String & binary data types

This topic describes the string/text data types, including binary strings, supported in Snowflake, along with the supported formats for string constants/literals.

Data types for text strings

Snowflake supports the following data types for text (that is, character) strings.

VARCHAR

VARCHAR values hold Unicode UTF-8 characters.

> **Note**
> In some systems outside of Snowflake, data types such as CHAR and VARCHAR store ASCII, while data types such as NCHAR and NVARCHAR store Unicode.
>
> In Snowflake, VARCHAR and all other string data types store Unicode UTF-8 characters. There is no difference with respect to Unicode handling between CHAR and NCHAR data types. Synonyms such as NCHAR are primarily for syntax compatibility when porting DDL commands to Snowflake.

When you declare a column of type VARCHAR, you can specify an optional parameter `(N)`, which is the maximum number of characters to store. For example:

```
CREATE TABLE t1 (v VARCHAR(16777216));
```

If no length is specified, the default is the maximum allowed length (16,777,216).

Although a VARCHAR value's maximum length is specified in ***characters***, a VARCHAR value is also limited to a maximum of 16,777,216 ***bytes*** (16 MB). The maximum number of Unicode characters that can be stored in a VARCHAR column is shown below:

Single-byte 16,777,216.

Multi-byte Between 8,388,608 (2 bytes per character) and 4,194,304 (4 bytes per character).

For example, if you declare a column as `VARCHAR(16777216)`, the column can hold a maximum of 8,388,608 2-byte Unicode characters, even though you specified a maximum length of 16777216.

When choosing the maximum length for a VARCHAR column, consider the following:

- **Storage:** A column consumes storage for only the amount of actual data stored. For example, a 1-character string in a `VARCHAR(16777216)` column only consumes a single character.
- **Performance:** There is no performance difference between using the full-length VARCHAR declaration `VARCHAR(16777216)` and a smaller length.

  In any relational database, SELECT statements in which a WHERE clause references VARCHAR columns or string

- **Collation:** When you specify a collation for a VARCHAR column, the number of characters that are allowed varies, depending on the number of bytes each character takes and the collation specification of the column.

  When comparing values in a collated column, Snowflake follows the Unicode Collation Algorithm (UCA). This algorithm affects the maximum number of characters allowed. Currently, around 1.5 million to 8 million characters are allowed in a VARCHAR column that is defined with a maximum size and a collation specification.

  As an example, the following table shows how the maximum number of characters can vary for a `VARCHAR(16777216)` column, depending on the number of bytes per character and the collation specification used:

Number of bytes per character	Collation specification	Maximum number of characters allowed (approximate)
1 byte	`en-ci` or `en-ci-pi-ai`	Around 7 million characters
1 byte	`en`	Around 4 million characters
2 byte	`en-ci-pi-ai`	Around 8 million characters
2 byte	`en-ci` or `en-ci-pi`	Around 2.7 million characters
2 byte	`en`	Around 1.5 million characters

CHAR, CHARACTER, NCHAR

Synonymous with VARCHAR, except that if the length is not specified, `CHAR(1)` is the default.

Snowflake currently deviates from common CHAR semantics in that strings shorter than the maximum length are not space-padded at the end.

STRING, TEXT, NVARCHAR, NVARCHAR2, CHAR VARYING, NCHAR VARYING

Synonymous with VARCHAR.

String examples in table columns

```
CREATE OR REPLACE TABLE test_text(
  vd VARCHAR,
  v50 VARCHAR(50),
  cd CHAR,
  c10 CHAR(10),
  sd STRING,
  s20 STRING(20),
  td TEXT,
  t30 TEXT(30));

DESC TABLE test_text;
```

```
+------+-------------------+--------+-------+---------+-------------+------------+------+------------
| name | type              | kind   | null? | default | primary key | unique key | check | expression
```

```
| C10  | VARCHAR(10)       | COLUMN | Y    | NULL    | N           | N          | NULL  | NULL
| SD   | VARCHAR(16777216) | COLUMN | Y    | NULL    | N           | N          | NULL  | NULL
| S20  | VARCHAR(20)       | COLUMN | Y    | NULL    | N           | N          | NULL  | NULL
| TD   | VARCHAR(16777216) | COLUMN | Y    | NULL    | N           | N          | NULL  | NULL
| T30  | VARCHAR(30)       | COLUMN | Y    | NULL    | N           | N          | NULL  | NULL
+------+-------------------+--------+------+---------+-------------+------------+-------+-----------
```

Data types for binary strings

Snowflake supports the following data types for binary strings.

BINARY

The maximum length is 8 MB (8,388,608 bytes). Unlike VARCHAR, the BINARY data type has no notion of Unicode characters, so the length is always measured in terms of bytes.

BINARY values are limited to 8 MB so that they fit within 16 MB when converted to hexadecimal strings, for example using `TO_CHAR(<binary_expression>, 'HEX')`.

If a length isn't specified, the default is the maximum length.

VARBINARY

Internal representation

The BINARY data type holds a sequence of 8-bit bytes.

When Snowflake displays BINARY data values, Snowflake often represents each byte as two hexadecimal characters. For example, the word `HELP` might be displayed as `48454C50`, where `48` is the hexadecimal equivalent of the ASCII (Unicode) letter `H`, `45` is the hexadecimal representation of the letter `E`, and so on.

For more information about entering and displaying BINARY data, see Binary input and output.

Binary examples in table columns

```
CREATE OR REPLACE TABLE test_binary(
  bd BINARY,
  b100 BINARY(100),
  vbd VARBINARY);

DESC TABLE test_binary;
```

```
+------+-----------------+--------+-------+---------+-------------+------------+-------+------------+-
| name | type            | kind   | null? | default | primary key | unique key | check | expression |
|------+-----------------+--------+-------+---------+-------------+------------+-------+------------+-
| BD   | BINARY(8388608) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       |
| B100 | BINARY(100)     | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       |
| VBD  | BINARY(8388608) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       |
```

String constants

Constants (also known as *literals*) refer to fixed data values. String constants in Snowflake must always be enclosed between delimiter characters. Snowflake supports using either of the following to delimit string constants:

- Single quotes
- Pairs of dollar signs

Single-quoted string constants

A string constant can be enclosed between single quote delimiters (for example, `'This is a string'`). To include a single quote character within a string constant, type two adjacent single quotes (for example, `''`).

For example:

```
SELECT 'Today''s sales projections', '-''''-';
```

```
+------------------------------+----------+
| 'TODAY''S SALES PROJECTIONS' | '-''''-' |
|------------------------------+----------|
| Today's sales projections    | -''-     |
+------------------------------+----------+
```

Two single quotes is ***not*** the same as the double quote character (`"`), which is used (as needed) for delimiting object identifiers. For more information, see Identifier requirements.

Escape sequences in single-quoted string constants

To include a single quote or other special characters (for example, newlines) in a single-quoted string constant, you must escape these characters by using *backslash escape sequences*. A backslash escape sequence is a sequence of characters that begins with a backslash (`\`).

Note

If the string contains many single quotes, backslashes, or other special characters, you can use a dollar-quoted string constant instead to avoid escaping these characters.

You can also use escape sequences to insert ASCII characters by specifying their code points (the numeric values that correspond to those characters) in octal or hexadecimal. For example, in ASCII, the code point for the space character is 32, which is 20 in hexadecimal. To specify a space, you can use the hexadecimal escape sequence `\x20`.

You can also use escape sequences to insert Unicode characters, for example `\u26c4`.

The following table lists the supported escape sequences in four categories: simple, octal, hexadecimal, and Unicode:

Escape sequence	Character represented
Simple escape sequences	

Escape sequence	Character represented
`\"`	A double quote (`"`) character.
`\\`	A backslash (`\`) character.
`\b`	A backspace character.
`\f`	A formfeed character.
`\n`	A newline (linefeed) character.
`\r`	A carriage return character.
`\t`	A tab character.
`\0`	An ASCII NUL character.
Octal escape sequences	
`\ooo`	ASCII character in octal notation (that is, where each `o` represents an octal digit).
Hexadecimal escape sequences	
`\xhh`	ASCII character in hexadecimal notation (that is, where each `h` represents a hexadecimal digit).
Unicode escape sequences	

As shown in the table above, if a string constant must include a backslash character (for example, `C:\` in a Windows path or `\d` in a regular expression), you must escape the backslash with a second backslash. For example, to include `\d` in a regular expression in a string constant, use `\\d`.

If a backslash is used in sequences other than the ones listed above, the backslash is ignored. For example, the sequence of characters `'\z'` is interpreted as `'z'`.

The following example demonstrates how to use backslash escape sequences. This includes examples of specifying:

- A tab character.
- A newline.
- A backslash.
- The octal and hexadecimal escape sequences for an exclamation mark (code point 33, which is `\041` in octal and `\x21` in hexadecimal).
- The Unicode escape sequence for a small image of a snowman.
- Something that is not a valid escape sequence.

```
SELECT $1, $2 FROM
  VALUES
    ('Tab','Hello\tWorld'),
    ('Newline','Hello\nWorld'),
    ('Backslash','C:\\user'),
    ('Octal','-\041-'),
    ('Hexadecimal','-\x21-'),
    ('Unicode','-\u26c4-'),
    ('Not an escape sequence', '\z');
```

```
+-------------------------+---------------+
| $1                      | $2            |
|-------------------------+---------------|
| Tab                     | Hello   World |
| Newline                 | Hello         |
|                         | World         |
| Backslash               | C:\user       |
| Octal                   | -!-           |
| Hexadecimal             | -!-           |
| Unicode                 | -⛄-          |
| Not an escape sequence | z             |
+-------------------------+---------------+
```

Dollar-quoted string constants

In some cases, you might need to specify a string constant that contains:

- Single quote characters.
- Backslash characters (for example, in a regular expression).
- Newline characters (for example, in the body of a stored procedure or function that you specify in CREATE PROCEDURE or CREATE FUNCTION).

In these cases, you can avoid escaping these characters by using a pair of dollar signs (`$$`) rather than a single quote (`'`) to delimit the beginning and ending of the string.

The following examples are equivalent ways of specifying string constants:

Example using single quote delimiters

```
'string with a \' character'
```

```
'regular expression with \\ characters:
\\d{2}-\\d{3}-\\d{4}'
```

```
'string with a newline\ncharacter'
```

Example using double dollar sign delimiters

```
$$string with a ' character$$
```

```
$$regular expression with \ characters:
\d{2}-\d{3}-\d{4}$$
```

```
$$string with a newline
character$$
```

The following example uses a dollar-quoted string constant that contains newlines and several escape sequences:

```
SELECT $1, $2 FROM VALUES (
  'row1',
  $$a
                                ' \ \t
                                \x21 z $ $$);
```

```
+------+-----------------------------------------------+
| $1   | $2                                            |
|------+-----------------------------------------------|
| row1 | a                                             |
|      |                                      ' \ \t    |
|      |                                      \x21 z $  |
+------+-----------------------------------------------+
```

In this example, the escape sequences are interpreted as their individual characters (for example, a backslash followed by a `t`), rather than as escape sequences.