Data Types and Storage Requirements

Data Types and Storage Requirements

MySQL supports many different data types, most of which we'll discuss in the following paragraphs. The term data type refers to the classification of data based on its possible values, the set of operations we can perform on it, and its storage requirements. Values of the INTEGER type can only consist of integers like 0, 42, and 1337. This is different from the DECIMAL type which consists of decimal numbers like 1.61, 3.14, and 100.0. We can perform operations like addition, subtraction, multiplication, and division on INTEGER and DECIMAL values, but these cannot be per-formed on text-based types like CHAR and TEXT.

Numeric Types

MySQL offers the INTEGER (also abbreviated as INT), TINYINT, SMALLINT, MEDIUMINT, and BIGINT data types for storing integer data. These types differ in the number of bytes they occupy to represent a value. This in turn limits the range of integers each type can hold. For example, TINYINT uses 1 byte, so its range is -128 to 127—the range of numbers than can be expressed in binary with 8 bits. INTEGER uses 4 bytes, so its range is larger: -2,147,483,648 to 2,147,483,647.

We can also specify the UNSIGNED attribute with integer-based types. The type consumes the same amount of space but negative values are disallowed in exchange for raising the upper bound. For example, the range of TINYINT UNSIGNED becomes 0 to 255. Both TINYINT and TINYINT UNSIGNED represent a range of 256 integers, but their starting points are -128 and 0 respectively.

The following table shows the storage requirements and range for each of MySQL's integer types, both signed and unsigned:

Data Type	Storage Used (Bytes)	Min. Signed	Max. Signed	Min. Unsigned	Max. Unsigned
TINYINT	1	-128,127	127	0	255
SMALLINT	2	-32,768	32,767	0	65,535
MEDIUMINT	3	-8,388,608	8,388,607	0	6,777,215
INTEGER	4	-2,147,483,648	2,147,483,647	0	4,294,967,295
BIGINT	8	-9,223,372, 036,854,775,	9,223,372,	0	18,446,744,

		808	036,854,775 807		073,709,551, 615

DECIMAL, FLOAT, and DOUBLE are types that support real numbers. We also must provide the precision (the number of total digits) and scale (the number of digits that follow the decimal point) when we use one of these types. DECIMAL(5,2) has a range of -999.99 to 999.99—that is, five digits in total with two of them following the decimal point. We can specify UNSIGNED for these types as well, but doing so only disallows negative values. This is because the upper limit is defined by the precision and scale we provide.

The DECIMAL type is a fixed-point data type which means it preserves the exact precision of its value in calculations. This is useful for representing values like monetary amounts. The maximum precision we can specify for DECIMAL is 65, and the maximum scale is 30. On the other hand, FLOAT and DOUBLE are both floating-point types. Calculations with these types are approximate because some rounding may occur due to how the values are represented internally in the computer. The difference between FLOAT and DOUBLE is the amount of space they occupy, which in turn affects their accuracy. FLOAT is 4-byte single-precision which is generally accurate up to 7 decimal places. DOUBLE is 8-byte double-precision which is generally accurate up to 15 decimal places.

The BIT data type stores a bit-sequence. This is useful for storing bit-field values like flags and bit masks. BIT has a capacity of 1 to 64 bits. BIT(1) can only hold 0 or 1; BIT(2) can hold the binary values 00, 01, 10, and 11; BIT(3) can hold the binary values 000, 001, 010, 011, 100, 101, 110, and 111, and so on. MySQL uses the notation b'value' to specify the value as string of binary digits, like b'101010'.

String Types

MySQL devotes several data types to storing textual data: CHAR, VARCHAR, BINARY, VARBINARY, TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT, BLOB, TINYBLOB, MEDIUMBLOB, and LONGBLOB. The sized types like TINYTEXT and MEDIUMTEXT behave exactly like TEXT although each is constrained by a different maximum amount of text it can hold. The same is true for BLOB and its sized counterparts, TINYBLOB, MEDIUMBLOB, and LONGBLOB.

We must provide a length when we specify a CHAR or VARCHAR type. CHAR(255), for instance, stores text strings 255 characters long, and VARCHAR(255) stores strings up to 255 characters in length. Notice that I said “255 characters” and “up to 255 characters.” CHAR is intended to store fixed-length strings, values that will always have the same number of characters across all rows in the table. The amount of space remains constant. VARCHAR stores variable-length strings, values that can have different lengths across the rows. The amount of space each value occupies is determined by the length of the string.

I'll highlight the difference between CHAR and VARCHAR using the string "Hello World". The string is 11 characters long, and it will occupy 11 bytes (plus an extra byte or two that MySQL needs to add for its own bookkeeping) if we store it in a VARCHAR(255) column. But with CHAR(255), the storage space is constant across all rows in the table. MySQL pads the string with 244 spaces. The padding is removed when we retrieve the string and the original 11-character "Hello World" string is returned, but all CHAR(255) strings occupy 255 bytes when they're stored.

Maximum Lengths: CHAR and VARCHAR have different maximum lengths. CHAR is allowed up to 255 characters and VARCHAR is allowed up to 65,535 characters. You probably won't want to use VARCHAR(65535) though. MySQL limits the size of a row to 25,535 bytes. Almost all of the row's columns contribute to the size (TEXT and BLOB are excluded), so you wouldn't have space left for the other columns. You find detailed information about the limits on row and column sizes in the **online documentation (https://dev.mysql.com/doc/refman/5.6/en/column-count-limit.html)**.

BINARY and VARBINARY behave similarly to CHAR and VARCHAR except they're used for binary strings. MySQL treats binary strings as a series of bytes, not characters, and doesn't take collation or character set into consideration when working with them. No special semantics are applied; any sorting or comparison operations performed are based on the ordinal value of each byte. MySQL uses the NULL byte 0x00 to pad/strip BINARY values.

TEXT and BLOB are variable-length data types for storing larger amounts of text. Neither performs padding/stripping, which makes them ideal for preserving the exact nature of the data. TEXT values are treated as character strings and BLOB values are treated as binary strings. The TEXT and BLOB columns (and their sized variants) are also excluded when MySQL calculates the length of a row, so consider using one of them when you need to store more than 65,535 bytes of data.

The non-binary string types CHAR, VARCHAR, and TEXT can be given the CHARACTER SET attribute to specify the data's encoding. A character set determines how the underlying bits and bytes are interpreted as human-readable characters. Common sets include ASCII (ascii), ISO 8859-1 (latin1), and UTF-8 (utf8). The default character set when none is specified is latin1.

```
CREATE TABLE charset_example (
id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
ascii_string VARCHAR(255) CHARACTER SET ascii NOT NULL,
latin1_string VARCHAR(255) CHARACTER SET latin1 NOT NULL,
utf8_string VARCHAR(255) CHARACTER SET utf8 NOT NULL,

PRIMARY KEY (id)
);
```

The ENUM and SET data types restrict a value to those allowed by a defined list. A column defined as ENUM('Alpha', 'Beta', 'Gamma') can only contain one of the strings listed in the definition, either "Alpha", "Beta", or "Gamma". SET holds strings with one or more comma-separated values from its list. SET('Alpha', 'Beta', 'Gamma') can hold values like "Alpha", "Alpha,Beta,Gamma", "Beta,Gamma", and so on.

Use ENUM with Caution: I have nothing against ENUM when it's use suits my data, but this seemingly innocent data type is not without controversy. Chris Komlenic's **blog post (http://komlenic.com/244)** "8 Reasons Why MySQL's ENUM Data Type Is Evil" is a good read on the subject.

The following table shows the storage requirements and the maximum length allowed for MySQL's string types:

Data Type	Storage Used (Bytes)	Maximum
CHAR(m)	m × maximum-size character in the character set	255 chars
VARCHAR(m)	up to 2 bytes + m × maximum-size character in the character set	65,535 chars
TEXT	size of string in bytes + 2	65,535 chars
TINYTEXT	size of string in bytes + 1	255 chars
MEDIUMTEXT	size of string in bytes + 3	16,777,215 chars
LONGTEXT	size of string in bytes + 4	4,294,967,295 chars
BINARY(m)	m	255 bytes
VARBINARY(m)	up to 2 bytes + m	65,535 bytes
BLOB	size of string in bytes + 2	65,535 bytes
TINYBLOB	size of string in bytes + 1	255 bytes
MEDIUMBLOB	size of string in bytes + 3	16,777,215 bytes
LONGBLOB	size of string in bytes + 4	4,294,967,295 bytes
ENUM	up to 2 bytes	65,535 values
SET	up to 8 bytes	64 members

Temporal Types

The data types DATETIME, TIMESTAMP, DATE, TIME, and YEAR are for working with date and time values. Both DATETIME and TIMESTAMP hold values containing date and time parts using the format 'YYYY-MM-DD HH:mm:ss' (YYYY is a four-digit year, MM is a two-digit month, DD a two-digit day, HH a two-digit hour, mm two-digit minutes, and ss two-digit seconds). For example, '2015-03-15 13:15:00' is 1:15 p.m. on the third of March, 2015. The DATE, TIME, and YEAR types all store their single respective part values. Besides 'HH:mm:ss' to represent a time of day—such as 13:15:00—TIME values may also be given like 'HHH:mm:ss' to represent an elapsed amount of time—such as 293:23:10, meaning 293 hours, 23 minutes, and 10 seconds.

MySQL can automatically initialize and update DATETIME and TIMESTAMP values with the current date and time whenever a row is added or updated. If the INSERT statement adds a new row to the

table but doesn't have a value for the TIMESTAMP column, MySQL will use the current date and time as the value. MySQL also updates a TIMESTAMP column's value when any of the values in its row are updated. While this behavior is automatic for TIMESTAMP, we can also apply it to DATETIME columns by specifying the attributes DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition.

Variations in Temporal Types: There are a few subtleties to the behavior of temporal data types across different releases of MySQL, especially with TIMESTAMP and YEAR. Review the documentation on **timestamp initialization (http://dev.mysql.com/doc/refman/5.6/en/timestamp-initialization.html)** and **two-digit years (http://dev.mysql.com/doc/refman/5.6/en/two-digit-years.html)** .

The following table shows the range and storage requirements for each of MySQL's temporal types:

Data Type	Storage Used (Bytes)	Minimum	Maximum
DATETIME	8	1000-01-01 00:00:00	9999-12-31 23:59:59
TIMESTAMP	4	1970-01-01 00:00:01 UTC	2038-01-19 03:14:07 UTC
DATE	3	1000-01-01	9999-12-31
TIME	3	-838:59:59	838:59:59
YEAR	1	1901	2155