Numeric data types

This topic describes the numeric data types supported in Snowflake, along with the supported formats for numeric constants/literals.

Data types for fixed-point numbers

Snowflake supports the following data types for fixed-point numbers.

NUMBER ¶

Numbers up to 38 digits, with an optional precision and scale:

Precision	Total number of digits allowed.
Scale	Number of digits allowed to the right of the decimal point.

By default, precision is 38, and scale is 0 (that is, NUMBER(38, 0)). Precision limits the range of values that can be inserted into (or cast to) columns of a given type. For example, the value `999` fits into NUMBER(38,0) but not into NUMBER(2,0).

Because precision is the total number of digits allowed, you can't load a value into a NUMBER column if the number of

The maximum scale (number of digits to the right of the decimal point) is 37. Numbers that have fewer than 38 significant digits, but whose least significant digit is past the 37th decimal place, for example 0.0000000000000000000000000000000000000012 (1.2e-39), can't be represented without losing some digits of precision.

Note

If data is converted to another data type with lower precision, then back to the higher-precision data type, the data can lose precision. For example, precision is lost if you convert a NUMBER(38,37) value to a DOUBLE value (which has a precision of approximately 15 decimal digits), and then back to NUMBER.

Snowflake also supports the FLOAT data type, which allows a wider range of values, although with less precision.

DECIMAL , DEC , NUMERIC

Synonymous with NUMBER.

INT , INTEGER , BIGINT , SMALLINT , TINYINT , BYTEINT

Synonymous with NUMBER, except that precision and scale can't be specified (that is, it always defaults to NUMBER(38, 0)). Therefore, for all INTEGER data types, the range of values is all integer values from -99999999999999999999999999999999999999 to +99999999999999999999999999999999999999 (inclusive).

The various names (TINYINT, BYTEINT, and so on) are to simplify porting from other systems and to suggest the expected

Impact of precision and scale on storage size

Precision (total number of digits) doesn't affect storage. The storage requirements for the same number in columns with different precisions, such as NUMBER(2,0) and NUMBER(38,0), are the same. For each micro-partition, Snowflake determines the minimum and maximum values for a given column and uses that information to determine the storage size for all values for that column in the partition. For example:

- If a column contains only values between `-128` and `+127`, then each of the values consumes 1 byte (uncompressed).
- If the largest value in the column is `10000000`, then each of the values consumes 4 bytes (uncompressed).

However, scale (the number of digits following the decimal point) affects storage. For example, the same value stored in a column of type NUMBER(10,5) consumes more space than NUMBER(5,0). Also, processing values with a larger scale might be slightly slower and consume more memory.

To save space, Snowflake compresses values before writing them to storage. The amount of compression depends on the data values and other factors.

Examples of fixed-point data types in a table

The following statement creates a table with columns of various fixed-point data types:

```
CREATE OR REPLACE TABLE test_fixed(
  num0 NUMBER,
  num10 NUMBER(10,1),
  dec20 DECIMAL(20,2),
DESC TABLE test_fixed;
```

```
+-----------+-------------+--------+-------+---------+-------------+------------+-------+-----------
| name      | type        | kind   | null? | default | primary key | unique key | check | expression
|-----------+-------------+--------+-------+---------+-------------+------------+-------+-----------
| NUM0      | NUMBER(38,0) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL
| NUM10     | NUMBER(10,1) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL
| DEC20     | NUMBER(20,2) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL
| NUMERIC30 | NUMBER(30,3) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL
| INT1      | NUMBER(38,0) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL
| INT2      | NUMBER(38,0) | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL
+-----------+-------------+--------+-------+---------+-------------+------------+-------+-----------
```

Data types for floating-point numbers

Snowflake supports the following data types for floating-point numbers.

FLOAT , FLOAT4 , FLOAT8

The names FLOAT, FLOAT4, and FLOAT8 are for compatibility with other systems. Snowflake treats all three as 64-bit floating-point numbers.

Snowflake uses double-precision (64 bit) IEEE 754 floating-point numbers.

Precision is approximately 15 digits. For example, for integers, the range is from -9007199254740991 to +9007199254740991 ($-2^{53} + 1$ to $+2^{53} - 1$). Floating-point values can range from approximately 10^{-308} to 10^{+308}. (Snowflake can represent more extreme values between approximately 10^{-324} and 10^{-308} with less precision.) For more details, see the Wikipedia article on double-precision numbers.

Snowflake supports the fixed-point data type NUMBER, which allows greater precision, although a smaller range of exponents.

Special values

Snowflake supports the following special values for FLOAT:

- `'NaN'` (not a number)
- `'inf'` (infinity)
- `'-inf'` (negative infinity)

The symbols `'NaN'`, `'inf'`, and `'-inf'` must be in single quotes, and are case-insensitive.

Comparison semantics for `'NaN'` differ from the IEEE 754 standard in the following ways:

Condition	Snowflake	IEEE 754	Comment
`'NaN' = 'NaN'`	`TRUE`	`FALSE`	In Snowflake, `'NaN'` values are all equal.

Condition	Snowflake	IEEE 754	Comment
infinity (other than NaN itself).			including infinity.

Rounding errors

Floating point operations can have small rounding errors in the least significant digits. Rounding errors can occur in any type of floating-point processing, including trigonometric, statistical, and geospatial functions.

The following are considerations for rounding errors:

- Errors can vary each time the query is executed.
- Errors can be larger when operands have different precision or scale.
- Errors can accumulate, especially when aggregate functions (for example, SUM or AVG) process large numbers of rows. Casting to a fixed-point data type before aggregating can reduce or eliminate these errors.
- Rounding errors can occur not only when working with SQL, but also when working with other code (for example, Java, JavaScript, or Python) that runs inside Snowflake (for example, in UDFs and stored procedures).
- When comparing two floating-point numbers, Snowflake recommends comparing for approximate equality rather than exact equality.

DOUBLE , DOUBLE PRECISION , REAL

Svnonvmous with FLOAT.

Examples of floating-point data types in a table

The following statement creates a table with columns of various floating-point data types:

```
CREATE OR REPLACE TABLE test_float(
  double1 DOUBLE,
  float1 FLOAT,
  dp1 DOUBLE PRECISION,
  real1 REAL);

DESC TABLE test_float;
```

```
+---------+-------+--------+-------+---------+-------------+------------+-------+------------+--------
| name    | type  | kind   | null? | default | primary key | unique key | check | expression | comment
|---------+-------+--------+-------+---------+-------------+------------+-------+------------+--------
| DOUBLE1 | FLOAT | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       | NULL
| FLOAT1  | FLOAT | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       | NULL
| DP1     | FLOAT | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       | NULL
| REAL1   | FLOAT | COLUMN | Y     | NULL    | N           | N          | NULL  | NULL       | NULL
+---------+-------+--------+-------+---------+-------------+------------+-------+------------+-------
```

Note

The DESC TABLE command's `type` column displays the data type FLOAT not only for FLOAT, but also for synonyms of FLOAT (for example, DOUBLE, DOUBLE PRECISION, and REAL).

Numeric constants

The term ***constants*** (also known as ***literals***) refers to fixed data values. The following formats are supported for numeric constants:

```
[+-][digits][.digits][e[+-]digits]
```

Where:

- `+` or `-` indicates a positive or negative value. The default is positive.
- `digits` is one or more digits from 0 to 9.
- `e` (or `E`) indicates an exponent in scientific notation. At least one digit must follow the exponent marker if present.

The following numbers are all examples of supported numeric constants:

```
15
+1.34
0.2
15e-03
1.234E2
1.234E+2
-1
```