

[Reference](#) > [SQL data types reference](#) > [String & binary](#) > [Binary input and output formats](#)

Binary input and output

Snowflake supports three binary formats or encoding schemes: hex, base64, and UTF-8.

Overview of supported binary formats

This section describes the supported binary formats.

hex (default) ¶

The “hex” format refers to the hexadecimal, or base 16, system. In this format, each byte is represented by two characters (digits from 0 to 9 and letters from A to F). When using hex to perform conversion:

From	To	Notes
Binary	String	hex uses uppercase letters.
String	Binary	hex is case-insensitive.

Hex is the default binary format

We use cookies to improve your experience on our site. By accepting, you agree to our privacy policy.

[Customize](#)

[Decline](#)

[Accept](#)

base64

The “base64” format encodes binary data (or string data) as printable ASCII characters (letters, digits, and punctuation marks or mathematical operators). (The base64 encoding scheme is defined in [RFC 4648](#).)

Base64-encoded data has the following advantages:

- Because base64-encoded data is pure ASCII text, it can be stored in systems that support ASCII character data but not BINARY data. For example, binary data that represents music (digital samples), or UTF data that represents Mandarin language characters, can be encoded as ASCII text and stored in systems that support only ASCII characters.
- Because base64-encoded data doesn't contain control characters (for example, end-of-transmission characters, tab characters), base64-encoded data can be transmitted and received without risk that control characters could be interpreted as commands rather than as data. Base64-encoded data is compatible with older modems and other telecommunications equipment that transmit and receive data one character at a time (without packet headers or protocols that indicate which parts of a packet are data and which are header or control information).



Base64-encoded data has the following disadvantages:

- Converting data back and forth between binary and printable ASCII representations consumes computation resources.
- Base64-encoded data requires approximately 1/3 more storage space than the original data.

The following sections provide technical details on base64 encoding.

Details of base64 encoding

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

- Uppercase letters (A - Z)
- Lowercase letters (a - z)
- Decimal digits (0 - 9)
- 
- 

In addition, the character  is used for padding if the length of the input that isn't an exact multiple of 3.

Because base64-encoded data doesn't contain whitespace characters (for example, blanks and line breaks), base64-encoded data can be mixed with whitespace if desired. For example, if the transmitter or receiver has a maximum limit on line length, the base64-encoded data can be split into individual lines by adding newline characters without corrupting the data. When using base64 to perform conversion:

From	To	Notes
Binary	String	Base64 does not insert any whitespace or line breaks.
String	Binary	Base64 ignores all whitespace and line breaks.

UTF-8

The UTF-8 format refers to the UTF-8 character encoding for Unicode.

UTF-8 is used for text-to-binary encoding. UTF-8 can't be used for binary-to-text encoding because not all possible

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

This format is convenient for performing one-to-one conversion between binary and string, for reinterpreting the underlying data as one type or the other rather than actually encoding and decoding.

Session parameters for binary values

There are two session parameters that determine how binary values are passed into and out of Snowflake:

- **BINARY_INPUT_FORMAT**: Specifies the format of VARCHAR input to functions that convert from VARCHAR to BINARY. It is used for:
 - Performing conversion to BINARY in the one-argument version of [TO_BINARY](#).
 - Loading data into Snowflake (if no file format option is specified; see below for details).

The parameter can be set to `HEX`, `BASE64`, or `UTF-8` (or `UTF8`). The parameter values are case-insensitive. The default is `HEX`.

- **BINARY_OUTPUT_FORMAT**: Specifies the format of VARCHAR output from functions that convert from BINARY to VARCHAR. It is used for:
 - Performing conversion to VARCHAR in the one-argument version of [TO_CHAR](#), [TO_VARCHAR](#).
 - Unloading data from Snowflake (if no file format option is specified; see below for details).
 - Displaying binary data in human-readable format (for example, in the Snowflake web interface) when no binary-to-varchar conversion was called explicitly.

The parameter can be set to `HEX` or `BASE64`. The parameter values are case-insensitive. The default is `HEX`.

Note

Because conversion from binary to string can fail with the UTF-8 format, **BINARY_OUTPUT_FORMAT** can't be set

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

The parameters can be set at the account, user, and session levels. Execute the [SHOW PARAMETERS](#) command to view the current parameter settings that apply to all operations in the current session.

File format option for loading/unloading binary values

Separate from the binary input and output session parameters, Snowflake provides the `BINARY_FORMAT` file format option, which can be used to explicitly control binary formatting when loading data into or unloading data from Snowflake tables.

This option can be set to `HEX`, `BASE64`, or `UTF-8` (values are case-insensitive). The option affects both data loading and unloading and, similar to other file format options, can be specified in the following ways:

- In a named file format, which can then be referenced in a named stage or directly in a `COPY` command.
- In a named stage, which can then be referenced directly in a `COPY` command.
- Directly in a `COPY` command.

Data loading

When used for data loading, `BINARY_FORMAT` specifies the format of binary values in your staged data files. This option **overrides** any value set for the `BINARY_INPUT_FORMAT` parameter in the session (see [Session parameters for binary values](#)).

If the option is set to `HEX` or `BASE64`, data loading can fail if the strings in the staged data file aren't valid hex or base64. In

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

Data unloading

When used in data unloading, the `BINARY_FORMAT` option specifies the format applied to binary values unloaded to the files in the specified stage. This option **overrides** any value set for the `BINARY_OUTPUT_FORMAT` parameter in the session (see [Session parameters for binary values](#)).

If the option is set to `UTF-8`, data unloading fails if any binary values in the table contain invalid UTF-8. In this case, Snowflake returns an error.

Example input/output

BINARY input/output can be confusing because “what you see isn’t necessarily what you get.”

Consider the following example:

```
CREATE OR REPLACE TABLE binary_table (v VARCHAR, b BINARY);

INSERT INTO binary_table (v, b)
  SELECT 'AB', TO_BINARY('AB');

SELECT v, b FROM binary_table;
```

```
+-----+-----+
| V     | B     |
+-----+-----+
```

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

The outputs for column `v` (VARCHAR) and column `b` appear to be identical. Yet the value for column `b` was converted to binary. Why does the value in column `b` look unchanged?

The answer is that the argument to `TO_BINARY` is treated as a sequence of hexadecimal digits (even though it is inside quotes and therefore looks like a string). The two characters you see are actually interpreted as a pair of hexadecimal digits that represent one byte of binary data, not two bytes of string data. (This wouldn't have worked if the input "string" had contained characters other than hexadecimal digits; the result would have been an error message similar to `"String '...' isn't a legal hex-encoded string"`.)

Also, when BINARY data is displayed, by default it is displayed as a sequence of hexadecimal digits. Thus the data went in as hexadecimal digits (not a string) and is displayed as hexadecimal digits, so it appears unchanged.

In fact, if the goal was to store the two-character string `AB`, then the code was wrong. The proper code would use the function `HEX_ENCODE` to convert the string to a sequence of hexadecimal digits (or use another "encode" function to convert to another format, such as base64) before storing the data. Examples of that are below.

Hexadecimal ("HEX") format example

One way to enter BINARY data is to encode it as a string of hexadecimal characters, as shown in the following example.

Start by creating a table with a BINARY column:

```
CREATE OR REPLACE TABLE demo_binary_hex (b BINARY);
```

```
INSERT INTO demo_binary_hex (b) VALUES ('TO_BINARY(HEX_ENCODE(''AB'', 'UTF8'))'); -- BINARY(2) = AB
```

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

```
INSERT INTO demo_binary_hex (b) SELECT TO_BINARY('HELP', 'HEX');
```

Here's the error message:

```
100115 (22000): The following string is not a legal hex-encoded value: 'HELP'
```

This time, explicitly convert the input to a string of hexadecimal digits before inserting it (this will succeed):

```
INSERT INTO demo_binary_hex (b) SELECT TO_BINARY(HEX_ENCODE('HELP'), 'HEX');
```

Now, retrieve the data:

```
SELECT TO_VARCHAR(b), HEX_DECODE_STRING(TO_VARCHAR(b)) FROM demo_binary_hex;
```

```
+-----+-----+
| TO_VARCHAR(B) | HEX_DECODE_STRING(TO_VARCHAR(B)) |
|-----+-----|
| 48454C50      | HELP                               |
+-----+-----+
```

As you can see, by default the output is shown as hexadecimal. To get back the original string, use the function

We use cookies to improve your experience on our site. By accepting, you agree to our privacy policy.


```

SELECT 'HELP',
       HEX_ENCODE('HELP'),
       b,
       HEX_DECODE_STRING(HEX_ENCODE('HELP')),
       TO_VARCHAR(b),
       HEX_DECODE_STRING(TO_VARCHAR(b))
FROM demo_binary_hex;

```

'HELP'	HEX_ENCODE('HELP')	B	HEX_DECODE_STRING(HEX_ENCODE('HELP'))	TO_VARCHAR(B)	HEX_DECODE_STRING(TO_VARCHAR(B))
HELP	48454C50	48454C50	HELP	48454C50	HELP

BASE64 format example

Before reading this section, consider reading [Hexadecimal \("HEX"\) format example](#). The basic concepts are similar, and [Hexadecimal \("HEX"\) format example](#) explains them in more detail.

Start by creating a table with a BINARY column:

```

CREATE OR REPLACE TABLE demo_binary_base64 (b BINARY);

```

We use cookies to improve your experience on our site. By accepting, you agree to our [privacy policy](#).

```
INSERT INTO demo_binary_base64 (b) SELECT TO_BINARY(BASE64_ENCODE('HELP'), 'BASE64');
```

Retrieve that row:

```
SELECT 'HELP',
       BASE64_ENCODE('HELP'),
       BASE64_DECODE_STRING(BASE64_ENCODE('HELP')),
       TO_VARCHAR(b, 'BASE64'),
       BASE64_DECODE_STRING(TO_VARCHAR(b, 'BASE64'))
FROM demo_binary_base64;
```

'HELP'	BASE64_ENCODE('HELP')	BASE64_DECODE_STRING(BASE64_ENCODE('HELP'))	TO_VARCHAR(b, 'BASE64')
HELP	SEVMUAA==	HELP	SEVMUAA==

UTF-8 format example

Start by creating a table with a BINARY column:

We use cookies to improve your experience on our site. By accepting, you agree to our privacy policy.

Insert a row:

```
INSERT INTO demo_binary_utf8 (b) SELECT TO_BINARY('HELP', 'UTF-8');
```

Retrieve that row:

```
SELECT 'HELP',  
       TO_VARCHAR(b, 'UTF-8')  
FROM demo_binary_utf8;
```

```
+-----+-----+  
| 'HELP' | TO_VARCHAR(B, 'UTF-8') |  
|-----+-----|  
| HELP   | HELP                     |  
+-----+-----+
```

We use cookies to improve your experience on our site. By accepting, you agree to our privacy policy.