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.

The MySQL manual lists 60 functions in this area- we will break these down into groups- and we will not cover them all.

2. Data types for temporal data

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

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 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. The Year type is more efficient for storing these values than an integer value. But 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. We will discuss this type when we talk about updating tables.

In class we will work mostly with the Date and Datetime types.

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

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 |
+------------+----------------+--------------+
| 2013-08-07 | 2013-08-07     | 2013-08-07   |
+------------+----------------+--------------+

Select now();
+---------------------+
| now()               |
+---------------------+
| 2013-08-07 21:10:43 |
+---------------------+
```

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. Note what happens with the last column- does that make sense from a business point of view?

```
Select
   current_date
,  left(current_date,4) as Str1
,  substring(current_date,6,2) as Str2
,  substring(current_date,9,2) as Str3
,  right(current_date,2) as Str4;
+--------------+------+------+------+------+
| current_date | Str1 | Str2 | Str3 | Str4 |
+--------------+------+------+------+------+
| 2013-08-07   | 2013 | 08   | 07   | 07   |
+--------------+------+------+------+------+
```

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 |
+--------------+------+------+------+
| 2013-08-07   | 2013 |    8 |    7 |
+--------------+------+------+------+
```

5. Date formatting as a string

Formatting is tedious but gives you some flexibility in how a date value should be display. **When you format a date, the result is a string.**

5.1. Date _Format()

First an example: this shows formatting codes which start with the % character and literals such as the / character and blanks. This also shows that the formatting codes are case specific %m is a two digit month number and %M is the month named spelled out.

Demo 06: This uses a variable @d which is assigned a string; the string will be treated as a date by the function Date_Format

```
set @d := '2011-02-20';
select Date_Format(@d, '%Y/%m/%d'), Date_format(@d, '%M %D');
+-----------------------------+--------------------------+
| Date_format(@d, '%Y/%m/%d') | Date_format(@d, '%M %D') |
+-----------------------------+--------------------------+
| 2011/02/20                  | February 20th            |
+-----------------------------+--------------------------+
```

Some of the format codes and their meaning (see the manual for more)

- %Y 4 digit year
- %m two digit month number
- %c one or two digit month number
- %M month name
- %b month name abbreviated
- %d two digit day number
- %e one or two digit day number
- %D day number with suffix as 4th

- %W weekday name
- %a weekday name abbreviated
- %j day of year as number

5.2. Str_To_Date()

The str_to_Date function uses the same format codes to take a string and return a date. This can be useful if you have been supplied date strings in a particular format.

Demo 07:

```
Select str_to_date( 'July 4, 2012', '%M %e, %Y');
+-----------------------------------------+
| str_to_date( 'July 4, 2012', '%M %e, %Y') |
+-----------------------------------------+
| 2012-07-04                              |
+-----------------------------------------+
```

In this version I skipped the comma after the day format and get a null return value and a warning. Extra blanks do not cause an error.

```
Select str_to_date( 'July 4, 2012', '%M %e %Y');
+-----------------------------------------+
| str_to_date( 'July 4, 2012', '%M %e %Y') |
+-----------------------------------------+
| NULL                                    |
+-----------------------------------------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1411): Incorrect datetime value: 'July 4, 2012' for function
str_to_date
```

6. Extracting part of a date value

These functions are used to retrieve parts of a data value either as a number or a string

```
EXTRACT( temp_component FROM date_exp)
YEAR(date_exp)
MONTH(date_exp)
WEEK(date_exp)
DAYOFMONTH(date_exp)
DAYOFWEEK(date_exp)
DAYOFYEAR(date_exp)
MONTHNAME(date_exp)
DAYNAME(date_exp)
```

Demo 08: Date parts using a different value for the variable @d

```
Set @d = '2009-08-15';

select
YEAR(@d), MONTH(@d), WEEK(@d), DAYOFMONTH(@d),
DAYOFWEEK(@d), DAYOFYEAR(@d),
MONTHNAME(@d), DAYNAME(@d)
\G
*************************** 1. row ***************************
         YEAR(@d): 2009
        MONTH(@d): 8
         WEEK(@d): 32
```

```
   DAYOFMONTH(@d): 15
    DAYOFWEEK(@d): 7
    DAYOFYEAR(@d): 227
    MONTHNAME(@d): August
      DAYNAME(@d): Saturday
```

Demo 09: Extract date part

```
Select hire_date
,        EXTRACT(YEAR   FROM hire_date) AS YearHired
,        EXTRACT(MONTH  FROM hire_date) AS MonthHired
,        EXTRACT(DAY    FROM hire_date) AS DayHired
From a_emp.employees
limit 4;
+------------+-----------+------------+----------+
| hire_date  | YearHired | MonthHired | DayHired |
+------------+-----------+------------+----------+
| 1989-06-17 |      1989 |          6 |       17 |
| 2008-06-17 |      2008 |          6 |       17 |
| 2010-06-12 |      2010 |          6 |       12 |
| 2010-08-01 |      2010 |          8 |        1 |
+------------+-----------+------------+----------+
```

Demo 10: We want the people hired in August of any year.

```
Select emp_id, hire_date
From a_emp.employees
where    EXTRACT(MONTH FROM hire_date) = 8;
+--------+------------+
| emp_id | hire_date  |
+--------+------------+
|    103 | 2010-08-01 |
|    201 | 2004-08-25 |
+--------+------------+
```

Demo 11: Who has been hired in the current year?

```
Select emp_id, hire_date
From a_emp.employees
where    EXTRACT(YEAR FROM hire_date) = EXTRACT(YEAR FROM current_date() );

Empty set (0.01 sec)
```

Demo 12: You can use the simpler functions of Year, Quarter, Month, Day, Hour, Minute, Second to extract parts of the dates.

```
Set @d = '2011-08-15  20:15:33';
Select @d
,       year(@d)  as year,    quarter(@d) as quarter
,       month(@d) as month,   day(@d)     as day
,       hour(@d)  as hour,    minute(@d)  as minute;
+---------------------+------+---------+-------+------+------+--------+
| @d                  | year | quarter | month | day  | hour | minute |
+---------------------+------+---------+-------+------+------+--------+
| 2011-08-15  20:15:33 | 2011 |       3 |     8 |   15 |   20 |     15 |
+---------------------+------+---------+-------+------+------+--------+
```

Demo 13: You can get the word for the date parts.

```
Select @d, dayName(@d), monthname(@d);
```

```
+---------------------+-------------+---------------+
| @d                  | dayName(@d) | monthname(@d) |
+---------------------+-------------+---------------+
| 2011-08-15  20:15:33 | Monday      | August        |
+---------------------+-------------+---------------+
```

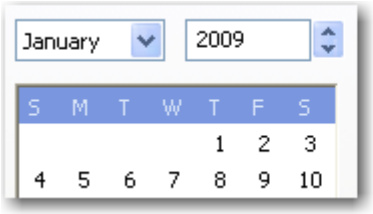
The question- which day of the week is it- as a number- is more complex. Do you consider Sunday to be the first day of the week? Or Monday? Is the first day a numeric value of 0? Or 1?

DayOfWeek uses Sunday =1 and Saturday =7

Weekday uses Monday =0 and Sunday = 6

```
Select @d, dayofWeek(@d), weekday(@d) ;
+---------------------+---------------+-------------+
| @d                  | dayofWeek(@d) | weekday(@d) |
+---------------------+---------------+-------------+
| 2011-08-15  20:15:33 |             2 |           0 |
+---------------------+---------------+-------------+
```

The questions- which week of the year is it is even more complex. In 2009, Jan 1 was a Thursday. So is the first week of the year Jan 1, 2, 3 or Jan 4, 5, 6, 7, 8, 9, 10. Or maybe we start with Monday and the first week is Jan 1, 2, 3, 4 or Jan 5, 6, 7, 8, 9, 10, 11. And is the first week- week 0 or week 1?



MySQL gives you eight choices! You include a mode argument for your choice

Mode	First day of week	Range	Week 1 is the first week
0	Sunday	0-53	with a Sunday in this year
2	Sunday	1-53	with a Sunday in this year
4	Sunday	0-53	with more than 3 days this year
6	Sunday	1-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
7	Monday	1-53	with a Monday in this year
1	Monday	0-53	with more than 3 days this year
3	Monday	1-53	with more than 3 days this year

If you do not include a mode then the default is used. What is the default? This is stored in a variable. The value of this variable can be changed.

```
show variables like 'default_week_format';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| default_week_format | 0     |
+---------------------+-------+
```

A few examples:

Demo 14: using Jan 3, 2009 as the date.

```
set @dtm = '2009-01-03';
Select week(@dtm) as m_default
,   week(@dtm,0) as m_0,    week(@dtm, 1) as m_1
,   week(@dtm,2) as m_2,    week(@dtm, 3) as m_3
,   week(@dtm,4) as m_4,    week(@dtm, 5) as m_5
,   week(@dtm,6) as m_6,    week(@dtm, 7) as m_7;
+-----------+------+------+------+------+------+------+------+------+
| m_default | m_0  | m_1  | m_2  | m_3  | m_4  | m_5  | m_6  | m_7  |
+-----------+------+------+------+------+------+------+------+------+
|         0 |    0 |    1 |   52 |    1 |    0 |    0 |   53 |   52 |
+-----------+------+------+------+------+------+------+------+------+
```

Demo 15: using Jan 10, 2009 as the date.

```
set @dtm = '2009-01-10';
Select week(@dtm) as m_default
,   week(@dtm,0) as m_0,    week(@dtm, 1) as m_1
,   week(@dtm,2) as m_2,    week(@dtm, 3) as m_3
,   week(@dtm,4) as m_4,    week(@dtm, 5) as m_5
,   week(@dtm,6) as m_6,    week(@dtm, 7) as m_7;
+-----------+------+------+------+------+------+------+------+------+
| m_default | m_0  | m_1  | m_2  | m_3  | m_4  | m_5  | m_6  | m_7  |
+-----------+------+------+------+------+------+------+------+------+
|         1 |    1 |    2 |    1 |    2 |    1 |    1 |    1 |    1 |
+-----------+------+------+------+------+------+------+------+------+
```

This is a good time to remind you that you do **not** need to memorize all of these details. I want you to understand that some of these issues can be complex and to be careful when interpreting results from queries that you may be using.

7. Date Arithmetic

To do date manipulation use the date functions. This is a more standard approach to date arithmetic.

- Date_Add(), AddDate()
- Date_Sub(), SubDate()
- DateDiff()

7.1. Intervals

One of the issues with trying to do arithmetic with dates by simply adding a number- such as 40- is that it is not obvious if this is 40 days or 40 years or 40 seconds. A more precise way to handle this is to specify the unit of time you want to add. This is called the Interval. Some of the intervals are shown here. There also are time intervals and more of the combined intervals

- Year
- Month
- Hour
- Quarter
- Day
- Year_Month

Some of the functions default to an interval of Days.

7.2. Date_Add(), AddDate()

Demo 16: Date_Add

```
set @d = '2011-02-20';
select @d
,  Date_add(@d, interval 40 day)   as '40days'
,  Date_add(@d, interval 40 year)  as '40years'
,  Date_add(@d, interval 51 month) as '51months';
+----------------+------------+------------+------------+
```

```
| @d | 40days     | 40years    | 51months   |
+----------------+------------+------------+------------+
| 2011-02-20     | 2011-04-01 | 2051-02-20 | 2015-05-20 |
+----------------+------------+------------+------------+
```

Demo 17: Adding one month to Jan 31 does not result in Feb31- instead the return value is adjusted to the end of the month.

```
Select Date_add('2009-01-31', interval 1 month) as Jan31
,      Date_add('2009-01-28', interval 1 month) as Jan28;
+------------+------------+
| Jan31      | Jan28      |
+------------+------------+
| 2009-02-28 | 2009-02-28 |
+------------+------------+
```

Demo 18: Compound intervals. The delimiter between the components can be any punctuation character

```
set @d := '2011-02-20 19:55:09';
select @d
,  Date_add(@d , interval '4 3' year_month) as '4 years 3 months'
,  Date_add(@d , interval '2-23' day_hour)  as '2 days 23 hours'
;
+---------------------+---------------------+---------------------+
| now()               | 4 years 3 months    | 2 days 23 hours     |
+---------------------+---------------------+---------------------+
| 2011-02-20 19:55:09 | 2015-05-20 19:55:09 | 2011-02-23 18:55:09 |
+---------------------+---------------------+---------------------+
```

Demo 19: The interval value can be negative; if the variable has a time components, then MySQL casts it to a datetime value; if there is no time component, then the value is cast to a date

```
set @d := '2011-02-20';
select @d
,  Date_add(@d, interval '-4 3' year_month)
   as 'minus 4 years 3 months'
,  Date_add(@d, interval '-2' day) as 'minus 2 days'
;
+----------------+-----------------------+--------------+
| @d             | minus 4 years 3 months | minus 2 days |
+----------------+-----------------------+--------------+
| 2011-02-20     | 2006-11-20            | 2011-02-18   |
+----------------+-----------------------+--------------+
```

The AddDate function will default to an interval of days. Date_Add uses the Interval syntax only.

7.3. Date_Sub(), SubDate()

Demo 20: Date_sub is used to subtract a date interval.

```
set @d := '2011-02-20';
select @d
,  Date_sub(@d, interval 40 day)   as '40days'
,  Date_sub(@d, interval 40 year)  as '40years'
,  Date_sub(@d, interval 51 month) as '51months';
+----------------+------------+------------+------------+
| @d             | 40days     | 40years    | 51months   |
```

```
+----------------+------------+------------+------------+
| 2011-02-20     | 2011-01-11 | 1971-02-20 | 2006-11-20 |
+----------------+------------+------------+------------+
```

Demo 21: The SubDate function will default to an interval of days. Date_Sub uses the Interval syntax only.

```
set @d = '2009-07-06' ;
select @d
,  SubDate(@d, interval 40 day)   as '40days'
,  SubDate(@d, 40)   as '40days'
;
+------------+------------+------------+
| @d         | 40days     | 40days     |
+------------+------------+------------+
| 2009-07-06 | 2009-05-27 | 2009-05-27 |
+------------+------------+------------+
```

7.4. DateDiff()

Demo 22: The DateDiff function returns the number of days between two dates.

```
select @d
,  DateDiff(@d, '20090720')   as col_2
,  DateDiff(@d, '20090620')   as col_3
,  DateDiff(@d, '20070706')   as col_4;
+----------------+-------+-------+-------+
| @d             | col_2 | col_3 | col_4 |
+----------------+-------+-------+-------+
| 2009-07-06     |   -14 |    16 |  1325 |
+----------------+-------+-------+-------+
```

Demo 23: The DateDiff function deals with the date component only. The following returns a value of -1.

```
Select DateDiff('2009-07-06 23:59:59', '2009-07-07 00:00:01')   ;
```

Demo 24: DateDiff with order dates.

```
Select ord_id
, ord_date
From a_oe.order_headers
Where dateDiff(date '2013-06-30', ord_date) between 25 and 50;
+--------+---------------------+
| ord_id | ord_date            |
+--------+---------------------+
|    301 | 2013-06-04 00:00:00 |
|    302 | 2013-06-04 00:00:00 |
|    306 | 2013-06-04 00:00:00 |
|    307 | 2013-06-04 00:00:00 |
|    390 | 2013-06-04 00:00:00 |
|    395 | 2013-06-04 00:00:00 |
|    529 | 2013-05-12 00:00:00 |
|    535 | 2013-05-12 00:00:00 |
|    536 | 2013-05-12 00:00:00 |
|    540 | 2013-06-02 00:00:00 |
+--------+---------------------+
```

8. Other temporal functions

8.1. To_Days()

Demo 25: To_Days returns the number of days since the start of the calendar. By itself this is not too interesting.

```
Select current_date()
, To_Days(current_date()) as NowCount
, TO_DAYS('2009-07-20') as "20090720"
, TO_DAYS('1066-10-14') as "btlHst"
, TO_DAYS('0079-08-24') as "MtVsv";
+----------------+----------+----------+--------+-------+
| current_date() | NowCount | 20090720 | btlHst | MtVsv |
+----------------+----------+----------+--------+-------+
| 2013-08-07     |   735452 |   733973 | 389635 | 29090 |
+----------------+----------+----------+--------+-------+
```

Demo 26: You can subtract two of the values to get the number of days between the two dates- or use DateDiff.

```
set @d1 := '2011-08-17';
set @d2 := '2011-12-17';

Select To_Days(@d2)- To_Days(@d1);
+----------------------------+
| To_Days(@d2)- To_Days(@d1) |
+----------------------------+
|                        122 |
+----------------------------+
```

8.2. From_Days()

Demo 27: From_Days gets a number and returns a date based on the number of days since the start of the calendar.

```
Select From_days(5), From_days(500), From_days(689798), From_days(733736);
+--------------+----------------+------------------+-------------------+
| From_days(5) | From_days(500) | From_days(689798) | From_days(733736) |
+--------------+----------------+------------------+-------------------+
| 0000-00-00   | 0001-05-15     | 1888-08-08        | 2008-11-25        |
+--------------+----------------+------------------+-------------------+
```

8.3. Last_Day

Demo 28: Last_day gets date argument and returns the last day of that month

```
set @d1 = '2011-03-25';
set @d2 = '2011-03-31';
select last_day(@d1), last_day(@d2);
+---------------+---------------+
| last_day(@d1) | last_day(@d2) |
+---------------+---------------+
| 2011-03-31    | 2011-03-31    |
+---------------+---------------+

select last_day (From_days(689798));
+-----------------------------+
```

```
| last_day (From_days(689798)) |
+------------------------------+
| 1888-08-31                   |
+------------------------------+
```

9. Direct manipulation of date values

You can do date value plus a number. But you need to pay attention to how this is handled. This is very error prone.

Demo 29: Now() + 40 is **not** 40 days from now and the value returned is numeric in format. Note where the 40 got added.

```
select now()
,      now() + 40   as Plus40
;
+---------------------+----------------------+
| now()               | Plus40               |
+---------------------+----------------------+
| 2013-08-07 21:16:01 | 20130807211641.000000 |
+---------------------+----------------------+
```

Demo 30: Now try this with CurDate()- this is not a good idea!

```
select curdate()
,      curdate()+ 40   as Plus40
,      Month(curdate()+ 40)   as Month40;
+------------+----------+---------+
| curdate()  | Plus40   | Month40 |
+------------+----------+---------+
| 2013-08-07 | 20130847 |    NULL |
+------------+----------+---------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '20130847'
```

Demo 31: Now repeat this by adding 4; In this case we get a date value

```
select now()
,      now()+ 4   as Plus4
,      Month(now()+ 4)   as Month4;
+---------------------+----------------------+--------+
| now()               | Plus4                | Month4 |
+---------------------+----------------------+--------+
| 2013-08-07 21:17:29 | 20130807211733.000000 |      8 |
+---------------------+----------------------+--------+
```

You are advised to use the date functions rather than trying to use these manipulations.

10. 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 32:

```
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.

11. More complex date manipulation

Often you may need to combine various techniques to build dates.

Suppose we want to find a date one month in the future. We can use the following expression.

Demo 33:

```
set @d := '2011-02-20';
select @d
, Date_Add(@d, interval + 1 month);
+----------------+-----------------------------------------------+
| @d             | Date_Add(@d, interval + 1 month) |
+----------------+-----------------------------------------------+
| 2011-02-20     | 2011-03-20                                    |
+----------------+-----------------------------------------------+
```

But what if we want the 15th of next month; we still want to use the Date_Add 1 month approach since we might be running the query in December and need MySQL to get to Jan of the next year. But we want to pull off the day part and replace it with 15. Since we have a set date format of YYYY-MM-DD, we can take the yyyy-mm- part of the string for that date and then concat in the string '15'

Demo 34:

```
  select @d
, cast(Date_Add(@d, interval + 1 month) as char(8)) as WorkingDate;
+----------------+-------------+
| @d             | WorkingDate |
+----------------+-------------+
| 2011-02-20     | 2011-03-    |
+----------------+-------------+
```

Demo 35:

```
  select @d
, concat(cast(Date_Add(@d, interval + 1 month) as char(8)), '15')
  as WorkingDate;
+----------------+-------------+
| @d             | WorkingDate |
+----------------+-------------+
| 2011-02-20     | 2011-03-15  |
+----------------+-------------+
```

We can generally rely on the MySQL casting this string ('2011-03-15') to a date if we use it in a situation where a date is expected. Or we could cast it explicitly.

```
  select @d
```

```
    , cast(
        concat(
          cast(
            Date_Add(@d, interval + 1 month)
            as char(8))
          , '15')
        as Date)
    as WorkingDate;
```

As you can see these get complex very quickly. Remember to build these one step at a time and test as you go.

Demo 36:

We can use this with the order table to calculate a payment due date.

```
    select cast(ord_date as date) as OrderDate,
    cast(concat(cast(Date_Add(ord_date, interval + 1 month) as char(8)), '15')
        as Date)    as DueDate
    from a_oe.order_headers;
```

```
selected rows from the return set
+------------+------------+
| OrderDate  | DueDate    |
+------------+------------+
| 2012-10-01 | 2012-11-15 |
| 2012-10-01 | 2012-11-15 |
| 2012-10-02 | 2012-11-15 |
| 2012-10-12 | 2012-11-15 |
| 2012-11-01 | 2012-12-15 |
| 2013-01-02 | 2013-02-15 |
| 2013-01-03 | 2013-02-15 |
| 2013-01-23 | 2013-02-15 |
| 2013-05-01 | 2013-06-15 |
| 2013-05-01 | 2013-06-15 |
| 2013-05-12 | 2013-06-15 |
| 2013-06-02 | 2013-07-15 |
| 2013-01-02 | 2013-02-15 |
| 2013-01-03 | 2013-02-15 |
```

12. How DateTime 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!