



# AeroManageX

- Marguerite McGahay
- Bashir Dahir
- Nihar Lodaya

---

# The JSON Data Type



# Creating JSON Values

- A JSON array contains a list of values separated by commas and enclosed within [ and ] characters:

```
["abc", 10, null, true, false]
```

A JSON object contains a set of key-value pairs separated by commas and enclosed within { and } characters:

```
{"k1": "value", "k2": 10}
```

- As the examples illustrate, JSON arrays and objects can contain scalar values that are strings or numbers, the JSON null literal, or the JSON boolean true or false literals. Keys in JSON objects must be strings. Temporal (date, time, or datetime) scalar values are also permitted:

```
["12:18:29.000000", "2015-07-29", "2015-07-29  
12:18:29.000000"]
```

- Nesting is permitted within JSON array elements and JSON object key values:
- ```
[99, {"id": "HK500", "cost": 75.99}, ["hot", "cold"]]  
{ "k1": "value", "k2": [10, 20] }
```



## Normalization, Merging and Auto-wrapping of JSON Values

- When a string is parsed and found to be a valid JSON document, it is also normalized. This means that members with keys that duplicate a key found later in the document, reading from left to right, are discarded. The object value produced by the following `JSON_OBJECT()` call includes only the second `key1` element because that key name occurs earlier in the value, as shown here:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');
+-----+
| JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def') |
+-----+
| {"key1": 1, "key2": "abc"}                          |
+-----+
```



# Normalization, Merging and Auto-wrapping of JSON Values

- Normalization is also performed when values are inserted into JSON columns, as shown here:

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES
>     ('{"x": 17, "x": "red"}'),
>     ('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql> SELECT c1 FROM t1;
+-----+
| c1                |
+-----+
| {"x": "red"}      |
| {"x": [3, 5, 7]}  |
+-----+
```



## Searching and Modifying JSON Values

A JSON path expression selects a value within a JSON document.

Path expressions are useful with functions that extract parts of or modify a JSON document, to specify where within that document to operate. For example, the following query extracts from a JSON document the value of the member with the name key:

```
mysql> SELECT JSON_EXTRACT('{ "id": 14, "name": "Aztalan"}', '$.name');
+-----+
| JSON_EXTRACT('{ "id": 14, "name": "Aztalan"}', '$.name') |
+-----+
| "Aztalan" |
+-----+
```



## Insert JSON Values

In this case, the path `$$[1].b[0]` selects an existing value (`true`), which is replaced with the value following the path argument (`1`). The path `$$[2][2]` does not exist, so the corresponding value (`2`) is added to the value selected by `$$[2]`.

`JSON_INSERT()` adds new values but does not replace existing values:

```
mysql> SELECT JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----+
| JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----+
| ["a", {"b": [true, false]}, [10, 20, 2]]      |
+-----+
```



## Replace JSON Values

In this case, the path `$$[1].b[0]` selects an existing value (`true`), which is replaced with the value following the path argument (`1`). The path `$$[2][2]` does not exist, so the corresponding value (`2`) is added to the value selected by `$$[2]`.

`JSON_INSERT()` adds new values but does not replace existing values:

```
mysql> SELECT JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----+
| JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----+
| ["a", {"b": [true, false]}, [10, 20, 2]]      |
+-----+
```



---

# Exploring Date and Time Data Types in MySQL

# Date and Time Data Type Syntax



The date and time data types for representing temporal values are **DATE**, **TIME**, **DATETIME**, **TIMESTAMP**, and **YEAR**.

| Data Type | Description                 | Supported Range                                                      | Display Format                   |
|-----------|-----------------------------|----------------------------------------------------------------------|----------------------------------|
| DATE      | A date                      | '1000-01-01' to '9999-12-31'                                         | 'YYYY-MM-DD'                     |
| DATETIME  | A date and time combination | '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.499999'         | 'YYYY-MM-DD hh:mm:ss[.fraction]' |
| TIMESTAMP | A timestamp                 | '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.499999' UTC | N/A                              |
| TIME      | A time                      | '-838:59:59.000000' to '838:59:59.000000'                            | 'hh:mm:ss[.fraction]'            |
| YEAR(4)   | A year in 4-digit format    | 1901 to 2155, or 0000                                                | YYYY                             |

# Automatic Initialization and Updating for TIMESTAMP and DATETIME

- **TIMESTAMP and DATETIME columns can auto-initiate and auto-update with the current timestamp.**
- **Auto-initiated columns use the current timestamp when no value is specified for new rows.**
- **Auto-updated columns refresh when any other column in the row changes, except when set explicitly to their current value.**
- **Utilize `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` for automatic properties.**
- **The first `TIMESTAMP` column has both properties enabled by default, unless `explicit_defaults_for_timestamp` is activated.**
- **Consistent fractional seconds precision must be maintained within a column.**



## Fractional Seconds in Time Values

- MySQL has fractional seconds support for TIME, DATETIME, and TIMESTAMP values, with up to microseconds (6 digits) precision.
- To define a column that includes a fractional seconds part, This syntax `type_name (fsp)` is used where `type_name` is TIME, DATETIME, TIMESTAMP, and `fsp` is the fractional seconds precision.

```
CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
```

# What Calendar Is Used By MySQL?



- MySQL uses a proleptic Gregorian calendar, applying Gregorian rules to all dates, even before its official adoption.
- The Julian-to-Gregorian calendar switch in October 1582 resulted in a 10-day discontinuity.
- There are no dates between October 4 and October 15.
- Dates before the cutover are Julian, those after are Gregorian, and during the cutover, dates are nonexistent.
- A calendar applied to dates when it was not actually in use is called proleptic. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL.

# Conversion Between Date and Time Types



- To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information.

## Conversion of DATE values:

- Conversion to a DATETIME or TIMESTAMP value adds a time part of '00:00:00' because the DATE value contains no time information.
- Conversion to a TIME value is not useful; the result is '00:00:00'.

## Conversion of DATETIME and TIMESTAMP values:

- Conversion to a DATE value takes fractional seconds into account and rounds the time part. For example, '1999-12-31 23:59:59.499' becomes '1999-12-31', whereas '1999-12-31 23:59:59.500' becomes '2000-01-01'.
- Conversion to a TIME value discards the date part because the TIME type contains no date information.

## 2-Digit Years in Dates



### 1. Ambiguity of 2-Digit Years

- Dates with 2-digit years are ambiguous as the century is unknown.
- MySQL internally stores years using 4 digits.

### 2. Interpretation Rules

- For DATETIME, DATE, and TIMESTAMP types:
- Year values in the range 00-69 are interpreted as 2000-2069.
- Year values in the range 70-99 are interpreted as 1970-1999.

## 2-Digit Years in Dates



### 3. Exception for YEAR Type

- Numeric 00 becomes 0000 (not 2000).
- To specify zero as 2000, use '0' or '00' as a string.




### 4. Ordering and Functions

- ORDER BY correctly sorts YEAR values with 2-digit years.
- Functions like MIN() and MAX() may convert a YEAR to a number, which can lead to issues with 2-digit years.
- To ensure proper functionality with these functions, convert the YEAR to a 4-digit year format.



# Date and Time Data Types Demo #1

```
1 • use aeromanagex;
2
3 • create table flight (flight_ID int, dep_time time(3), dep_date datetime(6), dep_ts timestamp(0), dep_year year(4));
4
5 • insert into flight
6     (dep_time, dep_date, dep_ts, flight_ID, dep_year)
7     values ('00:08:05', '2023-10-01 00:08:05', '2023-10-01 00:08:05', '1234567890', '2023');
8 • insert into flight
9     (dep_time, dep_date, dep_ts, flight_ID, dep_year)
10    values ('00:12:30', '2023-12-25 00:12:30', '2023-12-25 00:12:30', '0987654321', '2023');
11
12 • table flight;
13
```

Result Grid  Filter Rows:  Export:  Wrap Cell Content: 

|   | flight_ID  | dep_time     | dep_date                   | dep_ts              | dep_year |
|---|------------|--------------|----------------------------|---------------------|----------|
| ▶ | 1234567890 | 00:08:05.000 | 2023-10-01 00:08:05.000000 | 2023-10-01 00:08:05 | 2023     |
|   | 987654321  | 00:12:30.000 | 2023-12-25 00:12:30.000000 | 2023-12-25 00:12:30 | 2023     |



Result  
Grid



Form  
Editor



# Date and Time Data Types Demo #2

Limit to 1000 rows

```
1 • use aeromanagex;
2
3 • create table pilot (pilot_ID int, pilot_FName varchar(45), pilot_LName varchar(45),hire_date date, pilot_DOB datetime(6));
4
5 • insert into pilot
6     (pilot_ID, pilot_FName, pilot_LName, hire_date, pilot_DOB)
7     values (135792468, 'George', 'Washington', '1776-07-04','1732-02-22 10:30:00');
8 • insert into pilot
9     (pilot_ID, pilot_FName, pilot_LName, hire_date, pilot_DOB)
10    values (246813579, 'Dwayne', 'Johnson', '1996-03-10','1972-05-02 13:45:00');
11 • insert into pilot
12     (pilot_ID, pilot_FName, pilot_LName, hire_date, pilot_DOB)
13     values (667428, 'Isaac', 'Newton', '1687-07-07','1643-01-04 12:25:16.42');
14
15 • table pilot;
```

| Result Grid |             | Filter Rows: | Exports:   | Wrap Cell Content:         |
|-------------|-------------|--------------|------------|----------------------------|
| pilot_ID    | pilot_FName | pilot_LName  | hire_date  | pilot_DOB                  |
| 135792468   | George      | Washington   | 1776-07-04 | 1732-02-22 10:30:00.000000 |
| 246813579   | Dwayne      | Johnson      | 1996-03-10 | 1972-05-02 13:45:00.000000 |
| 667428      | Isaac       | Newton       | 1687-07-07 | 1643-01-04 12:25:16.420000 |



Result Grid

# JSON Demo #1

```

31 • select * from people;
32
33 • SELECT people.* FROM people,
34     JSON_TABLE(json_col, '$.people[*]' COLUMNS (
35         FandLname VARCHAR(40) PATH '$.name',
36         address VARCHAR(100) PATH '$.address')
37     ) people;
38
39

```

| Result Grid |                                                    | Filter Rows: | Export:              | Wrap Cell Content: |
|-------------|----------------------------------------------------|--------------|----------------------|--------------------|
|             | json_col                                           | FandName     | address              |                    |
| ▶           | {"people": [{"name": "John Smith", "address": "... | John Smith   | 123 1st Ave, NY, NY  |                    |
|             | {"people": [{"name": "John Smith", "address": "... | Sally Brown  | 246 2nd Ave, NY, NY  |                    |
|             | {"people": [{"name": "John Smith", "address": "... | John Johnson | 1357 3rd Ave, NY, NY |                    |

# JSON Demo #2

```
8
9 • INSERT INTO book (title, tags)
10 VALUES (
11     'Pride and Predjudice',
12     ['Novel', 'Fiction', 'Romance'])
13 );
14
15 • INSERT INTO book (title, tags)
16 VALUES (
17     'Moby Dick',
18     ['Adventure', 'Fiction', 'Epic'])
19 );
20 • select * from book;
21
22 • select json_arrayagg(title) from book;
```

Result Grid |   Filter Rows:  | Export:  | Wrap Cell Content: 

| json_arrayagg(title)                    |
|-----------------------------------------|
| ▶ ["Pride and Predjudice", "Moby Dick"] |