17

MySQL Data Types

MySQL supports a wide variety of data types to support the storage of different kinds of data. This chapter lists the full range of these data types as well as a description of their functionality, syntax, and data storage requirements. For each data type, the syntax shown uses square brackets ([]) to indicate optional parts of the syntax. The following example shows how BIGINT is explained in this chapter:

```
BIGINT[(display_size)]
```

This indicates that you can use `BIGINT` alone or with a display size value. The italics indicate that you do not enter `display_size` literally, but instead place your own value in there. Thus possible uses of `BIGINT` include:

```
BIGINT
BIGINT(20)
```

Like the `BIGINT` type above, many MySQL data types support the specification of a display size. Unless otherwise specified, this value must be an integer between 1 and 255.

Table 17-1 lists the data types and categorizes them as numeric, string, date, or complex. You can then find the full description of the data type in the appropriate section of this chapter.

Table 17-1. . MySQL Data Types.

Data Type	Classification
`BIGINT`	Numeric
`BLOB`	String
`CHAR`	String
`CHARACTER`	String
`CHARACTER VARYING`	String

Table 17-1. . MySQL Data Types.

Data Type	Classification
DATE	Date
DATETIME	Date
DEC	Numeric
DECIMAL	Numeric
DOUBLE	Numeric
DOUBLE PRECISION	Numeric
ENUM	Complex
FLOAT	Numeric
INT	Numeric
INTEGER	Numeric
LONGBLOB	String
LONGTEXT	String
MEDIUMBLOB	String
MEDIUMINT	Numeric
MEDIUMTEXT	String
NCHAR	String
NATIONAL CHAR	String
NATIONAL CHARACTER	String
NATIONAL VARCHAR	String
NUMERIC	Numeric
REAL	Numeric
SET	Complex
SMALLINT	Numeric
TEXT	String
TIME	Date
TIMESTAMP	Date
TINYBLOB	String
TINYINT	Numeric
TINYTEXT	String
VARCHAR	String
YEAR	Date

In some cases, MySQL silently changes the column type you specify in your table creation to something else. These cases are:

- VARCHAR -> CHAR: When the specified VARCHAR column size is less than 4 characters, it is converted to CHAR.

- CHAR -> VARCHAR: When a table has at least one column of a variable length, all CHAR columns greater than three characters in length are converted to VARCHAR.
- TIMESTAMP display sizes: Display sizes for TIMESTAMP fields must be an even value between 2 and 14. A display size of 0 or greater than 14 will convert the field to a display size of 14. An odd-valued display size will be converted to the next highest even value.

Numeric Data Types

MySQL supports all ANSI SQL2 numeric data types. MySQL numeric types break down into two groups: integer and floating point types. Within each group, the types differ basically by the amount of storage required for them. The floating types allow you to optionally specify the number of digits that follow the decimal point. In such cases, the digits value should be an integer from 0 to 30 and no greater than two less than the display size. If you do make the digits value greater than two less than the display size, the display size will automatically change to be two greater than the digits value.

When you insert a value into a column that requires more storage than the data type allows, it will be clipped to the minimum value for that data type (negative values) or the maximum value for that data type (positive values). MySQL will issue a warning when such clipping occurs during ALTER TABLE, LOAD DATA INFILE, UPDATE and multi-row INSERT statements.

The AUTO_INCREMENT attribute may be supplied for at most one column of an integer type in a table. In addition to being an integer type, the column must be either a primary key or the sole column in a unique index. When you attempt an insert into a table with such an integer field and fail to specify a value for that field (or specify a NULL value), a value of one greater than the column's current maximum value will be automatically inserted.

The UNSIGNED attribute may be used with any numeric type. An unsigned column may contain only positive integers or floating point values.

The ZEROFILL attribute indicates that the column should be left padded with zeroes when displayed by MySQL. The number of zeroes padded is determined by the column's display width.

BIGINT

Syntax

```
BIGINT[(display_size)] [AUTO_INCREMENT] [UNSIGNED]
[ZEROFILL]
```

Storage

8 bytes

Description

Largest integer type supporting the range of whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (0 to 18,446,744,073,709,551,615 unsigned). `BIGINT` has some issues when you perform arithmetic on unsigned values. MySQL performs all arithmetic using signed `BIGINT` or `DOUBLE` values. You should therefore avoid performing any arithmetic operations on unsigned `BIGINT` values greater than 9,223,372,036,854,775,807. If you do, you may end up with imprecise results.

DEC

Synonym for `DECIMAL`.

DECIMAL

Syntax

```
DECIMAL[(precision, [scale])] [ZEROFILL]
```

Storage

precision + 2 bytes

Description

Stores floating point numbers where precision is critical, such as for monetary values. `DECIMAL` types require you to specify the precision and scale. The precision is the number of significant digits in the value. The scale is the number of those digits that come after the decimal point. A `BALANCE` column declared as `DECIMAL(9, 2)` would thus store numbers with 9 significant digits, two of which are to the right of the decimal point. The range for this declaration would be -9,999,999.99 to 9,999,999.99. If you specify a number with more decimal points, it is rounded so it fits the proper scale. Values beyond the range of the `DECIMAL` are clipped to fit within the range.

MySQL actually stores `DECIMAL` values as strings, not floating point numbers. It uses one character for each digit as well as one character for the decimal points when the scale is greater than 0 and one character for the sign of negative numbers. When the scale is 0, the value contains no fractional part. Prior to MySQL 3.23, the precision actually had to include space for the decimal and sign. This requirement is no longer in place in accordance with the ANSI specification.

ANSI SQL supports the omission of precision and/or scale where the omission of scale creates a default scale of zero and the omission of precision defaults to an implementation-specific value. In the case of MySQL, the default precision is 10.

DOUBLE

Syntax

```
DOUBLE[(display_size, digits)] [ZEROFILL]
```

Storage

8 bytes

Description

A double-precision floating point number. This type is used for the storage of large floating point values. `DOUBLE` columns can store negative values between -1.7976931348623157E+308 and -2.2250738585072014E-308, 0, and positive numbers between 2.2250738585072014E-308 and 1.7976931348623157E+308.

DOUBLE PRECISION

Synonym for `DOUBLE`.

FLOAT

Syntax

```
FLOAT[(display_size, digits)] [ZEROFILL]
```

Storage

4 bytes

Description

A single-precision floating point number. This type is used for the storage of small floating point numbers. FLOAT columns can store negative values between -3.402823466E+38 and -1.175494351E-38, 0, and positive values between 1.175494351E-38 and 3.402823466E+38.

INT

Syntax

```
INT[(display_size)] [AUTO_INCREMENT] [UNSIGNED]
[ZEROFILL]
```

Storage

4 bytes

Description

A basic whole number with a range of -2,147,483,648 to 2,147,483,647 (0 to 4,294,967,295 unsigned).

INTEGER

Synonym for `INT`.

MEDIUMINT

Syntax

```
MEDIUMINT[(display_size)] [AUTO_INCREMENT] [UNSIGNED]
[ZEROFILL]
```

Storage

3 bytes

Description

A basic whole number with a range of -8,388,608 to 8,388,607 (0 to 16,777,215 unsigned).

NUMERIC

Synonym for `DECIMAL`.

REAL

Synonym for `DOUBLE`.

SMALLINT

Syntax

`SMALLINT[(`*`display_size`*`)] [AUTO_INCREMENT] [UNSIGNED] [ZEROFILL]`

Storage

2 bytes

Description

A basic whole number with a range of -32,768 to 32,767 (0 to 65,535 unsigned).

TINYINT

Syntax

`TINYINT[(`*`display_size`*`)] [AUTO_INCREMENT] [UNSIGNED] [ZEROFILL]`

Storage

1 byte

Description

A basic whole number with a range of -128 to 127 (0 to 255 unsigned).

String Data Types

String data types store various kinds of text data. By default, string columns are sorted and compared in a case-insensitive fashion in accordance with the sorting rules for the default character set. Each string type has a corresponding binary type. For CHAR and VARCHAR, the binary types are declared using the BINARY attribute. All of the TEXT types, however, have corresponding BLOB types as their binary counterparts.

Text fields are compared and sorted in a case-insensitive fashion in accordance with the default character set of the server. Binary fields, on the other hand, are not interpreted according to a character set. Comparisons and sorts occur on a byte-by-byte basis without interpretation. In other words, binary values are treated case-sensitive.

BLOB

Binary form of `TEXT`.

CHAR

Syntax

    CHAR(size) [BINARY]

Size

Specified by the *size* value in a range of 0 to 255 (1 to 255 prior to MySQL 3. 23).

Storage

size bytes

Description

A fixed-length text field. String values with fewer characters than the column's size will be right-padded with spaces. The right-padding is removed on retrieval of the value from the database.

`CHAR(0)` fields are useful for backwards compatibility with legacy systems that no longer store values in the column or for binary values (`NULL` vs. `''`).

CHARACTER

Synonym for `CHAR`.

CHARACTER VARYING

Synonym for `VARCHAR`.

LONGBLOB

Binary form of `LONGTEXT`.

LONGTEXT

Syntax

    LONGTEXT

Size

0 to 4,294,967,295

Storage

length of value + 4 bytes

Description

Storage for large text values.

While the theoretical limit to the size of the text that can be stored in a `LONGTEXT` column exceeds 4GB, the practical limit is much less due to limitations of the MySQL communication protocol and the amount of memory available on both the client and server ends of the communication.

MEDIUMBLOB

Binary form of `MEDIUMTEXT`.

MEDIUMTEXT

Syntax

```
MEDIUMTEXT
```

Size

0 to 16,777,215

Storage

length of value + 3 bytes

Description

Storage for medium-sized text values.

NCHAR

Synonym of `CHAR`.

NATIONAL CHAR

Synonym of `CHAR`.

NATIONAL CHARACTER

Synonym of `CHAR`.

NATIONAL VARCHAR

Synonym of `VARCHAR`.

TEXT

Syntax

`TEXT`

Size

0 to 65,535

Storage

length of value + 2 bytes

Description

Storage for most text values.

TINYBLOB

Binary form of `TINYTEXT`.

TINYTEXT

Syntax

`TINYTEXT`

Size

0 to 255

Storage

length of value + 1 byte

Description

Storage for short text values.

VARCHAR

Syntax

`VARCHAR(`*`size`*`) [BINARY]`

Size

Specified by the *size* value in a range of 0 to 255 (1 to 255 prior to MySQL 3. 23).

Storage

length of value + 1 byte

Description

Storage for variable-length text. Trailing spaces are removed from VARCHAR values when stored in the database in conflict with the ANSI specification.

Date Data Types

MySQL date types are extremely flexible tools for storing date information. They are also extremely forgiving in the belief that it is up to the application, not the database, to validate date values. MySQL only checks that months are in the range of 0-12 and dates are in the range of 0-31. February 31, 2001 is therefore a legal MySQL date. More useful, however, is the fact that February 0, 2001 is a legal date. In other words, you can use 0 to signify dates where you do not know a particular piece of the date.

Though MySQL is somewhat forgiving on the input format, your applications should actually attempt to format all date values in MySQL's native format to avoid any confusion. MySQL always expects the year to be the left-most element of a date format. If you assign illegal values in a SQL operation, MySQL will insert zeroes for that value.

MySQL will also perform automatic conversion of date and time values to integer values when used in an integer context.

DATE

Syntax

`DATE`

Format

YYYY-MM-DD (2001-01-01)

Storage

3 bytes

Description

Stores a date in the range of January 1, 1000 (`'1000-01-01'`) to December 31, 9999 (`'9999-12-31'`) in the Gregorian calendar.

DATETIME

Syntax

`DATETIME`

Format

YYYY-MM-DD hh:mm:ss (2001-01-01 01:00:00)

Storage

8 bytes

Description

Stores a specific time in the range of 12:00:00 AM, January 1, 1000 (`'1000-01-01 00:00:00'`) to 11:59:59 PM, December 31, 9999 (`'9999-12-31 23:59:59'`) in the Gregorian calendar.

TIME

Syntax

`TIME`

Format

hh:mm:ss (06:00:00)

Storage

3 bytes

Description

Stores a time value in the range of midnight (`'00:00:00'`) to 1 second before midnight (`'23:59:59'`).

TIMESTAMP

Syntax

`TIMESTAMP[(`*`display_size`*`)]`

Format

YYYYMMDDhhmmss (20010101060000)

Storage

4 bytes

Description

A simple representation of a point in time down to the second in the range of midnight on January 1, 1970 to one minute before midnight on December 31, 2037. Its primary utility is in keeping track of table modifications. When you insert a `NULL` value into a `TIMESTAMP` column, the current date and time will be inserted instead. When you modify any value in a row with a `TIMESTAMP` column, the first `TIMESTAMP` column will automatically be updated with the current date and time.

YEAR

Syntax

`YEAR`

Format

YYYY (2001)

Storage

1 byte

Description

Stores a year of the Gregorian calendar in the range of 1900 to 2155.

Complex Data Types

MySQL’s complex data types `ENUM` and `SET` are really nothing more than special string types. We break them out because they are conceptually more complex and represent a lead into the SQL3 data types that MySQL may one day support.

ENUM

Syntax

`ENUM(`*`value1, value2, ...`*`)`

Storage

1-255 members: 1 byte
256-65,535 members: 2 bytes

Description

Stores one value of a predefined list of possible strings. When you create an ENUM column, you provide a list of all possible values. Inserts and updates are then allowed to set the column to values only from that list. Any attempt to insert a value that is not part of the enumeration will cause an empty string to be stored instead.

You may reference the list of possible values by index where the index of the first possible value is 0. For example:

```
SELECT COLID FROM TBL WHERE COLENUM = 0;
```

Assuming `COLID` is a primary key column and `COLENUM` is the column of type `ENUM`, this SQL will retrieve the primary keys of all rows with `COLENUM` value equals the first value of that list. Similarly, sorting on `ENUM` columns happens according to index, not string value.

The maximum number of elements allowed for an `ENUM` column is 65,535.

SET

Syntax

`SET(`*`value1, value2, ...`*`)`

Storage

1-8 members: 1 byte
9-16 members: 2 bytes
17-24 members: 3 bytes
25-32 members: 4 bytes
33-64 members: 8 bytes

Description

A list of values taken from a pre-defined set of values. A SET is basically an ENUM that allows multiple values. A SET, however, is not stored according to index but instead as a complex bit map. Given a SET with the members "Orange", "Apple", "Pear", and "Banana", each element is represented by an "on" bit in a byte as shown in Table 17-2

Table 17-2. . MySQL Representations of Set Elements

Member	Decimal Value	Bitwise Representation
Orange	1	0001
Apple	2	0010
Pear	4	0100
Banana	8	1000

The values "Orange" and "Pear" are therefore stored in the database as 5 (0101).

You can store a maximum of 64 values in a SET column. Though you can assign the same value multiple times in a SQL statement updating a SET column, only a single value will actually be stored.