Data type conversion

In many cases, a value of one data type can be converted to another data type. For example, an INTEGER value can be converted to a floating-point data type value. Converting a data type is called *casting*.

Explicit casting vs implicit casting

Users can explicitly convert a value from one data type to another. This is called *explicit casting*.

In some situations, Snowflake converts a value to another data type automatically. This is called *implicit casting* or *coercion*.

Explicit casting

Users can explicitly cast a value by using any of the following options:

- The CAST function.
- The `::` operator (called the *cast operator*).
- The appropriate SQL function (for example, TO_DOUBLE).

```
SELECT CAST('2022-04-01' AS DATE);

SELECT '2022-04-01'::DATE;

SELECT TO_DATE('2022-04-01');
```

Casting is allowed in most contexts in which a general expression is allowed, including the WHERE clause. For example:

```
SELECT date_column
  FROM log_table
  WHERE date_column >= '2022-04-01'::DATE;
```

Implicit casting (coercion)

Coercion occurs when a function (or operator) requires a data type that is different from, but compatible with, the arguments (or operands).

- Examples for functions or stored procedures:
  - The following code coerces the INTEGER value in column `my_integer_column` to FLOAT so that the value can be passed to the function `my_float_function()`, which expects a FLOAT:

    ```
    SELECT my_float_function(my_integer_column)
      FROM my_table;
    ```

- The following code coerces the INTEGER value `17` to VARCHAR so that the values can be concatenated by using the `||` operator:

```
SELECT 17 || '76';
```

  The result of this SELECT statement is the string `'1776'`.

- The following statement coerces the INTEGER value in column `my_integer_column` to FLOAT so that the value can be compared to the value `my_float_column` using the `<` comparison operator:

```
SELECT ...
  FROM my_table
  WHERE my_integer_column < my_float_column;
```

Not all contexts (for example, not all operators) support coercion.

Casting and precedence

When casting inside an expression, the code must take into account the precedence of the cast operator relative to other operators in the expression.

Consider the following example:

```
SELECT height * width::VARCHAR || ' square meters'
```

The cast operator has higher precedence than the arithmetic operator `*` (multiply), so the statement is interpreted as:

```
... height * (width::VARCHAR) ...
```

To cast the result of the expression `height * width`, use parentheses, as shown below:

```
SELECT (height * width)::VARCHAR || ' square meters'
  FROM dimensions;
```

As another example, consider the following statement:

```
SELECT -0.0::FLOAT::BOOLEAN;
```

You might expect this to be interpreted as:

```
SELECT (-0.0::FLOAT)::BOOLEAN;
```

Therefore, it would be expected to return FALSE (0 = FALSE, 1 = TRUE).

However, the cast operator has higher precedence than the unary minus (negation) operator, so the statement is interpreted as:

Therefore, the query results in an error message because the unary minus can't be applied to a BOOLEAN.

Data types that can be cast

The following table shows the valid data type conversions in Snowflake. The table also shows which coercions Snowflake can perform automatically.

> **Note**
> Internally, the CAST function and the `::` operator call the appropriate conversion function. For example, if you cast a NUMBER to a BOOLEAN, then Snowflake calls the TO_BOOLEAN function. The usage notes for each conversion function apply when the function is called indirectly using a cast, as well as when the function is called directly. For example, if you execute `CAST(my_decimal_column AS BOOLEAN)`, the rules for calling TO_BOOLEAN with a DECIMAL value apply. For convenience, the table includes links to the relevant conversion functions.

For details on conversions between semi-structured types and structured types, see Converting structured and semi-structured types.

Source data type	Target data type	Castable	Coercible	Conversion function	Notes
ARRAY					
	VARCHAR	✓	✗	TO_VARCHAR	
	VARIANT	✓	✓	TO_VARIANT	

Source data type	Target data type	Castable	Coercible	Conversion function	Notes
	VECTOR	✓	✓		Use explicit casting for conversion. For more information, see Vector conversion.
BINARY					
	VARCHAR	✓	✗	TO_VARCHAR	
	VARIANT	✓	✗	TO_VARIANT	
BOOLEAN					
	NUMBER	✓	✗	TO_NUMBER	
	VARCHAR	✓	✓	TO_VARCHAR	For example, from `TRUE` to `'true'`.
	VARIANT	✓	✓	TO_VARIANT	
DATE					
	TIMESTAMP	✓	✓	TO_TIMESTAMP	
	VARCHAR	✓	✓	TO_VARCHAR	
	VARIANT	✓	✗	TO_VARIANT	
FLOAT					

Source data type	Target data type	Castable	Coercible	Conversion function	Notes
	BOOLEAN	✓	✓	TO_BOOLEAN	For example, from `0.0` to `FALSE`.
	NUMBER[(p,s)]	✓	✓	TO_NUMBER	
	VARCHAR	✓	✓	TO_VARCHAR	
	VARIANT	✓	✓	TO_VARIANT	
GEOGRAPHY					
	VARIANT	✓	✗	TO_VARIANT	
GEOMETRY					
	VARIANT	✓	✗	TO_VARIANT	
NUMBER[(p,s)] *(Fixed-point numbers, including INTEGER)*					
	BOOLEAN	✓	✓	TO_BOOLEAN	For example, from `0` to `FALSE`.
	FLOAT	✓	✓	TO_DOUBLE	

Source data type	Target data type	Castable	Coercible	Conversion function	Notes
	VARIANT	✓	✓	TO_VARIANT	
OBJECT					
	ARRAY	✓	✗	TO_ARRAY	
	VARCHAR	✓	✗	TO_VARCHAR	
	VARIANT	✓	✓	TO_VARIANT	
TIME					
	VARCHAR	✓	✓	TO_VARCHAR	
	VARIANT	✓	✗	TO_VARIANT	
TIMESTAMP					
	DATE	✓	✓	TO_DATE , DATE	
	TIME	✓	✓	TO_TIME , TIME	
	VARCHAR	✓	✓	TO_VARCHAR	
	VARIANT	✓	✗	TO_VARIANT	

Source data type	Target data type	Castable	Coercible	Conversion function	Notes
	DATE	✓	✓	TO_DATE , DATE	
	FLOAT	✓	✓	TO_DOUBLE	For example, from `'12.34'` to `12.34`.
	NUMBER[(p,s)]	✓	✓	TO_NUMBER	For example, from `'12.34'` to `12.34`.
	TIME	✓	✓	TO_TIME , TIME	
	TIMESTAMP	✓	✓	TO_TIMESTAMP	
	VARIANT	✓	✗	TO_VARIANT	
VARIANT					
	ARRAY	✓	✓	TO_ARRAY	
	BOOLEAN	✓	✓	TO_BOOLEAN	For example, from a VARIANT containing `'false'` to `FALSE`.
	DATE	✓	✓	TO_DATE , DATE	
	FLOAT	✓	✓	TO_DOUBLE	
	GEOGRAPHY	✓	✗	TO_GEOGRAPHY	
	NUMBER[(p,s)]	✓	✓	TO_NUMBER	

Source data type	Target data type	Castable	Coercible	Conversion function	Notes
	TIME	✓	✓	TO_TIME , TIME	
	TIMESTAMP	✓	✓	TO_TIMESTAMP	
	VARCHAR	✓	✓	TO_VARCHAR	
	VECTOR	✓	✗		The VARIANT must contain an ARRAY of type FLOAT or INT.
VECTOR					
	ARRAY	✓	✓	TO_ARRAY	

[1]

NUMBER can be converted to TIMESTAMP because the values are treated as seconds since the beginning of the epoch (1970-01-01 00:00:00).

Note

For each listed data type (for example, FLOAT), the rules apply to all aliases for that data type (for example, the rules for FLOAT apply to DOUBLE, which is an alias for FLOAT).

Usage notes

- Conversion depends not only on the data type, but also the value, of the source. For example:
  - The VARCHAR value `'123'` can be converted to a numeric value, but the VARCHAR value `'xyz'` can't be converted to a numeric value.
  - The ability to cast a specific value of type VARIANT depends on the type of the data ***inside*** the VARIANT. For example, if the VARIANT contains a value of type TIME, then you can't cast the VARIANT value to a TIMESTAMP value, because you can't cast a TIME value to a TIMESTAMP value.
- If possible, pass in arguments of the same type. Avoid passing in arguments of different types.
- If one of the arguments is a number, the function coerces non-numeric string arguments (for example, `'a string'`) and string arguments that aren't constants to the type NUMBER(18,5).

  For numeric string arguments that aren't constants, if NUMBER(18,5) isn't sufficient to represent the numeric value, then cast the argument to a type that can represent the value.
- For some pairs of data types, conversion can result in loss of precision. For example:
  - Converting a FLOAT value to an INTEGER value rounds the value.
  - Converting a value from fixed-point numeric (for example, NUMBER(38, 0)) to floating point (for example, FLOAT) can result in rounding or truncation if the fixed-point number can't be precisely represented in a floating point number.
  - Converting a TIMESTAMP value to a DATE value removes the information about the time of day.
- Although Snowflake converts values in some situations where loss of precision can occur, Snowflake doesn't allow conversion in other situations where a loss of precision would occur. For example, Snowflake doesn't allow conversion when conversion would:
  - Truncate a VARCHAR value. For example, Snowflake doesn't cast VARCHAR(10) to VARCHAR(5), either implicitly or explicitly.
  - Result in the loss of digits other than the least significant digits. For example, the following fails:

In this example, the number `12.3` has two digits before the decimal point, but the data type `NUMBER(3,2)` has room for only one digit before the decimal point.

- When converting from a type with less precision to a type with more precision, conversion uses default values. For example, converting a DATE value to a TIMESTAMP_NTZ value causes the hour, minute, second, and fractional seconds to be set to `0`.
- When a FLOAT value is cast to a VARCHAR value, trailing zeros are omitted.

  For example, the following statements create a table and insert a row that contains a VARCHAR value, a FLOAT value, and a VARIANT value. The VARIANT value is constructed from JSON that contains a floating-point value represented with trailing zeros.

```
CREATE OR REPLACE TABLE convert_test_zeros (
  varchar1 VARCHAR,
  float1 FLOAT,
  variant1 VARIANT);

INSERT INTO convert_test_zeros SELECT
  '5.000',
  5.000,
  PARSE_JSON('{"Loan Number": 5.000}');
```

  The following SELECT statement explicitly casts both the FLOAT column and the FLOAT value inside the VARIANT column to VARCHAR. In each case, the VARCHAR contains no trailing zeros:

```
SELECT varchar1,
       float1::VARCHAR,
       variant1:"Loan Number"::VARCHAR
  FROM convert_test_zeros;
```

```
+----------+-----------------+---------------------------------+
| VARCHAR1 | FLOAT1::VARCHAR | VARIANT1:"LOAN NUMBER"::VARCHAR |
|----------+-----------------+---------------------------------|
| 5.000    | 5               | 5                               |
+----------+-----------------+---------------------------------+
```

- Some operations can return different data types, depending upon a conditional expression. For example, the following IFNULL calls return slightly different data types depending upon the input values:

```
SELECT SYSTEM$TYPEOF(IFNULL(12.3, 0)),
       SYSTEM$TYPEOF(IFNULL(NULL, 0));
```

```
+--------------------------------+--------------------------------+
| SYSTEM$TYPEOF(IFNULL(12.3, 0)) | SYSTEM$TYPEOF(IFNULL(NULL, 0)) |
|--------------------------------+--------------------------------|
| NUMBER(3,1)[SB1]               | NUMBER(1,0)[SB1]               |
+--------------------------------+--------------------------------+
```

  If the expression has more than one possible data type, then Snowflake chooses the data type based on the actual result. For more information about precision and scale in calculations, see Scale and precision in arithmetic operations. If the query generates more than one result (for example, multiple rows of results), then Snowflake chooses a data type that is capable of holding each of the individual results.

- Some application programs, such as SnowSQL, and some graphical user interfaces, such as the Classic Console, apply their own conversion and formatting rules when displaying data. For example, SnowSQL displays BINARY values as a string that contains only hexadecimal digits; that string is generated by implicitly calling a conversion