

Data Types

XINA has a fixed set of **data types** which apply to attributes and fields. They are intended to provide consistent behavior across MySQL, Java, and JavaScript data types.

Numeric Types

Type	Java	MySQL	JavaScript	Notes
<code>int(1)</code>	<code>byte</code>	<code>tinyint</code>	<code>number</code>	signed 1 byte integer, -2 ⁷ to 2 ⁷ -1
<code>int(2)</code>	<code>short</code>	<code>smallint</code>	<code>number</code>	signed 2 byte integer, -2 ¹⁵ to 2 ¹⁵ -1
<code>int(4)</code>	<code>int</code>	<code>int</code>	<code>number</code>	signed 4 byte integer, -2 ³¹ to 2 ³¹ -1
<code>int(8)</code>	<code>long</code>	<code>bigint</code>	<code>number</code>	signed 8 byte integer, -2 ⁶³ to 2 ⁶³ -1 ??
<code>float(4)</code>	<code>float</code>	<code>float</code>	<code>number</code>	IEEE 754 4 byte floating point
<code>float(8)</code>	<code>double</code>	<code>double</code>	<code>number</code>	IEEE 754 8 byte floating point
<code>boolean</code>	<code>boolean</code>	<code>tinyint</code>	<code>boolean</code>	MySQL treats <code>0</code> as <code>false</code> , non-zero as <code>true</code>

?? JavaScript number is 8 byte float, so only -2⁵³ to 2⁵³-1 is stored with exact precision

Character Types

Character data types offer two encoding options:

- **UTF-8** - default encoding, variable length, 1 to 4 bytes per character
- **ASCII** - subset of UTF-8, fixed length, 1 byte per character

Two SQL types:

- **char(n)** - data stored in the table, fastest search and index, uses fixed amount of space per row (n * max_bytes_per_character)
- **varchar(n)** - data stored in the table, fast search and index, uses variable amount of space per row (up to n * max_bytes_per_character)
- **text** - data stored outside the table, slower search and index, uses only as much space as needed

Two general types:

- **string** - text is **normalized** before insertion
 - leading and trailing whitespace is trimmed
 - all internal whitespace is reduced to a single space character
- **text** - text is inserted as provided

Note, all string operations are **case-insensitive** by default. This can be overridden with the [collate](#) expression by specifying a binary collation.

Type	Java	MySQL	JavaScript	Notes
<code>utf8string(n)</code>	<code>string</code>	<code>char(n)</code>	<code>string</code>	n up to 128, uses <code>n*4</code> bytes, normalized
<code>utf8vstring(n)</code>	<code>string</code>	<code>varchar(n)</code>	<code>string</code>	n up to 128, uses up to <code>n*4</code> bytes, normalized
<code>utf8string</code>	<code>string</code>	<code>mediumtext</code>	<code>string</code>	up to 2 ²⁴ bytes, normalized
<code>utf8text</code>	<code>string</code>	<code>mediumtext</code>	<code>string</code>	up to 2 ²⁴ bytes, not normalized
<code>asciistring(n)</code>	<code>string</code>	<code>char(n)</code>	<code>string</code>	n up to 256, uses n bytes, normalized
<code>asciivstring(n)</code>	<code>string</code>	<code>varchar(n)</code>	<code>string</code>	n up to 256, uses up to n bytes, normalized
<code>asciistring</code>	<code>string</code>	<code>mediumtext</code>	<code>string</code>	up to 2 ²⁴ bytes, normalized
<code>asciitext</code>	<code>string</code>	<code>mediumtext</code>	<code>string</code>	up to 2 ²⁴ bytes, not normalized

Temporal Types

Temporal data types store time data. There are two categories of temporal types:

- **instants** - identify specific moment in time, independent of time zone
 - stored numerically in the database in milliseconds
 - `datetime` and `date` use Unix epoch
 - `datetime` and `date` comparable in database
 - `date` + `time` = `datetime`
 - typically displayed in local time zone in front-end applications
- **timestamps** - identify specific formatted time without time zone consideration (thus `local`)
 - stored as ISO 8601 formatted `string` in database
 - `localdate` and `localdatetime` comparable in database
 - `CONCAT(localdate, 'T', localtime)` = `localdatetime`

Type	Java	MySQL	JavaScript	Notes
------	------	-------	------------	-------

<code>datetime</code>	<code>DateTime</code>	<code>bigint</code>	<code>date</code>	instant with millisecond precision, as Unix time
<code>date</code>	<code>XDate</code>	<code>bigint</code>	<code>date</code>	instant at start of date UTC, as Unix time
<code>time</code>	<code>LocalTime</code>	<code>int</code>	<code>number</code>	length of time up to 23:59:59.999, as millisecond count
<code>localdatetime</code>	<code>LocalDateTime</code>	<code>char(24)</code>	<code>string</code>	full timestamp without timezone, stored as <code>string</code>
<code>localdate</code>	<code>LocalDate</code>	<code>char(10)</code>	<code>string</code>	date without timezone, stored as <code>string</code>
<code>localtime</code>	<code>LocalTime</code>	<code>char(12)</code>	<code>string</code>	length of time up to 23:59:59.999, as <code>string</code>

JSON Types

JSON data types store JSON data directly in the database.

Type	Java	MySQL	JavaScript
<code>json</code>	<code>JsonValue</code>	<code>json</code>	<code>*</code>
<code>jsonarray</code>	<code>JsonArray</code>	<code>json</code>	<code>array</code>
<code>jsonobject</code>	<code>JsonObject</code>	<code>json</code>	<code>object</code>

Enum Types

Enum types map a series of discrete numeric integer values to text names. Though additional values may be added in the future, existing values will not change names or IDs.

`notification_level`

ID	Name	Notes
<code>0</code>	<code>none</code>	default level, no associated formatting
<code>1</code>	<code>success</code>	green
<code>2</code>	<code>info</code>	cyan
<code>3</code>	<code>notice</code>	yellow
<code>4</code>	<code>warning</code>	red
<code>5</code>	<code>primary</code>	blue, elevated over <code>none</code>
<code>6</code>	<code>secondary</code>	grey, below <code>none</code>

`notification_type`

ID	Name	Notes
0	post	
1	task	
2	request	request received
3	response	response to request received

post_level

ID	Name	Notes
0	none	default level, no associated formatting
1	success	green
2	info	cyan
3	notice	yellow
4	warning	red
5	primary	blue, elevated over none
6	secondary	grey, below none