the format CCYY-MM-DD hh:mm:ss.

2.1. Date literals

MySQL has a standard format for dates which is `'YYYY-MM-DD'`

When you are entering date values, they need to have the year first, then the month and then the day. You can enter a date value with a variety of delimiter between the components- such as

'2015-05-31', '2015/05/31', '2015/5/3', '2015^12#29'. You can use any punctuation character.

You can also enter date values without delimiters- as a string '20150625' or as an integer 20150623- as long as the value "makes sense as a date". Since there is no delimiter with this format you need a 2 digit month and a 2 digit day.

It probably makes the most sense to use the format '2025-08-15', but if you are getting date values from another source you might not have to reformat them for input.

2.2. 4 digit years

For class I will assume that you will enter all date literals using a 4 digit year. Anything else is asking for trouble.

But if you are looking at old code (or code from old coders) you need to be aware of other options. It is a good idea to avoid problems by entering data value as string with 4 digit year values. In these demo the string '000731' is treated as YY=00 and MM=07 and DD=31. The second and third columns use a numeric argument and both are evaluated the same way. Some parsers have had trouble with the second and third version.

Demo 01: The year function assumes you are passing a date and returns the year of that date value.

```
select year('000731'), year (000731), year (731);
+----------------+--------------+------------+
| year('000731') | year (000731) | year (731) |
+----------------+--------------+------------+
|           2000 |          2000 |       2000 |
+----------------+--------------+------------+

select month('000731'), month (000731), month (731);
+-----------------+---------------+-------------+
| month('000731') | month (000731) | month (731) |
+-----------------+---------------+-------------+
|               7 |             7 |           7 |
+-----------------+---------------+-------------+
```

MySQL has rules for interpreting 2 digit year values into 4 digit years. Using 4 digits years is the appropriate style.

3. Getting the current date: CURDATE(), CURRENT_DATE() , NOW()

Most of the demos will focus on the date and datetime types. Most functions treat these interchangeable. There are similar functions for time which are not generally mentioned here.

Demo 02: Current date functions

```
select curdate(), current_date(), current_date;  -- these are synonyms
+------------+---------------+--------------+
| curdate()  | current_date() | current_date |
+------------+---------------+--------------+
| 2015-02-11 | 2015-02-11     | 2015-02-11   |
+------------+---------------+--------------+

select now();
+---------------------+
| now()               |
+---------------------+
| 2015-02-11 20:31:26 |
+---------------------+
```

4. Date values versus string values

Hopefully you noticed that the date literal '2015-05-31' is really a string literal. This is common in database systems and the string '2015-05-31' will be treated as a date if it is used in an expression where MySQL expects a date. We have seen this when we do an insert statement, inserting a value such as '2015-05-31' into a date attribute.

Demo 03: We can use an explicit cast if we want. But you do not commonly see this done.

```
select
  cast('2015-05-31' as date)
, cast('2015-05-31' as datetime)
;
+----------------------------+---------------------------------+
| cast('2015-05-31' as date) | cast('2015-05-31' as datetime) |
+----------------------------+---------------------------------+
| 2015-05-31                 | 2015-05-31 00:00:00            |
+----------------------------+---------------------------------+
```

You will see a lot of MySQL code which treats date values as if they are strings, depending on the format of YYYY-MM-DD. We will also see more standard ways of doing these manipulations.

Demo 04: MySQL is willing to do cast from dates to strings or numbers depending on the usage.

```
select
  current_date
, left(current_date,4) as Str1
, substring(current_date,6,2) as Str2
, substring(current_date,9,2) as Str3;
+--------------+------+------+------+
| current_date | Str1 | Str2 | Str3 |
+--------------+------+------+------+
| 2015-02-11   | 2015 | 02   | 11   |
+--------------+------+------+------+
```

Demo 05: But this is certainly easier and clearer to do. Note that these function return numbers.

```
select
  current_date
, year(current_date) as Str1
, month(current_date) as Str2
, dayofmonth(current_date) as Str3;
+--------------+------+------+------+
| current_date | Str1 | Str2 | Str3 |
+--------------+------+------+------+
| 2015-02-11   | 2015 | 02   | 11   |
+--------------+------+------+------+
```

5. MySQL issues with temporal data

For these demos use a full date value such as '2009-06-25'. All of the date values we use in class will be complete date values, but you do need to be aware of other MySQL date issues such as values such as 2010-00-00.

Prior to version 5.0.2, MySQL checked date values for validity by checking that the month component is between 1 and 12 and the date component is between 1 and 31. This allowed for date values such as February 31, 2009. The current version of MySQL rejects invalid dates.

The temporal types have a 0 value used for invalid dates.

Some date functions accept a 0 value for the date components.

Some date functions return 0 for incomplete dates.

Demo 06:

```
select  month ('2525-00-00');
+----------------------+
| month ('2525-00-00') |
+----------------------+
|                    0 |
+----------------------+
```

In general, if you supply invalid date values to functions, the results could be unexpected.

6. How temporal values are stored

MySQL stores the temporal values as integers. The manual explains this as:

MySQL temporal values Storage:

- **DATE**: A three-byte integer packed as **DD** + **MM**×32 + **YYYY**×16×32
- **DATETIME**: Eight bytes:
- A four-byte integer packed as **YYYY**×10000 + **MM**×100 + **DD**
- A four-byte integer packed as **HH**×10000 + **MM**×100 + **SS**
- **YEAR**: A one-byte integer
- **TIME**: A three-byte integer packed as **DD**×24×3600 + **HH**×3600 + **MM**×60 + **SS**
- **TIMESTAMP**: A four-byte integer representing seconds UTC since the epoch (**'1970-01-01 00:00:00'** UTC)

Since one of the reasons for using a dbms is to isolate the user from the physical storage patterns of the data you should not really need to know that- but in reality sometimes things make more sense when you have some idea about data storage. Don't memorize these!