Data Type Storage Requirements

The storage requirements for table data on disk depend on several factors. Different storage engines represent data types and store raw data differently. Table data might be compressed, either for a column or an entire row, complicating the calculation of storage requirements for a table or column.

Despite differences in storage layout on disk, the internal MySQL APIs that communicate and exchange information about table rows use a consistent data structure that applies across all storage engines.

This section includes guidelines and information for the storage requirements for each data type supported by MySQL, including the internal format and size for storage engines that use a fixed-size representation for data types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) or [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) columns, which contribute only 9 to 12 bytes toward this size. For [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) and [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this data in different ways, according to the method they use for handling the corresponding types.

NDB Table Storage Requirements

**Important**

[NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) tables use 4-byte alignment; all [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) data storage is done in multiples of 4 bytes. Thus, a column value that would typically take 15 bytes requires 16 bytes in an [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) table. For example, in [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) tables, the [TINYINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), [SMALLINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), [MEDIUMINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), and [INTEGER](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) ([INT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html)) column types each require 4 bytes storage per record due to the alignment factor.

Each BIT(***M***) column takes ***M*** bits of storage space. Although an individual [BIT](https://dev.mysql.com/doc/refman/8.0/en/bit-type.html) column is *not* 4-byte aligned, [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) reserves 4 bytes (32 bits) per row for the first 1-32 bits needed for BIT columns, then another 4 bytes for bits 33-64, and so on.

While a NULL itself does not require any storage space, [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) reserves 4 bytes per row if the table definition contains any columns allowing NULL, up to 32 NULL columns. (If an NDB Cluster table is defined with more than 32 NULL columns up to 64 NULL columns, then 8 bytes per row are reserved.)

Every table using the [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) storage engine requires a primary key; if you do not define a primary key, a “hidden” primary key is created by [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html). This hidden primary key consumes 31-35 bytes per table record.

You can use the [**ndb\_size.pl**](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster-programs-ndb-size-pl.html) Perl script to estimate [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) storage requirements. It connects to a current MySQL (not NDB Cluster) database and creates a report on how much space that database would require if it used the [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) storage engine.

Numeric Type Storage Requirements

| **Data Type** | **Storage Required** |
| --- | --- |
| [TINYINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) | 1 byte |
| [SMALLINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) | 2 bytes |
| [MEDIUMINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) | 3 bytes |
| [INT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html), [INTEGER](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) | 4 bytes |
| [BIGINT](https://dev.mysql.com/doc/refman/8.0/en/integer-types.html) | 8 bytes |
| FLOAT(***p***) | 4 bytes if 0 <= ***p*** <= 24, 8 bytes if 25 <= ***p*** <= 53 |
| [FLOAT](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) | 4 bytes |
| DOUBLE [PRECISION], [REAL](https://dev.mysql.com/doc/refman/8.0/en/floating-point-types.html) | 8 bytes |
| DECIMAL(***M***,***D***), NUMERIC(***M***,***D***) | Varies; see following discussion |
| BIT(***M***) | approximately (***M***+7)/8 bytes |

Values for [DECIMAL](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html) (and [NUMERIC](https://dev.mysql.com/doc/refman/8.0/en/fixed-point-types.html)) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

| **Leftover Digits** | **Number of Bytes** |
| --- | --- |
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 3 |
| 6 | 3 |
| 7 | 4 |
| 8 | 4 |

Date and Time Type Storage Requirements

For [TIME](https://dev.mysql.com/doc/refman/8.0/en/time.html), [DATETIME](https://dev.mysql.com/doc/refman/8.0/en/datetime.html), and [TIMESTAMP](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) columns, the storage required for tables created before MySQL 5.6.4 differs from tables created from 5.6.4 on. This is due to a change in 5.6.4 that permits these types to have a fractional part, which requires from 0 to 3 bytes.

| **Data Type** | **Storage Required Before MySQL 5.6.4** | **Storage Required as of MySQL 5.6.4** |
| --- | --- | --- |
| [YEAR](https://dev.mysql.com/doc/refman/8.0/en/year.html) | 1 byte | 1 byte |
| [DATE](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) | 3 bytes | 3 bytes |
| [TIME](https://dev.mysql.com/doc/refman/8.0/en/time.html) | 3 bytes | 3 bytes + fractional seconds storage |
| [DATETIME](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) | 8 bytes | 5 bytes + fractional seconds storage |
| [TIMESTAMP](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) | 4 bytes | 4 bytes + fractional seconds storage |

As of MySQL 5.6.4, storage for [YEAR](https://dev.mysql.com/doc/refman/8.0/en/year.html) and [DATE](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) remains unchanged. However, [TIME](https://dev.mysql.com/doc/refman/8.0/en/time.html), [DATETIME](https://dev.mysql.com/doc/refman/8.0/en/datetime.html), and [TIMESTAMP](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) are represented differently. [DATETIME](https://dev.mysql.com/doc/refman/8.0/en/datetime.html) is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

| **Fractional Seconds Precision** | **Storage Required** |
| --- | --- |
| 0 | 0 bytes |
| 1, 2 | 1 byte |
| 3, 4 | 2 bytes |
| 5, 6 | 3 bytes |

For example, [TIME(0)](https://dev.mysql.com/doc/refman/8.0/en/time.html), [TIME(2)](https://dev.mysql.com/doc/refman/8.0/en/time.html), [TIME(4)](https://dev.mysql.com/doc/refman/8.0/en/time.html), and [TIME(6)](https://dev.mysql.com/doc/refman/8.0/en/time.html) use 3, 4, 5, and 6 bytes, respectively. [TIME](https://dev.mysql.com/doc/refman/8.0/en/time.html) and [TIME(0)](https://dev.mysql.com/doc/refman/8.0/en/time.html) are equivalent and require the same storage.

String Type Storage Requirements

In the following table, ***M*** represents the declared column length in characters for nonbinary string types and bytes for binary string types. ***L*** represents the actual length in bytes of a given string value.

| **Data Type** | **Storage Required** |
| --- | --- |
| CHAR(***M***) | The compact family of InnoDB row formats optimize storage for variable-length character sets. See [COMPACT Row Format Storage Characteristics](https://dev.mysql.com/doc/refman/8.0/en/innodb-row-format.html#innodb-compact-row-format-characteristics). Otherwise, ***M*** × ***w*** bytes, <= ***M*** <= 255, where ***w*** is the number of bytes required for the maximum-length character in the character set. |
| BINARY(***M***) | ***M*** bytes, 0 <= ***M*** <= 255 |
| VARCHAR(***M***), VARBINARY(***M***) | ***L*** + 1 bytes if column values require 0 − 255 bytes, ***L*** + 2 bytes if values may require more than 255 bytes |
| [TINYBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [TINYTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) | ***L*** + 1 bytes, where ***L*** < 28 |
| [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) | ***L*** + 2 bytes, where ***L*** < 216 |
| [MEDIUMBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [MEDIUMTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) | ***L*** + 3 bytes, where ***L*** < 224 |
| [LONGBLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html), [LONGTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) | ***L*** + 4 bytes, where ***L*** < 232 |
| ENUM('***value1***','***value2***',...) | 1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum) |
| SET('***value1***','***value2***',...) | 1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum) |

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is ***L*** (the byte length of the string). For example, storage for a [MEDIUMTEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) value requires ***L*** bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular [CHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html), or [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) column value, you must take into account the character set used for that column and whether the value contains multibyte characters. In particular, when using a utf8 Unicode character set, you must keep in mind that not all characters use the same number of bytes. utf8mb3 and utf8mb4 character sets can require up to three and four bytes per character, respectively. For a breakdown of the storage used for different categories of utf8mb3 or utf8mb4 characters, see [Section 10.9, “Unicode Support”](https://dev.mysql.com/doc/refman/8.0/en/charset-unicode.html).

[VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html%22%20%5Co%20%2211.3.2%C2%A0The%20CHAR%20and%20VARCHAR%20Types), [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html), and the [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) and [TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) types are variable-length types. For each, the storage requirements depend on these factors:

* The actual length of the column value
* The column's maximum possible length
* The character set used for the column, because some character sets contain multibyte characters

For example, a VARCHAR(255) column can hold a string with a maximum length of 255 characters. Assuming that the column uses the latin1 character set (one byte per character), the actual storage required is the length of the string (***L***), plus one byte to record the length of the string. For the string 'abcd', ***L*** is 4 and the storage requirement is five bytes. If the same column is instead declared to use the ucs2 double-byte character set, the storage requirement is 10 bytes: The length of 'abcd' is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of *bytes* that can be stored in a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) or [VARBINARY](https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html) column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column that stores multibyte characters, the effective maximum number of *characters* is less. For example, utf8mb4 characters can require up to four bytes per character, so a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column that uses the utf8mb4 character set can be declared to be a maximum of 16,383 characters.

InnoDB encodes fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

The [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html) storage engine supports variable-width columns. This means that a [VARCHAR](https://dev.mysql.com/doc/refman/8.0/en/char.html) column in an NDB Cluster table requires the same amount of storage as would any other storage engine, with the exception that such values are 4-byte aligned. Thus, the string 'abcd' stored in a VARCHAR(50) column using the latin1 character set requires 8 bytes (rather than 5 bytes for the same column value in a MyISAM table).

[TEXT](https://dev.mysql.com/doc/refman/8.0/en/blob.html) and [BLOB](https://dev.mysql.com/doc/refman/8.0/en/blob.html) columns are implemented differently in [NDB](https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster.html); each row in a TEXT column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2000 bytes long. This means that the size of a TEXT column is 256 if ***size*** <= 256 (where ***size*** represents the size of the row); otherwise, the size is 256 + ***size*** + (2000 × (***size*** − 256) % 2000).

The size of an [ENUM](https://dev.mysql.com/doc/refman/8.0/en/enum.html) object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values.

The size of a [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) object is determined by the number of different set members. If the set size is ***N***, the object occupies (***N***+7)/8 bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A [SET](https://dev.mysql.com/doc/refman/8.0/en/set.html) can have a maximum of 64 members.

Spatial Type Storage Requirements

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. The [LENGTH()](https://dev.mysql.com/doc/refman/8.0/en/string-functions.html#function_length) function returns the space in bytes required for value storage.

JSON Storage **Requirements**

In general, the storage requirement for a [JSON](https://dev.mysql.com/doc/refman/8.0/en/json.html) column is approximately the same as for a LONGBLOB or LONGTEXT column; that is, the space consumed by a JSON document is roughly the same as it would be for the document's string representation stored in a column of one of these types. However, there is an overhead imposed by the binary encoding, including metadata and dictionaries needed for lookup, of the individual values stored in the JSON document. For example, a string stored in a JSON document requires 4 to 10 bytes additional storage, depending on the length of the string and the size of the object or array in which it is stored.

In addition, MySQL imposes a limit on the size of any JSON document stored in a JSON column such that it cannot be any larger than the value of [max\_allowed\_packet](https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html%22%20%5Cl%20%22sysvar_max_allowed_packet).