



Database Management Systems: Fundamentals and Introduction to SQL

MySQL Data Types and Applications



MySQL Data Types - Date/Time

- DATE YYYY-MM-DD 2020-01-01
 - TIME hh:mm:ss 23:59:59
 - DATETIME YYYY-MM-DD hh:mm:ss 2020-01-01 23:59:59
 - TIMESTAMP YYYY-MM-DD hh:mm:ss 2020-01-01 23:59:59 UTC
-
- MySQL converts **TIMESTAMP** values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as DATETIME.)
 - include *fsp* (fractional seconds precision) value after data type name if microsecond level required ($fsp = 0 \dots 6$, number of fractional digits DATETIME(*fsp*))
 - MySQL stores dates in ISO format



Date/Time Format Conversion

- Non-ISO format
- `STR_TO_DATE('May 13, 2007','%M %d, %Y');`
- `DATE_FORMAT('1999-12-31','%M %d, %Y');`
- `TIME_FORMAT(time_val, format)`

Date/Time Format Conversion

Date and time formatting functions

Sequence	Meaning
%Y	Four-digit year
%y	Two-digit year
%M	Complete month name
%b	Month name, initial three letters
%m	Two-digit month of year (01..12)
%c	Month of year (1..12)
%d	Two-digit day of month (01..31)
%e	Day of month (1..31)
%W	Weekday name (Sunday..Saturday)
%r	12-hour time with AM or PM suffix
%T	24-hour time
%H	Two-digit hour
%i	Two-digit minute
%s	Two-digit second
%f	Six-digit microsecond
%%	Literal %

```
mysql> SELECT dt,  
-> DATE_FORMAT(dt,'%c/%e/%y %r') AS format1,  
-> DATE_FORMAT(dt,'%M %e, %Y %T') AS format2  
-> FROM datetime_val;
```

dt	format1	format2
1970-01-01 00:00:00	1/1/70 12:00:00 AM	January 1, 1970 00:00:00
1999-12-31 09:00:00	12/31/99 09:00:00 AM	December 31, 1999 09:00:00
2000-06-04 15:45:30	6/4/00 03:45:30 PM	June 4, 2000 15:45:30
2017-03-16 12:30:15	3/16/17 12:30:15 PM	March 16, 2017 12:30:15

Extracting Dates or Times

- Component-extraction functions

Function	Return value
YEAR()	Year of date
MONTH()	Month number (1..12)
MONTHNAME()	Month name (January..December)
DAYOFMONTH()	Day of month (1..31)
DAYNAME()	Day name (Sunday..Saturday)
DAYOFWEEK()	Day of week (1..7 for Sunday..Saturday)
WEEKDAY()	Day of week (0..6 for Monday..Sunday)
DAYOFYEAR()	Day of year (1..366)
HOUR()	Hour of time (0..23)
MINUTE()	Minute of time (0..59)
SECOND()	Second of time (0..59)
MICROSECOND()	Microsecond of time (0..59)
EXTRACT()	Varies



Calculation Using Dates or Times

- Intervals
 - $\text{DATEDIFF}(d1, d2) = d1 - d2$
 - $\text{TIMEDIFF}(t1, t2) = t1 - t2$
- Adding/subtracting values
 - $\text{DATE_ADD}(d, \text{INTERVAL val unit})$
 - $\text{DATE_SUB}(d, \text{INTERVAL val unit})$



Boolean

- MySQL regards Boolean data as TINYINT(1)
 - zero = FALSE
 - nonzero = TRUE
- Example: SET col= FALSE # or 0



Logical Functions

- IS NULL/ IS NOT NULL
- IFNULL(col, 'value if null')
- COALESCE(col1, col2 ... 'value if all null'): return the first non-NULL argument
- IF(condition, 'value if true', 'value if false')

CASE Operator

- One type of the flow control functions (if(), ifnull())
- implement a complex conditional construct

CASE

```
WHEN search_condition THEN statement_list  
[WHEN search_condition THEN statement_list] ...  
[ELSE statement_list]
```

END CASE

Data Type Default Values

- A DEFAULT value clause in a data type specification explicitly indicates a default value for a column.
- The default value specified in a DEFAULT clause can be a literal constant or an expression.

```
CREATE TABLE t1 (  
  -- literal defaults  
  i INT          DEFAULT 0,  
  c VARCHAR(10) DEFAULT '',  
  -- expression defaults  
  f FLOAT        DEFAULT (RAND() * RAND()),  
  b BINARY(16)   DEFAULT (UUID_TO_BIN(UUID())),  
  d DATE         DEFAULT (CURRENT_DATE + INTERVAL 1 YEAR),  
  p POINT        DEFAULT (Point(0,0)),  
  j JSON         DEFAULT (JSON_ARRAY())  
);
```



MySQL Data Types – Suggestions

- Always choose the smallest data type that meets your need.
 - e.g., VARCHAR(50) vs VARCHAR(255)
- Problems with storing files in a database:
 - Increased database size
 - Slower backup
 - Performance problem

Window Functions

- A window function performs an aggregate-like operation on a set of query rows.
- for each row from a query, perform a calculation using rows related to that row
- an aggregate operation groups query rows into a single result row, a window function produces a result for each query row:
 - The row for which function evaluation occurs is called the current row.
 - The query rows related to the current row **OVER** which function evaluation occurs comprise the window for the current row.



Window Function Execution

- Each window operation is signified by inclusion of an **OVER clause** that specifies how to partition query rows into groups for processing by the window function.
- Window functions are permitted only in the select list and ORDER BY clause.
- Query result rows are determined from the FROM clause, after WHERE, GROUP BY, and HAVING processing, and windowing execution occurs before ORDER BY, LIMIT, and SELECT DISTINCT.

Window Functions vs Aggregate Operation

an aggregate operation groups query rows into a single result row

```
mysql> SELECT * FROM sales ORDER BY country, year, product;
```

year	country	product	profit
2000	Finland	Computer	1500
2000	Finland	Phone	100
2001	Finland	Phone	10
2000	India	Calculator	75
2000	India	Calculator	75
2000	India	Computer	1200
2000	USA	Calculator	75
2000	USA	Computer	1500
2001	USA	Calculator	50
2001	USA	Computer	1500
2001	USA	Computer	1200
2001	USA	TV	150
2001	USA	TV	100

```
mysql> SELECT SUM(profit) AS total_profit  
FROM sales;
```

total_profit
7535

```
mysql> SELECT country, SUM(profit) AS country_profit  
FROM sales  
GROUP BY country  
ORDER BY country;
```

country	country_profit
Finland	1610
India	1350
USA	4575

Window Functions Example (1/2)

```
mysql> SELECT
    year, country, product, profit,
    SUM(profit) OVER() AS total_profit,
    SUM(profit) OVER(PARTITION BY country) AS country_profit
FROM sales
ORDER BY country, year, product, profit;
```

year	country	product	profit	total_profit	country_profit
2000	Finland	Computer	1500	7535	1610
2000	Finland	Phone	100	7535	1610
2001	Finland	Phone	10	7535	1610
2000	India	Calculator	75	7535	1350
2000	India	Calculator	75	7535	1350
2000	India	Computer	1200	7535	1350
2000	USA	Calculator	75	7535	4575
2000	USA	Computer	1500	7535	4575
2001	USA	Calculator	50	7535	4575
2001	USA	Computer	1200	7535	4575
2001	USA	Computer	1500	7535	4575
2001	USA	TV	100	7535	4575
2001	USA	TV	150	7535	4575

- **OVER** clause specifies how to partition query rows into groups (**PARTITION BY**) for processing by the window function
- The first OVER clause is empty, which treats the entire set of query rows as a single partition. The window function thus produces a global sum, but does so for each row.

Window Functions Example (2/2)

```
mysql> SELECT
    year, country, product, profit,
    SUM(profit) OVER() AS total_profit,
    SUM(profit) OVER(PARTITION BY country) AS country_profit
FROM sales
ORDER BY country, year, product, profit;
```

year	country	product	profit	total_profit	country_profit
2000	Finland	Computer	1500	7535	1610
2000	Finland	Phone	100	7535	1610
2001	Finland	Phone	10	7535	1610
2000	India	Calculator	75	7535	1350
2000	India	Calculator	75	7535	1350
2000	India	Computer	1200	7535	1350
2000	USA	Calculator	75	7535	4575
2000	USA	Computer	1500	7535	4575
2001	USA	Calculator	50	7535	4575
2001	USA	Computer	1200	7535	4575
2001	USA	Computer	1500	7535	4575
2001	USA	TV	100	7535	4575
2001	USA	TV	150	7535	4575

- The second OVER clause partitions rows by country, producing a sum per partition (per country). The function produces this sum for each partition row.

Window Functions

Name	Description
<u>CUME_DIST()</u>	Cumulative distribution value
<u>DENSE_RANK()</u>	Rank of current row within its partition, without gaps
<u>FIRST_VALUE()</u>	Value of argument from first row of window frame
<u>LAG()</u>	Value of argument from row lagging current row within partition
<u>LAST_VALUE()</u>	Value of argument from last row of window frame
<u>LEAD()</u>	Value of argument from row leading current row within partition
<u>NTH_VALUE()</u>	Value of argument from N-th row of window frame
<u>NTILE()</u>	Bucket number of current row within its partition.
<u>PERCENT_RANK()</u>	Percentage rank value
<u>RANK()</u>	Rank of current row within its partition, with gaps
<u>ROW_NUMBER()</u>	Number of current row within its partition