

# Graphisoft SQL Language Reference

Graphisoft SQL is a dialect of the SQL language defined by the SQL92 Entry Level standard. This chapter contains information about the commands, types, and functions you will need to know in order to use Graphisoft SQL efficiently and effectively. This chapter also serves as a reference to the functionality of Graphisoft SQL.

- [Literals](#)
- [Table, Column, and Alias Names](#)
- [Reserved Words](#)
- [Data Types](#)
- [Data Type Precedence](#)
- [Data Type Conversion](#)
- [Expressions](#)
- [Statements](#)
- [Graphisoft SQL Grammar](#)

## Literals

This section describes how to write string and number literals in Graphisoft SQL.

### Strings

A string is a sequence of characters, surrounded by single quote (') characters. Example:

```
'This is a string literal'
```

### Numbers

Integers are represented as a sequence of digits. Floats use '.' as a decimal separator. Either type of number may be preceded by '-' to indicate negative value.

Examples of valid integers:

```
1256  
0  
-55
```

Examples of valid floating-point numbers:

```
123.45  
12.3455e+10  
121.0
```

An integer may be used in a floating-point context; it's interpreted as the equivalent floating-point number.

## NULL Values

The NULL value means "no data" and is different from values such as 0 for numeric types or the empty string of string types.

## Table, Column, and Alias Names

Table, column, and alias names all follow the same identifier syntax rules. There are two kinds of identifiers:

- regular identifier
- delimited identifier

### Regular Identifiers

The regular identifier starts with a regular identifier start character followed by any number of regular identifier part characters. The regular identifier start characters are the latin small and caps letters and the underscore character. The regular identifier part characters are the latin small and caps letters, the underscore characters and the decimal digits. The length of the regular identifier is not limited.

Examples:

```
ID
CUSTOMER_ID
Order_Type21
```

### Delimited Identifiers

The delimited identifiers are delimited by two double quote (") or two square ([]) characters. The body of the delimited identifier consists of any number of latin small or caps letters, decimal digits, underscores or space characters. The length of the delimited identifier is not limited. Examples:

```
"Order Identifier"
"Customer order"
[My orders]
```

Note that if the identifier is a restricted word or contains special characters (space) you must always quote it with double quotes when you use it:

```
SELECT "FROM" FROM ARRIVALS WHERE ARRIVALS."FROM" = 'Budapest'
```

### Case Sensitivity in Names

Table, column, and alias names are case-insensitive in Graphisoft SQL as defined in the SQL92 standard. For identifiers in mixed case, every lower-case letter is replaced by its equivalent upper-case letter.

## Reserved Words

The reserved words (keywords) of Graphisoft SQL are case insensitive.

The following words are reserved in Graphisoft SQL:

ALL, AND, AS, ASC, BETWEEN, BIGINT, BINARY, BLOB, CHAR, CHARACTER, CLOB, CROSS, DESC, DISTINCT, DOUBLE, ESCAPE, FLAT, FLOAT, FROM, IN, INNER, INT, INTEGER, IS, JOIN, LEFT, LIKE, LONG, LONGVARBINARY, LONGVARCHAR, NOT, NULL, ON, OR, PLIKE, PRECISION, REAL, RIGHT, ROTATED\_FLAT, SELECT, SMALLINT, TEXT, TINYINT, VARBINARY, VARCHAR, VARYING, WHERE

## Data Types

Graphisoft SQL supports a number of data types. These may be grouped into four categories:

- numeric types
- string (character) types
- binary types
- variant types

This section gives an overview of available types and their storage requirements, then provides a more detailed description:

### BOOL

Boolean type. Storage size is 1 byte.

### TINYINT

A very small unsigned integer type. The range of the type is 0 to 255. Storage size is 1 byte.

### SMALLINT

A small signed integer type. The range of the type is -32768 to 32767. Storage size is 2 bytes.

### INT

A normal-size signed integer type. The range of the type is -2147483648 to 2147483647. Storage size is 4 bytes. Synonyms: **INTEGER**.

### BIGINT

A large signed integer type. The range of the type is -9223372036854775808 to 9223372036854775807. Storage size is 8 bytes.

### FLOAT (precision)

A floating-point number type. The range of type is -1.79e+308 to 1.79e+308. **precision** must be a value from 1 to 53. If **precision** is less than or equal to 24, then the float type represents a single-precision floating-point type; otherwise it represents a double-precision floating-point type.

**FLOAT (precision)** has the same range as the corresponding **REAL** and **DOUBLE** types, but the display size and number of decimals are undefined. Storage size is 4 or 8 bytes according to the specified binary precision.

### REAL

A small (single-precision) floating-point number type. The range of the type is -3.40e+38 to 3.40e+38. Storage size is 4 bytes.

### DOUBLE

A normal-size (double-precision) floating-point number type. The range of the type is -1.79e+308 to 1.79e+308. Storage size is 8 bytes. Synonyms: **DOUBLE PRECISION**.

### CHAR (length)

A fixed-length non-Unicode character string type with length of **length** characters. **length** must be

a value from 1 to 8,000. The data entered can be 0 characters in length. The **CHAR** values are sorted and compared in case-sensitive fashion according to the default character set. Synonyms: **CHARACTER (length)**.

#### **VARCHAR (maxlength)**

A variable-length non-Unicode character string type with maximum length **maxlength**. **maxlength** must be a value from 1 to 8,000. The storage size is the actual length in bytes of the data entered, not **maxlength** bytes. The data entered can be 0 characters in length. The **VARCHAR** values are sorted and compared in case-sensitive fashion according to the default character set. Synonyms: **CHAR VARYING (maxlength)**, **CHARACTER VARYING (maxlength)**.

#### **CLOB (maxlength)**

A large variable-length non-Unicode character string type with maximum length **maxlength**. **maxlength** must be a value from 1 to 2,147,483,647. The storage size is the actual length in bytes of the data entered, not **maxlength** bytes. The data entered can be 0 characters in length. The **VARCHAR** values are sorted and compared in case-sensitive fashion according to the default character set. Synonyms: **CHAR LARGE OBJECT (maxlength)**, **CHARACTER LARGE OBJECT (maxlength)**.

#### **TEXT**

A large variable-length non-Unicode character string type with maximum length 2,147,483,647. The storage size is the actual length in bytes of the data entered. The data entered can be 0 characters in length. The **TEXT** values are sorted and compared in case-sensitive fashion according to the default character set. Synonyms: **LONGVARCHAR**.

#### **NCHAR (length)**

A fixed-length Unicode character string type with length of **length** characters. **length** must be a value from 1 to 4,000. The data entered can be 0 characters in length. The **NCHAR** values are sorted and compared in case-sensitive fashion. Synonyms: **NATIONAL CHAR (length)**, **NATIONAL CHARACTER (length)**.

#### **NVARCHAR (maxlength)**

A variable-length Unicode character string type with maximum length **maxlength**. **maxlength** must be a value from 1 to 4,000. Storage size, in bytes, is two times the number of characters entered. The data entered can be 0 characters in length. The **NVARCHAR** values are sorted and compared in case-sensitive fashion. Synonyms: **NATIONAL CHAR VARYING (maxlength)**, **NATIONAL CHARACTER VARYING (maxlength)**.

#### **NCLOB (maxlength)**

A large variable-length Unicode character string type with maximum length **maxlength**. **maxlength** must be a value from 1 to 2,147,483,647. Storage size, in bytes, is two times the number of characters entered. The data entered can be 0 characters in length. The **NCLOB** values are sorted and compared in case-sensitive fashion. Synonyms: **NCHAR LARGE OBJECT (maxlength)**.

#### **NTEXT**

A large variable-length Unicode character string type with maximum length 2,147,483,647. Storage size, in bytes, is two times the number of characters entered. The data entered can be 0 characters in length. The **NTEXT** values are sorted and compared in case-sensitive fashion. Synonyms: **NLONGVARCHAR**.

#### **BINARY (length)**

A fixed-length binary data type with length of **length** bytes. **length** must be a value from 1 through 8,000. Storage size is **length**+4 bytes.

#### **VARBINARY (maxlength)**

A variable-length binary data type with maximum length of **maxlength** bytes. **maxlength** must be a value from 1 through 8,000. The storage size is the actual length in bytes of the data entered, not **maxlength** bytes.

**BLOB (maxlength)**

A large variable-length binary data type with maximum length of **maxlength** bytes. **maxlength** must be a value from 1 through 2,147,483,647. The storage size is the actual length in bytes of the data entered, not **maxlength** bytes. Synonyms: **BINARY LARGE OBJECT (maxlength)**.

**LONGVARBINARY**

A large variable-length binary data type with maximum length of 2,147,483,647 bytes. The storage size is the actual length in bytes of the data entered.

**IMAGE**

A binary image data type. The image values consist of two associated subvalues: the raw image data and the MIME type of the image.

**GDLVARIANT**

A data type that stores values of **INTEGER**, **DOUBLE**, **VARCHAR (8000)** or two-dimensional array of values of **INTEGER**, **DOUBLE**, **VARCHAR (8000)** data types.

## Data Type Conversion

There are two kinds of data type conversion. Implicit conversions are those conversions that occur without specifying the CAST function. Explicit conversions are those conversions that require the CAST function to be specified. The following chart shows the implicit/explicit conversions allowed for the data types of Graphisoft SQL.

To:\nFrom:	BINARY	VARBINARY	BLOB	LONGVARBINARY	CHAR	VARCHAR	CLOB	TEXT	NCHAR	NVARCHAR	NCLOB	NTEXT	BOOL	TINYINT	SMALLINT	INT	BIGINT	REAL	FLOAT	DOUBLE	IMAGE	GDLVARIANT
BINARY		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
VARBINARY	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
BLOB	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
LONGVARBINARY	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗
CHAR	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
VARCHAR	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
CLOB	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
TEXT	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
NCHAR	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
NVARCHAR	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
NCLOB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
NTEXT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
BOOL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✗	✗
TINYINT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✗	✗
SMALLINT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✗	✗
INT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✗	✗
BIGINT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✗	✗
REAL	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✗	✗
FLOAT	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✗	✗
DOUBLE	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✗	✗
IMAGE	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗		✗
GDLVARIANT	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	

- ✓ Implicit conversion
- ✓ Explicit conversion
- ✗ Conversion not allowed

## Expressions

This section gives a brief overview of the expressions that can be used in the select list and in the search condition (where clause) part of the SQL statements.

An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for the operators and functions involved in that expression.

- [Non-Type-Specific Operators](#)
- [String Operators](#)
- [Arithmetic Operators](#)
- [Image Functions](#)
- [Data Type Conversion Functions](#)

## Non-Type-Specific Operators

## Parentheses

Use parentheses to force the order of evaluation in an expression. Examples:

```
1*2+3  
-->5
```

```
1*(2+3)  
-->7
```

## Comparison Operators

Comparison operations result in a value of TRUE, FALSE or NULL. These operators can be used on numbers, strings, binaries. The different types are automatically converted to each other according to the implicit conversion rules. Available conversion operators:

- [= \(equals\)](#)
- [<> \(not equal to\)](#)
- [<= \(less than or equal to\)](#)
- [< \(less than\)](#)
- [>= \(greater than or equal to\)](#)
- [> \(greater than\)](#)
- [IS \[NOT\] NULL](#)
- [BETWEEN](#)
- [IN](#)

## Logical Operators

All logical operators are evaluated as TRUE or FALSE. Available logical operators:

- [NOT](#)
- [AND](#)
- [OR](#)

## < (Less Than)

Compares two expressions (a comparison operator). When you compare nonnull expressions, the result is TRUE if the left operand has a lower value than the right operand; otherwise, the result is FALSE. If either or both operands are NULL the result is FALSE if one of the operands is NULL, and TRUE if both operands are NULL.

## Syntax

```
<expression> < <expression>
```

## Arguments

expression

Any valid Graphisoft SQL expression. Both expressions must have implicitly convertible data

types. The conversion depends on the rules of data type precedence.

## Result Types

BOOL

# <= (Less Than or Equal To)

Compares two expressions (a comparison operator). When you compare nonnull expressions, the result is TRUE if the left operand has a lower or equal value than the right operand; otherwise, the result is FALSE. If either or both operands are NULL the result is FALSE if one of the operands is NULL, and TRUE if both operands are NULL.

## Syntax

```
<expression> <= <expression>
```

## Arguments

expression

Any valid Graphisoft SQL expression. Both expressions must have implicitly convertible data types. The conversion depends on the rules of data type precedence.

## Result Types

BOOL

# <> (Not Equal To)

Compares two expressions (a comparison operator). When you compare nonnull expressions, the result is TRUE if the left operand is not equal to the right operand; otherwise, the result is FALSE. If either or both operands are NULL the result is FALSE if one of the operands is NULL, and TRUE if both operands are NULL.

## Syntax

```
<expression> <> <expression>
```

## Arguments

expression

Any valid Graphisoft SQL expression. Both expressions must have implicitly convertible data types. The conversion depends on the rules of data type precedence.

## Result Types

BOOL



## = (Equals)

Compares two expressions (a comparison operator). When you compare nonnull expressions, the result is TRUE if both operands are equal; otherwise, the result is FALSE. If either or both operands are NULL the result is FALSE if one of the operands is NULL, and TRUE if both operands are NULL.

### Syntax

```
<expression> = <expression>
```

### Arguments

expression

Any valid Graphisoft SQL expression. Both expressions must have implicitly convertible data types. The conversion depends on the rules of data type precedence.

### Result Types

BOOL

## > (Greater Than)

Compares two expressions (a comparison operator). When you compare nonnull expressions, the result is TRUE if the left operand has a higher value than the right operand; otherwise, the result is FALSE. If either or both operands are NULL the result is FALSE if one of the operands is NULL, and TRUE if both operands are NULL.

### Syntax

```
<expression> > <expression>
```

### Arguments

expression

Any valid Graphisoft SQL expression. Both expressions must have implicitly convertible data types. The conversion depends on the rules of data type precedence.

### Result Types

BOOL

## >= (Greater Than or Equal To)

Compares two expressions (a comparison operator). When you compare nonnull expressions, the result

is TRUE if the left operand has a higher or equal value than the right operand; otherwise, the result is FALSE. If either or both operands are NULL the result is FALSE if one of the operands is NULL, and TRUE if both operands are NULL.

## Syntax

```
<expression> >= <expression>
```

## Arguments

expression

Any valid Graphisoft SQL expression. Both expressions must have implicitly convertible data types. The conversion depends on the rules of data type precedence.

## Result Types

BOOL

# AND

Combines two boolean expressions and returns TRUE when both expressions are TRUE. When more than one logical operator is used in a statement, AND operators are evaluated first. You can change the order of evaluation by using parentheses.

## Syntax

```
<boolean expression> AND <boolean expression>
```

## Arguments

boolean expression

Any valid Graphisoft SQL boolean expression.

## Result Types

BOOL

# BETWEEN

Specifies a range to test. BETWEEN returns TRUE if the value of **<test expression>** is greater than or equal to the value of **<begin expression>** and less than or equal to the value of **<end expression>**. NOT BETWEEN returns TRUE if the value of **<test expression>** is less than the value of **<begin expression>** or greater than the value of **<end expression>**.

## Syntax

```
<test expression> [ NOT ] BETWEEN <begin expression> AND <end expression>
```

## Arguments

test expression

The expression to test for in the range defined by **<begin expression>** and **<end expression>**.

**<test expression>** must be the same data type as both **<begin expression>** and **<end expression>**.

begin expression

Any valid Graphisoft SQL expression. **<begin expression>** must be the same data type as both

**<test expression>** and **<end expression>**.

end expression

Any valid Graphisoft SQL expression. **<end expression>** must be the same data type as both **<test**

**expression>** and **<begin expression>**.

## Result Types

BOOL

# IN

Determines if a given value matches any value in a subquery or a list. If the value of **<test expression>** is equal to any **<expression>** from the comma-separated list, the result value is TRUE. Otherwise, the result value is FALSE. Using NOT IN negates the returned value.

## Syntax

```
<test expression> [ NOT ] IN ( <expression> ... )
```

## Arguments

test expression

Any valid Graphisoft SQL numeric expression.

expression...

A list of expressions to test for a match. All expressions must be of the same type as **<test expression>**.

## Result Types

BOOL

# IS [ NOT ] NULL

Determines whether a given expression is NULL. If the value of **<expression>** is NULL, IS NULL returns TRUE; otherwise, it returns FALSE. If the value of **<expression>** is NULL, IS NOT NULL returns FALSE; otherwise, it returns TRUE.

## Syntax

```
<expression> IS [ NOT ] NULL
```

## Arguments

expression

Any valid Graphisoft SQL expression.

## Result Types

BOOL

# NOT

Negates a boolean input.

## Syntax

```
NOT <boolean expression>
```

## Arguments

boolean expression

Any valid Graphisoft SQL boolean expression.

## Result Types

BOOL

# OR

Combines two conditions. When more than one logical operator is used in a statement, OR operators are evaluated after AND operators. However, you can change the order of evaluation by using parentheses.

## Syntax

```
<boolean expression> OR <boolean expression>
```

## Arguments

boolean expression

Any valid Graphisoft SQL boolean expression.

## Result Types

BOOL

# String Operators

Operators that works on character string operands. Available string operators:

- [+ \(string concatenation\)](#)
- [LIKE](#)

## + (String Concatenation)

An operator in a string expression that concatenates two or more character string into one expression (a string operator).

### Syntax

```
<expression> + <expression>
```

### Arguments

expression

Any valid Graphisoft SQL expression of any of the data types in the character string data type category. Both expressions must be of the same data type, or one expression must be able to be implicitly converted to the data type of the other expression.

### Result Types

Returns the data type of the argument with the highest precedence.

## LIKE

Determines whether or not a given character string matches a specified pattern. A pattern can include regular characters and wildcard characters. During pattern matching, regular characters must exactly match the characters specified in the character string. Wildcard characters, however, can be matched with arbitrary fragments of the character string. Using wildcard characters makes the LIKE operator more flexible than using the = and <> string comparison operators. If any of the arguments are not of character string data type then it will be implicitly converted to character string data type, if possible.

### Syntax

```
<match expression> [ NOT ] LIKE <pattern> [ ESCAPE <escape chracter> ]
```

### Arguments

match expression

Any valid Graphisoft SQL expression of character string data type.

pattern

The pattern to search for in **<match expression>**, and can include these valid wildcard characters.

Wildcard character	Description	Example
%	Any string of zero or more characters.	SELECT * FROM windows WHERE library_part_name LIKE '%Casement%' finds all windows with the word 'Casement' anywhere in the name of the library part.
_ (underscore)	Any single character.	SELECT * FROM objects WHERE library_part_name LIKE 'Chair _ ' finds all objects whose library part name first is 7 character length and starts with the word 'Chair'.
[ ]	Any single character within the specified range ([a-f]) or set ([abcdef]).	SELECT * FROM objects WHERE library_part_name LIKE 'Chair [123]' finds all objects whose library part name is 'Chair 1' or 'Chair 2' or 'Chair 3'.
[^]	Any single character not within the specified range ([^a-f]) or set ([^abcdef]).	SELECT * FROM objects WHERE library_part_name LIKE '[^S]%' finds all objects whose library part name doesn't start with the letter 'S'.

escape character

Any valid Graphisoft SQL expression of any of the data types of the character string data type category. **<escape character>** has no default and must consist of only one character.

Result Types

BOOL

Arithmetic Operators

The usual arithmetic operators are available:

- [- \(unary minus\)](#)
- [+ \(addition\)](#)
- [- \(subtraction\)](#)

- [\\* \(multiplication\)](#)
- [/ \(division\)](#)
- [% \(modulo\)](#)

## + (Addition)

Adds two numbers.

### Syntax

```
<numeric expression> + <numeric expression>
```

### Arguments

numeric expression

Any valid Graphisoft SQL expression of any of the data types of the numeric data type category.

### Result Types

Returns the data type of the argument with the higher precedence.

## / (Division)

Divides one number by another (an arithmetic division operator). If an integer **<dividend expression>** is divided by an integer **<divisor expression>**, the result is an integer whose fractional part (if any) will be truncated.

### Syntax

```
<dividend expression> / <divisor expression>
```

### Arguments

dividend expression

The numeric expression to divide. **<dividend expression>** can be any valid Graphisoft SQL expression of any of the data types of the numeric data type category.

divisor expression

The numeric expression to divide the dividend by. **<divisor expression>** can be any valid Graphisoft SQL expression of any of the data types of the numeric data type category.

### Result Types

Returns the data type of the argument with the higher precedence.

## % (Modulo)

Provides the remainder of one number divided by another.

## Syntax

```
<dividend expression> % <divisor expression>
```

## Arguments

dividend expression

The numeric expression to divide. **<dividend expression>** must be any valid Graphisoft SQL expression of the integer data type category. (A modulo is the integer that remains after two integers are divided.)

divisor expression

The numeric expression to divide the dividend by. **<divisor expression>** must be any valid Graphisoft SQL expression of any of the data types of the integer data type category.

## Result Types

INT

# \* (Multiplication)

Multiplies two expressions (an arithmetic multiplication operator).

## Syntax

```
<numeric expression> * <numeric expression>
```

## Arguments

numeric expression

Any valid Graphisoft SQL expression of any of the data types of the numeric data type category.

## Result Types

Returns the data type of the argument with the higher precedence.

# - (Subtraction)

Subtracts two numbers.

## Syntax

```
<numeric expression> - <numeric expression>
```

## Arguments



numeric expression

Any valid Graphisoft SQL expression of any of the data types of the numeric data type category.

## Result Types

Returns the data type of the argument with the higher precedence.

# - (Unary Minus)

An unary operator that returns the negative value of a numeric expression.

## Syntax

- <numeric expression>

## Arguments

numeric expression

Any valid Graphisoft SQL expression of any of the data types of the numeric data type category.

## Result Types

Returns the data type of <numeric expression>, except that an unsigned **TINYINT** expression is promoted to a signed **SMALLINT** result.

# Image Functions

The following image functions are available:

- [GETIMAGEMIMETYPE](#)
- [CONVERTIMAGE](#)

# GETIMAGEMIMETYPE

Returns the MIME type of the specified IMAGE.

## Syntax

[GETIMAGEMIMETYPE](#) ( <image expression> )

## Arguments

image expression

Any valid Graphisoft SQL image expression.

## Result Types

VARCHAR

# CONVERTIMAGE

Converts the specified **<source image>** to the specified **<image format>**.

## Syntax

```
CONVERTIMAGE ( <source image> , <image format> )
```

## Arguments

source image

Any valid Graphisoft SQL image expression.

target format

Any valid character string expression. The MIME type of the required format of the result image. If the required conversion is not available, then an exception condition is raised. Available image formats: image/jpeg, image/gif, image/png, etc.

## Result Types

IMAGE

# Data Type Conversion Functions

Available data type conversion functions:

- [CAST](#)

# CAST

Explicitly converts an expression of one data type to another.

## Syntax

```
CAST ( <expression> AS <data type> )
```

## Arguments

expression

Any valid Graphisoft SQL expression.

data type

The target data type.

## Result Types

Returns the same value as **<data type>**.

# Statements

The following statements can be used in Graphisoft SQL:

- [SELECT](#)

## JOIN

### Syntax

```
<table reference> , <table reference>
<table reference> CROSS JOIN <table reference>
<table reference> [ INNER ] JOIN <table reference> <join condition>
<table reference> LEFT [ OUTER ] JOIN <table reference> <join condition>
<table reference> RIGHT [ OUTER ] JOIN <table reference> <join condition>
```

### Examples

```
SELECT * FROM walls
    INNER JOIN filltypes ON walls.filltype_id = filltypes.id
```

```
SELECT * FROM walls
    INNER JOIN materials AS ma ON walls.material_id_on_sidea = ma.id
    INNER JOIN materials AS mb ON walls.material_id_on_sideb = mb.id
```

### Remarks

- **INNER JOIN** and , (comma) are semantically equivalent; both do a full join between tables specified, but using **the INNER JOIN** form is much faster.

# SELECT

### Syntax

```
SELECT [ ALL | DISTINCT ] <select list> <from clause> [ <where clause> ] [ <order by clause> ]
```

### Description

**SELECT** statement is used to retrieve rows selected from one or more tables. The **<select list>** indicates the columns you want to retrieve.

### Remarks

- The expressions of the **<select list>** can be aliased using **AS**. These column aliases can be used in the **<order by clause>** but they can't be used in the **<where clause>**:

```
SELECT volume * 1.2 AS increased_volume FROM walls
       order by increased_volume DESC
```

- In the **<from clause>** you can define table aliases which can be used anywhere in the **SELECT** statement:

```
SELECT mywall.volume, walls.length FROM walls AS mywall WHERE mywall.length >
1
```

- You can refer to a column as **<column name>** or **<table name>.<column name>**. You need not specify a **<table name>** or prefix for a column reference in a **SELECT** statement unless the reference would be ambiguous.

```
SELECT walls.id, walls.volume, filltypes.id FROM walls
       INNER JOIN filltypes ON walls.filltype_id = filltypes.id
```

- The options **DISTINCT** and **ALL** specify whether duplicate rows should be returned. The default is (**ALL**): all matching rows are returned. **DISTINCT** specifies that duplicate rows in the result set should be removed.
- [Join syntaxes supported in Graphisoft SQL](#)

## Graphisoft SQL Grammar

The syntactic notation used in this grammar definition is an extended version of BNF ("Backus Naur Form" or "Backus Normal Form"). In BNF, each syntactic element of the language is defined by means of a production rule. This defines the element in terms of a formula consisting of the characters, character strings, and syntactic elements that can be used to form an instance of it.

The version of BNF used in this grammar definition makes use of the following symbols:

Symbol	Meaning
< >	Angle brackets delimit character strings that are the names of syntactic elements, the non-terminal symbols of the SQL language.
::=	The definition operator. This is used in a production rule to separate the element defined by the rule from its definition. The element being defined appears to the left of the operator and the formula that defines the element appears to the right.
[ ]	Square brackets indicate optional elements in a formula. The portion of the formula within the brackets may be explicitly specified or may be omitted.
{ }	Braces"{}" group elements in a formula. The portion of the formula within the braces shall be explicitly specified.
	The alternative operator. The vertical bar indicates that the portion of the formula following the bar is an alternative to the portion preceding the bar. If the vertical bar appears at a position where it is not enclosed in braces or square brackets, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of a formula enclosed in braces or square brackets, it specifies alternatives for the

	contents of the innermost pair of such braces or brackets.
...	The ellipsis indicates that the element to which it applies in a formula may be repeated any number of times. If the ellipsis appears immediately after a closing brace "}", then it applies to the portion of the formula enclosed between that closing brace and the corresponding opening brace "{". If an ellipsis appears after any other element, then it applies only to that element.
!!	Introduces ordinary English text. This is used when the definition of a syntactic element is not expressed in BNF.

Spaces are used to separate syntactic elements. Multiple spaces and line breaks are treated as a single space. Apart from those symbols to which special functions were given above, other characters and character strings in a formula stand for themselves. In addition, if the symbols to the right of the definition operator in a production consist entirely of BNF symbols, then those symbols stand for themselves and do not take on their special meaning.

Terminal Characters

```
<simple Latin letter> ::=
    <simple Latin upper case letter>
  | <simple Latin lower case letter>
```

```
<simple Latin upper case letter> ::=
    A | B | C | D | E | F | G | H | I | J | K | L | M
  | N | O | P | Q | R | S | T | U | V | W | X | Y | Z
```

```
<simple Latin lower case letter> ::=
    a | b | c | d | e | f | g | h | i | j | k | l | m
  | n | o | p | q | r | s | t | u | v | w | x | y | z
```

```
<digit> ::=
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
<hexa digit> ::=
    <digit> | A | B | C | D | E | F | a | b | c | d | e | f
```

```
<space> ::= !! space character in character set in use
```

<double quote> ::= "

<quote> ::= '

<left paren> ::= (

<right paren> ::= )

<asterisk> ::= \*

<solidus> ::= /

<percent> :: %

<comma> ::= ,

<period> ::= .

<plus sign> ::= +

<minus sign> ::= -

<less than operator> ::= <

<greater than operator> ::= >

<equals operator> ::= =

<question mark> ::= ?

## Tokens

<regular identifier> ::= <identifier body>

<identifier body> ::=  
    <identifier start> [ <identifier part>... ]

<identifier start> ::=  
    <simple Latin letter>  
    | <underscore>

<identifier part> ::=  
    <simple Latin letter>  
    | <digit>  
    | <underscore>

<delimited identifier> ::=  
    <double quote> <delimited identifier body> <double quote>

<delimited identifier body> ::=  
    <delimited identifier part>...

<delimited identifier part> ::=  
    <simple Latin letter>  
    | <digit>  
    | <underscore>  
    | <space>

<not equals operator> ::= <>

<greater than or equals operator> ::= >=

<less than or equals operator> ::= <=

## Literals

<literal> ::=  
    <boolean literal>  
    | <numeric literal>  
    | <character string literal>

<numeric literal> ::=  
    <unsigned numeric literal>  
    | <signed numeric literal>

<signed numeric literal> ::=  
    [ <sign> ] <unsigned numeric literal>

<sign> ::= <plus sign> | <minus sign>

<unsigned numeric literal> ::=  
    <exact numeric literal>  
    | <approximate numeric literal>

<exact numeric literal> ::=  
    <unsigned integer> [ <period> [ <unsigned integer> ] ]  
    | <period> <unsigned integer>

<approximate numeric literal> ::= <mantissa> E <exponent>



<mantissa> ::= <exact numeric literal>

<exponent> ::= <signed integer>

<signed integer> ::= [ <sign> ] <unsigned integer>

<unsigned integer> ::= <digit>...

<character string literal> ::=  
    <quote> [ <character representation>... ] <quote>

<character representation> ::= <nonquote character>

<nonquote character> ::= !! any character other than a <quote> in character set in use

<boolean literal> ::= TRUE | FALSE

## Names and Identifiers

<identifier> ::=  
    <regular identifier>  
    | <delimited identifier>

<table name> ::= <identifier>

<column name> ::= <identifier>

<correlation name> :: <identifier>

## Scalar Expressions

<data type> ::=  
    <character string type>  
    | <national character string type>  
    | <binary string type>  
    | <numeric type>  
    | <image type>

<character string type> ::=  
    <char type>  
    | <varchar type>  
    | <character large object type>  
    | <text type>

<national character string type> ::=  
    <national char type>  
    | <national varchar type>  
    | <national character large object type>  
    | <national text type>

<binary string type> ::=  
    <binary type>  
    | <varbinary type>  
    | <binary large object type>  
    | <longvarbinary type>

<numeric type> ::=  
    <boolean type>  
    | <exact numeric type>  
    | <approximate numeric type>

<exact numeric type> ::=  
    <tinyint type>  
    | <smallint type>  
    | <int type>

| <bigint type>

<approximate numeric type> ::=  
    <real type>  
    | <float type>  
    | <double type>

<char type> ::=  
    CHAR [ <left paren> <length> <right paren> ]  
    | CHARACTER [ <left paren> <length> <right paren> ]

<varchar type> ::=  
    VARCHAR [ <left paren> <length> <right paren> ]  
    | CHAR VARYING [ <left paren> <length> <right paren> ]  
    | CHARACTER VARYING [ <left paren> <length> <right paren> ]

<character large object type> ::=  
    CLOB [ <left paren> <length> <right paren> ]  
    | CHAR LARGE OBJECT [ <left paren> <length> <right paren> ]  
    | CHARACTER LARGE OBJECT [ <left paren> <length> <right paren> ]

<text type> ::=  
    TEXT  
    | LONGVARCHAR

<national char type> ::=  
    NCHAR [ <left paren> <length> <right paren> ]  
    | NATIONAL CHAR [ <left paren> <length> <right paren> ]  
    | NATIONAL CHARACTER [ <left paren> <length> <right paren> ]

<national varchar type> ::=  
    NVARCHAR [ <left paren> <length> <right paren> ]  
    | NCHAR VARYING [ <left paren> <length> <right paren> ]  
    | NATIONAL CHAR VARYING [ <left paren> <length> <right paren> ]  
    | NATIONAL CHARACTER VARYING [ <left paren> <length> <right paren> ]

<national character large object type> ::=  
    NCLOB [ <left paren> <length> <right paren> ]

| **NCHAR LARGE OBJECT** [ <left paren> <length> <right paren> ]

<national text type> ::=

**NTEXT**  
| **NLONGVARCHAR**

<binary type> ::=

**BINARY** [ <left paren> <length> <right paren> ]

<varbinary type> ::=

**VARBINARY** [ <left paren> <length> <right paren> ]

<binary large object type> ::=

**BLOB** [ <left paren> <length> <right paren> ]  
| **BINARY LARGE OBJECT** [ <left paren> <length> <right paren> ]

<longvarbinary type> ::= **LONGVARBINARY**

<boolean type> ::=

**BOOL**  
| **BOOLEAN**

<tinyint type> ::= **TINYINT**

<smallint type> ::= **SMALLINT**

<int type> ::=

**INT**  
| **INTEGER**

<bigint type> ::= **BIGINT**

<real type> ::= REAL

<float type> ::=  
    FLOAT [ <left paren> <precision> <right paren> ]

<double type> ::=  
    DOUBLE  
    | DOUBLE PRECISION

<image type> ::= IMAGE

<length> ::= <unsigned integer>

<precision> ::= <unsigned integer>

<value specification> ::=  
    <literal>  
    | <dynamic parameter specification>

<dynamic parameter specification> ::= <question mark>

<table reference> ::=  
    <table name> [ [ AS ] <correlation name> ]  
    | <joined table>

<column reference> ::= [ <qualifier> <period> ] <column name>

<qualified> ::=  
    <table name>  
    | <correlation name>

```
<cast specification> ::=  
    CAST <left paren> <cast operand> AS <cast target> <right paren>
```

```
<cast operand> ::= <value expression>
```

```
<cast target> ::= <data type>
```

```
<value expression> ::=  
    <numeric value expression>  
    | <string value expression>
```

```
<value expression primary> ::=  
    <unsigned value specification>  
    | <column reference>  
    | <left paren> <value expression> <right paren>  
    | <cast specification>
```

```
<numeric value expression> ::=  
    <term>  
    | <numeric value expression> <plus sign> <term>  
    | <numeric value expression> <minus sign> <term>
```

```
<term> ::=  
    <factor>  
    | <term> <asterisk> <factor>  
    | <term> <solidus> <factor>  
    | <term> <percent> <factor>
```

```
<factor> ::=  
    [ <sign> ] <numeric primary>
```

```
<numeric primary> ::=  
    <value expression primary>  
    | <numeric value function>
```

```
<string value expression> ::=
```

<character value expression>

<character value expression> ::=  
    <concatenation>  
    | <character factor>

<concatenation> ::=  
    <character value expression> <plus sign> <character factor>

<character factor> ::= <character primary>

<character primary> ::=  
    <value expression primary>  
    | <string value function>

## Query Expressions

<row value constructor> ::=  
    <row value constructor element>

<row value constructor element> ::=  
    <value expression>  
    | <null specification>

<null specification> ::= **NULL**

<table expression> ::=  
    <from clause> [ <where clause> ] [ <order by clause> ]

<from clause> ::=

```
FROM <table reference> [ { <comma> <table reference> }... ]
```

```
<joined table> ::=  
    <cross join>  
    | <qualified join>
```

```
<cross join> ::=  
    <table reference> CROSS JOIN <table reference>
```

```
<qualified join> ::=  
    <table reference> [ <join type> ] JOIN <table reference> [ <join  
specification> ]
```

```
<join specification> ::= <join condition>
```

```
<join condition> ::= <search condition>
```

```
<join type> ::=  
    INNER  
    | <outer join type> [ OUTER ]
```

```
<outer join type> ::=  
    LEFT  
    | RIGHT
```

```
<where clause> ::=  
    WHERE <search condition>
```

```
<query specification> ::=  
    SELECT [ <set quantifier> ] <select list> <table expression>
```

```
<select list> ::=  
    <asterisk>
```



```
| <select sublist> [ { <comma> <select sublist> }... ]
```

```
<select sublist> ::=  
    <derived column>  
    | <qualifier> <period> <asterisk>
```

```
<derived column> ::= <value expression> [ <as clause> ]
```

```
<as clause> ::= AS <column name>
```

```
<query expression> ::= <query specification>
```

## Predicates

```
<predicate> ::=  
    <comparison predicate>  
    | <between predicate>  
    | <in predicate>  
    | <like predicate>  
    | <null predicate>
```

```
<comparison predicate> ::=  
    <row value constructor> <comp op> <row value constructor>
```

```
<comp op> ::=  
    <equals operator>  
    | <not equals operator>  
    | <less than operator>  
    | <greater than operator>  
    | <less than or equals operator>  
    | <greater than or equals operator>
```

```
<between predicate> ::=
```

```
<row value constructor> [ NOT ] BETWEEN <row value constructor> AND <row value constructor>
```

```
<in predicate> ::=  
    <row value constructor> [ NOT ] IN <in predicate value>
```

```
<in predicate value> ::=  
    <left paren> <in value list> <right paren>
```

```
<in value list> ::=  
    <value expression> { <comma> <value expression> }...
```

```
<like predicate> ::=  
    <match value> [ NOT ] LIKE <pattern> ESCAPE <escape character>
```

```
<match value> ::= <character value expression>
```

```
<pattern> ::= <character value expression>
```

```
<escape character> ::= <character value expression>
```

```
<null predicate> ::=  
    <row value constructor> IS [ NOT ] NULL
```

```
<search condition> ::=  
    <boolean term>  
    | <search condition> OR <boolean term>
```

```
<boolean term> ::=  
    <boolean factor>  
    | <boolean term> AND <boolean factor>
```

```
<boolean factor> ::=  
    [ NOT ] <boolean primary>
```

```
<boolean primary> ::=  
    <predicate>  
    | <left paren> <search condition> <right paren>
```

## Statements

```
<statement> ::= <select statement>
```

```
<select statement> ::=  
    <query expression> [ <order by clause> ]
```

```
<order by clause> ::=  
    ORDER BY <sort specification list>
```

```
<sort specification list> ::=  
    <sort specification> [ { <comma> <sort specification> }... ]
```

```
<sort specification> ::=  
    <sort key> [ <ordering specification> ]
```

```
<sort key> ::= <column name>
```

```
<ordering specification> ::= ASC | DESC
```