

The SQL Guide to Ingres
Appendix B – Scalar Functions

Rick F. van der Lans

Appendix **A**

Scalar Functions

Ingres supports many scalar functions. For all functions, this appendix presents the name, a description, the data type of the result of the function, and a few examples. The functions are sorted by name.

ABS(*par1*)

Description: This function returns the absolute value of a numeric expression.

Data type: Numeric

ABS(-25) ⇒ 25
ABS(-25.89) ⇒ 25.89

ACOS(*par1*)

Description: This function returns, in radians, the angle size for any given arc cosine value. The value of the parameter must lie between -1 and 1 inclusive.

Data type: Numeric

ACOS(0) ⇒ 1.5707963267949
ACOS(-1) - PI() ⇒ 0
ACOS(1) ⇒ 0

ANSIDATE(*par1*)

Description: This function transforms the parameter in an ansidate value. The parameter must have the format of a correct date or time stamp.

Data type: Ansi date

```
ANSIDATE('2005-12-31')      ⇒ '2005-12-31'
ANSIDATE(DATE '2005-12-31') ⇒ '2005-12-31'
ANSIDATE('12-31-2005 12:13:14.123') ⇒ '2005-12-01'
```

ASCII(*par1*)

Description: This function transforms the value to an alphanumeric value. The parameter can have a numeric or alphanumeric data type only.

Data type: alphanumeric

```
ASCII(100)      ⇒ '100'
ASCII(123.45)   ⇒ '123.45'
ASCII('Database') ⇒ 'Database'
ASCII(NULL)     ⇒ NULL
```

ASIN(*par1*)

Description: This function returns, in radians, the angle size for any given arc sine value. The value of the parameter must lie between -1 and 1 inclusive; otherwise, the result is equal to the null value.

Data type: Numeric

```
ASIN(1)      ⇒ 1.5707963267949
ASIN(0)      ⇒ 0
ASIN(NULL)   ⇒ NULL
```

ATAN(*par1*)

Description: This function returns, in radians, the angle size for any given arc tangent value.

Data type: Numeric

```
ATAN(0)      ⇒ 0
ATAN(100)    ⇒ 1.56079666010823
ATAN(1)      ⇒ 0.78539816339745
```

ATAN2(*par1*, *par2*)

Description: This function returns, in radians, the angle size for x and y coordinates. These are, respectively, the first and second parameter of the function.

Data type: Numeric

```
ATAN2(30,30) ⇒ 0.78539816339745
ATAN2(-1,-1) ⇒ -2.35619449019234
```

BIGINT(*par1*)

Description: This function transforms (via casting) an expression into a value with the BIGINT data type. The expression may have a numeric or an alphanumeric data type. If the expression is alphanumeric, it must consist of digits.

Data type: Bigint

```
BIGINT(7)           ⇒ 7
BIGINT('7')        ⇒ 7
POWER(BIGINT(10),18) ⇒ 1000000000000000000
```

BIT_ADD(*par1*, *par2*)

Description: This function executes a logical ‘add’ on two hexadecimal values (the two parameters). If the value becomes too large, it is simply truncated. If the two values in bytes are unequal in length, the shortest will be filled with zeroes on the left-hand side.

Data type: Hexadecimal (length is equal to that of the longest value)

```
BIT_ADD(X'01',X'10')   ⇒ X'11'
BIT_ADD(X'08',X'08')   ⇒ X'10'
BIT_ADD(X'1000',X'1000') ⇒ X'2000'
```

BIT_AND(*par1*, *par2*)

Description: This function executes a logical ‘and’ on two hexadecimal values (the two parameters). If a 1 occurs at both values on the same position, then the result also has a 1 on that same position. If the two values in bytes are unequal in length, the shortest is filled with zeroes on the left-hand side.

Data type: Hexadecimal (length is equal to that of the longest value)

```
BIT_AND(X'01',X'10')   ⇒ X'00'
BIT_AND(X'01',X'01')   ⇒ X'01'
BIT_AND(X'08',X'0E')   ⇒ X'08'
BIT_AND(X'1000',X'1000') ⇒ X'1000'
```

BIT_NOT(*par1*)

Description: This function executes a logical ‘not’ on two hexadecimal values. This means that every 1 becomes a 0 and the other way round.

Data type: Hexadecimal

```
BIT_NOT(X'01')   ⇒ X'FE'
BIT_NOT(X'08')   ⇒ X'F7'
BIT_NOT(X'1000') ⇒ X'FFFF'
```

BIT_OR(*par1*, *par2*)

Description: This function executes a logical 'or' on two hexadecimal values (the two parameters). If a 1 occurs at one or two of the values on a certain position, then the result also has a 1 on that same position. If the two values in bytes are unequal in length, the shortest is filled with zeroes on the left-hand side.

Data type: Hexadecimal (length is equal to that of the longest value)

```
BIT_OR(X'01',X'10')    ⇒ X'11'
BIT_OR(X'01',X'01')    ⇒ X'01'
BIT_OR(X'08',X'0E')    ⇒ X'0E'
BIT_OR(X'1000',X'1000') ⇒ X'1000'
```

BIT_XOR(*par1*, *par2*)

Description: This function executes a logical 'xor' on two hexadecimal values (the two parameters). If a 1 occurs on a position at one of the values, then the result also has a 1 on that same position. If the two values in bytes are unequal in length, the shortest is filled with zeroes on the left-hand side.

Data type: Hexadecimal (length is equal to that of the longest value)

```
BIT_XOR(X'01',X'10')    ⇒ X'11'
BIT_XOR(X'01',X'01')    ⇒ X'11'
BIT_XOR(X'08',X'0E')    ⇒ X'00'
BIT_XOR(X'1000',X'1000') ⇒ X'0000'
```

BYTE(*par1*, *par2*)

Description: This function transforms the value of the first parameter to binary data. The second parameter is not mandatory and indicates the maximum length of the result. If the value of the second parameter is greater than the length of the first parameter value, the length of the latter is used.

Data type: Byte

```
BYTE('Database')    ⇒ 4461746162617365
BYTE('Database',4)  ⇒ 44617461
BYTE(4)              ⇒ 0400
```

C(*par1*, *par2*)

Description: This function transforms the value of the first parameter to a C string, or a string as used in the programming language C. The second parameter is not mandatory and indicates the maximum length of the result. If the value of the second parameter is greater than the length of the first parameter value, the length of the latter is used.

Data type: Byte

```
C('Database')    ⇒ 'Database'
C('Database',4)  ⇒ 'Data'
```

CAST(*par1* AS *par3*)

Description: This function converts the data type of the first parameter. The second parameter must be equal to one of the supported data types. This specification

Data type: Depends on the second parameter

```
CAST(45 AS CHAR(2))      ⇒ '45'
CAST('123' AS INTEGER)   ⇒ 123
CAST(123 AS DECIMAL(5,2)) ⇒ 123.00
CAST('1997-01-15' AS DATE) ⇒ 1997-01-15
```

CEIL(*par1*)

Description: This function returns the highest whole number that is greater than or equal to the value of the parameter. See also the CEILING function.

Data type: Numeric

```
CEIL(13.43) ⇒ 14
CEIL(-13.43) ⇒ -13
CEIL(13) ⇒ 13
```

CEILING(*par1*)

Description: This function returns the highest whole number that is greater than or equal to the value of the parameter. See also the CEIL function.

Data type: Numeric

```
CEILING(13.43) ⇒ 14
CEILING(-13.43) ⇒ -13
CEILING(13) ⇒ 13
```

CHAR(*par1*, *par2*)

Description: This function transforms the first parameter into an alphanumeric value. The data type of the parameter may be alphanumeric, temporal or numeric. The result always has an alphanumeric data type with a fixed length. A second parameter can be specified indicating the maximum length of the result. The length of the result is equal to the length of the alphanumeric value of the first parameter if it is smaller than the value of the maximum length.

Data type: Alphanumeric

```
CHAR('Database')          ⇒ 'Database'
CHAR('Database',4)        ⇒ 'Data'
CHAR('Database',10)       ⇒ 'Database'
LENGTH(CAST('Database' AS VARCHAR(10))) ⇒ 8
LENGTH(CHAR(CAST('Database' AS VARCHAR(10)),10)) ⇒ 10
```

Examples of the CHAR function in which the first parameter is equal to integer (the length of this value is equal to the number of digits of which the value consists):

```

CHAR(100)                ⇨ '100'
CHAR(100,1)              ⇨ '1'
CHAR(100/10)             ⇨ '10'
LENGTH(CHAR(CAST(100 AS SMALLINT))) ⇨ 3
LENGTH(CHAR(CAST(100 AS INTEGER))) ⇨ 3

```

Examples of the CHAR function in which the data type of the first parameter is equal to decimal:

```

CHAR(123.45) ⇨ '123.45'
CHAR(123.45,1) ⇨ '1'
CHAR(123.45,4) ⇨ '123.'
CHAR(123.45,5) ⇨ '123.4'

```

Examples of the CHAR function in which the data type of the first parameter is equal to float:

```

CHAR(123E5) ⇨ '1.23E+007'
CHAR(123E5,5) ⇨ '1.23'

```

Examples of the CHAR function in which the data type of the first parameter is equal to interval:

```

CHAR(INTERVAL '2007-05-25') ⇨ '2007-05-25'
CHAR(INTERVAL '2007-05-25',5) ⇨ '2007-'
CHAR(INTERVAL '8' DAY) ⇨ '8 00:00:00'
CHAR(INTERVAL '8' DAY,5) ⇨ '8 00:'

```

CHAREXTRACT(*par1*, *par2*)

Description: This function extracts a character from an alphanumeric value (the first parameter). The second parameter indicates which character that must be. If the value of the second parameter is greater than the length of the first parameter, the result is an empty string.

Data type: Alphanumeric

```

CHAREXTRACT('Database',3) ⇨ 't'
CHAREXTRACT('Database',10) ⇨ ''

```

CHARACTER_LENGTH(*par1*)

Description: This function returns the length of an alphanumeric expression.

Data type: Numeric

```

CHARACTER_LENGTH('database') ⇨ 8
CHARACTER_LENGTH('') ⇨ 0
CHARACTER_LENGTH(NULL) ⇨ NULL

```


CHR(*par1*)

Description: This function returns the alphanumeric character belonging to the ASCII code specified with the numeric parameter. If the value of the parameter is greater than 256, the function returns the result of the calculation parameter modulo 256.

Data type: Alphanumeric

```
CHR(80)                                ⇒ 'P'
CHR(82) + CHR(105) + CHR(99) + CHR(107) ⇒ 'Rick'
CHR(80+256)                             ⇒ 'P'
```

COALESCE(*par1, par2, par3, ...*)

Description: This function can have a variable number of parameters. The value of the function is equal to the value of the first parameter that is not equal to null.

If E_1 , E_2 , and E_3 are three expressions, the specification:

```
COALESCE( $E_1$ ,  $E_2$ ,  $E_3$ )
```

is equivalent to the following case expression:

```
CASE
  WHEN  $E_1$  IS NOT NULL THEN  $E_1$ 
  WHEN  $E_2$  IS NOT NULL THEN  $E_2$ 
  WHEN  $E_3$  IS NOT NULL THEN  $E_3$ 
  ELSE NULL
END
```

Data type: Depends on the parameters

```
COALESCE('John', 'Jim', NULL)          ⇒ 'John'
COALESCE(NULL, NULL, NULL, 'John', 'Jim') ⇒ 'John'
```

CONCAT(*par1, par2*)

Description: This function combines two alphanumeric values. You can achieve the same effect with the || operator.

Data type: Alphanumeric

```
CONCAT('Data', 'base') ⇒ 'Database'
```

COS(*par1*)

Description: This function returns, in radians, the cosine value for any angle size. The result is always a value between -1 and 1 inclusive.

Data type: Numeric

```
COS(0)           ⇒ 1
COS(PI())        ⇒ -1
COS(PI()/2)     ⇒ 6.12303176911189e-017
```

DATE(*par1*)

Description: This function transforms the parameter into a date value. The parameter should have the format of a correct date, time, or timestamp.

Data type: Ingresdate

```
DATE('12-31-2005')           ⇒ '2005-12-31 00:00:00'
DATE(DATE '2005-12-31')      ⇒ '2005-12-31 00:00:00'
DATE('12-31-2005 12:13:14') ⇒ '2005-12-01 12:13:14'
DATE('12:13:14')            ⇒ '2008-01-04 12:13:14'
DATE('12-31-2005 12:13:14.123') ⇒ '2005-12-01 12:13:14'
```

DATE(*par1*)

Description: This function determines the date when a number of seconds has elapsed since January 1, 1970 GMT at 00:00 hours. The parameter indicates the number of seconds. The result is a date with the format DD-MMM-YY. If the number of days is smaller than 10, a space is added.

Data type: Alphanumeric

```
DATE(10000)           ⇒ ' 1-jan-70'
DATE(60*60*24*365*38) ⇒ '23-dec-07'
```

DATE4(*par1*)

Description: This function determines the date when a number of seconds has elapsed since January 1, 1970 GMT at 00:00. The parameter indicates the number of seconds. The result is a date, according to the format defined with the II_DATE_FORMAT parameter; see also Section 5.2.8. Table 5.1 shows all the values that this parameter can have. The right column contains the belonging output formats.

Data type: Alphanumeric

Imagine that II_DATE_FORMAT has the value US, then these are the results of the following examples:

```
DATE4(10000)           ⇒ '01-jan-70'
DATE4(60*60*24*365*38) ⇒ '23-dec-07'
```

Imagine that `II_DATE_FORMAT` has the value `ISO`, then these are the results of the same examples:

```
DATE4(10000)           ⇒ '700101'
DATE4(60*60*24*365*38) ⇒ '071223'
```

DATE_GMT(*par1*)

Description: This function transforms the value of a time, date, or timestamp expression to the format `YYYY-MM-DD HH:MM:SS GMT`. Here, a local time is converted to the GMT time. In the following examples we assume that the expressions are processed in the time zone 1 hour east of GMT.

Data type: Alphanumeric

```
DATE_GMT('10-25-2008 15:00:00')           ⇒ '2008-10-25 13:00:00 GMT'
DATE_GMT(TIMESTAMP '2008-10-25 15:00:00') ⇒ '2008-10-25 15:00:00 GMT'
DATE_GMT(DATE '2008-10-25')                ⇒ '2008-10-25 00:00:00 GMT'
```

DATE_PART(*par1, par2*)

Description: This function subtracts a certain time unit from the value of a time, date, or timestamp expression (the second parameter). The first parameter indicates the time unit. Allowed time units are:

```
second, seconds, sec, secs
minute, minutes, min, mins
hour, hours, hr, hrs
day, days
week, weeks, wk, wks
iso-week, iso-wk
month, months, mo, mos
quarter, quarters, qtr, qtrs
year, years, yr, yrs
```

Datatype: integer

```
DATE_PART('SECONDS', TIMESTAMP '2008-11-12 13:14:15') ⇒ 15
DATE_PART('MINUTES', TIMESTAMP '2008-11-12 13:14:15') ⇒ 14
DATE_PART('DAYS', TIMESTAMP '2008-11-12 13:14:15')   ⇒ 12
DATE_PART('WEEKS', TIMESTAMP '2008-11-12 13:14:15')  ⇒ 45
DATE_PART('MONTHS', TIMESTAMP '2008-11-12 13:14:15') ⇒ 11
DATE_PART('QUARTERS', TIMESTAMP '2008-11-12 13:14:15') ⇒ 4
```

DATE_TRUNC(*par1, par2*)

Description: This function rounds off the value of a time, date, or timestamp expression (the second parameter) to a certain time unit. The first parameter indicates the time unit. Allowed time units are:

```

second, seconds, sec, secs
minute, minutes, min, mins
hour, hours, hr, hrs
day, days
week, weeks, wk, wks
iso-week, iso-wk
month, months, mo, mos
quarter, quarters, qtr, qtrs
year, years, yr, yrs

```

If the time units seconds, minutes, and hours are rounded off, they get the value 1. Rounding off by the quarter means that you round off to the first day of the quarter concerned and rounding off by the week means that you round off to the first day of the current week.

Data type: Equal to that of the second parameter, so date, time or timestamp

```

DATE_TRUNC('SECONDS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-11-12 13:14:15.000000000'
DATE_TRUNC('MINUTES', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-11-12 13:14:00.000000000'
DATE_TRUNC('HOURS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-11-12 13:00:00.000000000'
DATE_TRUNC('DAYS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-11-12 00:00:00.000000000'
DATE_TRUNC('WEEKS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-11-10 00:00:00.000000000'
DATE_TRUNC('MONTHS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-11-01 00:00:00.000000000'
DATE_TRUNC('QUARTERS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-10-01 00:00:00.000000000'
DATE_TRUNC('YEARS', TIMESTAMP '2008-11-12 13:14:15') ⇒ '2008-01-01 00:00:00.000000000'

```

DAY(*par1*)

Description: This function returns the number of the day of the month from a date or timestamp expression. The value of the result is always a whole number between 1 and 31 inclusive.

Data type: Numeric

```

DAY (DATE '2008-01-01')           ⇒ 1
DAY (TIMESTAMP '2008-01-20 09:11:11') ⇒ 20
DAY (CURRENT_DATE)               ⇒ 21
DAY (CURRENT_TIME)               ⇒ 0
DAY (CURRENT_TIMESTAMP)          ⇒ 21

```

DEC(*par1, par2, par3*)

Description: See also the DECIMAL function.

DECIMAL(*par1, par2, par3*)

Description: This function transforms the value of a numeric or alphanumeric expression in a value with a DECIMAL data type.

Data type: decimal

If the data type of the first parameter is numeric, two additional parameters *may* be specified. The second parameter represents the precision of the result, and must be big enough for the value of the first parameter. The third parameter represents the scale (the number of decimal spaces) of the result. If this parameter is left out, the scale is set to zero. If the second parameter is left out as well,

the precision depends on the data type of the value. If it is a smallint or integer1, the precision is equal to 5, if it is an integer then 11, and if the data type of the parameter is equal to float, float4, decimal or money, the precision is equal to 15.

```
DECIMAL(10)           ⇒ 10
DECIMAL(10,2)        ⇒ 10
DECIMAL(10.6666,5,1) ⇒ 10.6
```

If the data type of the first parameter is alphanumeric, you must be able to transform the value of the first parameter to a number, and you *must* specify two additional parameters: the precision and the scale.

```
DECIMAL('10',2,0)     ⇒ 10
DECIMAL('100',4,1)   ⇒ 100.0
DECIMAL('10.6666',10,6) ⇒ 10.666600
```

DOW(*par1*)

Description: This function returns the name of the day of the week from a date or timestamp expression. As parameter, an alphanumeric literal may also be specified, provided it has the format of a date literal. The result of this function is always equal to one of the following values: Mon, Tue, Wed, Thu, Fri, Sat or Sun.

Data type: Alphanumeric

```
DOW(DATE '2005-07-28') ⇒ 'Thu'
DOW(CURRENT_TIMESTAMP) ⇒ 'Fri'
```

EXP(*par1*)

Description: This function returns the result of the number *e* to the power of *x*, where *x* is the value of the parameter and *e* the basis of natural logarithms.

Data type: Numeric

```
EXP(1) ⇒ 2.71828182845905
EXP(2) ⇒ 7.38905609893065
```

EXTRACT(*par1* FROM *par2*)

Description: This function retrieves a component from a date, time, or timestamp expression (the second parameter). The first parameter indicates the component to extract. This must be one of the following values: SECOND, MINUTE, HOUR, DAY, MONTH, YEAR, TIMEZONE_MINUTE, or TIMEZONE_SECOND.

Data type: Numeric

```

EXTRACT(SECOND FROM TIMESTAMP '2005-07-29 12:13:14')      ⇒ 14
EXTRACT(SECOND FROM DATE '2005-07-29')                   ⇒ 0
EXTRACT(HOUR FROM TIME '09:00:00')                       ⇒ 9
EXTRACT(YEAR FROM TIMESTAMP '2005-07-29 12:13:14')      ⇒ 2005
EXTRACT(YEAR FROM CURRENT_DATE)                          ⇒ 2008
EXTRACT(TIMEZONE_HOUR FROM TIMESTAMP '2005-07-29 12:13:14+01:30') ⇒ 1

```

FLOAT4(*par1*)

Description: This function transforms the value of the parameter to a value with a FLOAT4 data type (single byte floating point).

Data type: Float4

```

FLOAT4(100)      ⇒ 1.000000e+02
FLOAT4(123.45)  ⇒ 1.234500e+02
FLOAT4(1E0/7)   ⇒ 0.142857

```

FLOAT8(*par1*)

Description: This function transforms the value of the parameter to a value with a FLOAT8 data type (double byte floating point).

Data type: Float8

```

FLOAT8(100)      ⇒ 1.0000000000000000e+002
FLOAT8(123.45)  ⇒ 1.2345000000000000e+002
FLOAT8(1E0/7)   ⇒ 0.14285714285714

```

FLOOR(*par1*)

Description: This function returns the smallest whole number that is smaller than or equal to the value of the parameter.

Data type: Numeric

```

FLOOR(13.9)      ⇒ 13
FLOOR(-130.9)   ⇒ -131

```

GMT_TIMESTAMP(*par1*)

Description: This function determines the date and time when a number of seconds have elapsed since January 1, 1970 GMT at 00:00 hours. The parameter indicates the number of seconds. The result is a timestamp with the format YYYY_MM_DD HH:MM:SS GMT.

Data type: Alphanumeric

```

GMT_TIMESTAMP(100)      ⇒ '1970_01_01 00:01:40 GMT'
GMT_TIMESTAMP(60*60*24*365*38) ⇒ '2007_12_23 00:00:00 GMT'

```

HASH(*par1*)

Description: This function generates a numeric value of four bytes long based on each type of value. For this, Ingres uses a hashing algorithm. Note that the actual length of the value can effect the result of the function.

Data type: Integer

```

HASH(100)                ⇒ -1078508637
HASH(CAST(100 AS INTEGER)) ⇒ 1795113143
HASH('Database')        ⇒ -807756921
HASH(DATE '2008-01-01') ⇒ 1454801109

```

HEX(*par1*)

Description: This function returns the internal hexadecimal representation of the internal value of the parameter. The parameter can have any data type.

Data type: Alphanumeric

```

HEX('B')                ⇒ '42'
HEX('Database')         ⇒ '4461746162617365'
HEX(DATE '2008-12-31') ⇒ 'D8070C1F'
HEX(1)                   ⇒ '0001'
HEX(8)                   ⇒ '0008'

```

HOUR(*par1*)

Description: This function returns the value of the hour component from a date, time, or time-stamp expression. The value of the result is always a whole number between 0 and 23 inclusive.

Data type: Numeric

```

HOUR(TIMESTAMP '2008-01-01 12:13:14') ⇒ 12
HOUR(TIME '12:13:14')                 ⇒ 12
HOUR(DATE '2008-10-31')               ⇒ 0
HOUR(CURRENT_TIME)                    ⇒ 13

```

IFNULL(*par1*, *par2*)

Description: If the value of the first parameter is equal to the null value, the result of the function is equal to the value of the second parameter; otherwise, it is equal to the value of the first parameter.

The specification

```
IFNULL(E1, E2)
```

in which E₁, and E₂ are expressions, is equal to the following case expression:

```

CASE E1
  WHEN NULL THEN E2
  ELSE E1
END

```

Data type: Depends on the parameters

```
IFNULL(NULL, 'John') ⇒ 'John'
IFNULL('John', 'Jim') ⇒ 'John'
```

II_IPADDR(*par1*)

Description: This function transforms an IP address to a hexadecimal value of four bytes.

Data type: Numeric

```
II_IPADDR('127.0.0.0') ⇒ X'7F000000'
II_IPADDR('213.35.52.7') ⇒ X'D5233407'
```

INGRESDATE(*par1*)

Description: This function transforms the parameter into an ingresdate value. The parameter should have the format of a correct date, time, or timestamp.

Data type: ingresdate

```
INGRESDATE('12-31-2005')           ⇒ '2005-12-31 00:00:00'
INGRESDATE(DATE '2005-12-31')      ⇒ '2005-12-31 00:00:00'
INGRESDATE('12-31-2005 12:13:14') ⇒ '2005-12-01 12:13:14'
INGRESDATE('12:13:14')            ⇒ '2008-01-04 12:13:14'
INGRESDATE('12-31-2005 12:13:14.123') ⇒ '2005-12-01 12:13:14'
```

INT1(*par1*)

Description: This function transforms the value of the parameter to a value with an INT1 data type (single byte integer). This value must be smaller than 128. If the value of the parameter contains decimal places, they are removed. No rounding off takes places.

Data type: Integer

```
INT1(100)           ⇒ 100
INT1(1.6)           ⇒ 1
INT1(1.2365E+02)   ⇒ 123
INT1('4')          ⇒ 4
INT1(8/3)           ⇒ 2
```

INT2(*par1*)

Description: This function transforms the value of the parameter to a value with an INT2 data type (two bytes integer). This value must be smaller than 32,767. If the value of the parameter contains decimal places, they are removed. No rounding off takes place.

Data type: Integer

INT2(100)	⇒ 100
INT2(1.6)	⇒ 1
INT2(1.2365E+02)	⇒ 123
INT2('4')	⇒ 4
INT2(8/3)	⇒ 2

INT4(*par1*)

Description: This function transforms the value of the parameter to a value with an INT4 data type (four bytes integer). This value must be smaller than 2,147,483,647. If the value of the parameter contains decimal places, they are removed. No rounding off takes place.

Data type: Integer

INT4(100)	⇒ 100
INT4(1.6)	⇒ 1
INT4(1.2365E+02)	⇒ 123
INT4('4')	⇒ 4
INT4(8/3)	⇒ 2

INTERVAL(*par1*, *par2*)

Description: This function transforms the value of an ingres-interval (the second) to a certain time unit, for example a number of days or a number of hours. Here, a month is equal to 30.436875 days, and a year is equal to 364.2425 days. The first parameter indicates the time unit. Allowed time units are:

second, seconds, sec, secs
 minute, minutes, min, mins
 hour, hours, hr, hrs
 day, days
 week, weeks, wk, wks
 iso-week, iso-wk
 month, months, mo, mos
 quarter, quarters, qtr, qtrs
 year, years, yr, yrs

Data type: Equal to that of the second parameter, so date, time, or timestamp

INTERVAL('WEEKS', '210 DAYS')	⇒ 30
INTERVAL('DAYS', '2 YEARS')	⇒ 7.304850000000000e+002
INTERVAL('SECONDS', '1 HOURS + 100 SECONDS')	⇒ 3700
INTERVAL('MONTHS', '1 YEARS')	⇒ 12

INTERVAL_DTOS(*par1*)

Description: This function transforms the value of a time, timestamp or alphanumeric expression to an ansi-interval value with the pattern DD HH:MM:SS.FF.

Data type: Ansi interval

```

INTERVAL_DTOS('10 05:12:18')           ⇒ 10 05:12:18.000000000
INTERVAL_DTOS(TIME '12:13:14')         ⇒ 0 12:13:14.000000000
INTERVAL_DTOS(INTERVAL '5' DAY)        ⇒ 5 00:00:00.000000000
INTERVAL_DTOS(INTERVAL '5' DAY + INTERVAL '10' HOUR) ⇒ 5 10:00:00.000000000

```

INTERVAL_YTOM(*par1*)

Description: This function transforms the value of a date, timestamp, or alphanumeric expression to an ansi-interval value with the format YY-MM.

Data type: Ansi interval

```

INTERVAL_YTOM('2008-100')              ⇒ 2016-04
INTERVAL_YTOM(INTERVAL '5' MONTH)      ⇒ 0-05
INTERVAL_YTOM(INTERVAL '5' YEAR + INTERVAL '10' MONTH) ⇒ 5-10

```

INEXTRACT(*par1*, *par2*)

Description: This function returns a byte (as integer) from a value (the first parameter). The second parameter indicates which byte that must be. If the value of the second parameter is greater than the length of the first parameter or if the second parameter is smaller than 1, the result is equal to 0.

Data type: Integer

```

INEXTRACT(X'AB',1) ⇒ 171
INEXTRACT(X'AB',2) ⇒ 0
INEXTRACT('AB',1) ⇒ 65
INEXTRACT('AB',2) ⇒ 66

```

LEFT(*par1*, *par2*)

Description: This function returns the left part of an alphanumeric value (the first parameter). The second parameter indicates the length of the part used.

Data type: Alphanumeric

```

LEFT('database', 4)           ⇒ 'data'
LEFT('database', 0)           ⇒ ''
LEFT('database', 10)          ⇒ 'database'
LEFT('database', NULL)        ⇒ NULL
LENGTH(LEFT('database', 0))   ⇒ 0
LENGTH(LEFT('database', 10)) ⇒ 8
LENGTH(LEFT('database', NULL)) ⇒ NULL

```

LENGTH(*par1*)

Description: This function returns the length in bytes of an alphanumeric value.

Data type: Numeric

```
LENGTH('database')      ⇒ 8
LENGTH('data  ')       ⇒ 8
LENGTH(RTRIM('abcd  ')) ⇒ 4
LENGTH('')              ⇒ 0
LENGTH(CAST(NULL AS CHAR(1))) ⇒ NULL
```

LN(*par1*)

Description: This function returns the logarithm to the base value e of the parameter. The value of the parameter should be greater than zero. See also the LOG function.

Data type: Numeric

```
LN(50)      ⇒ 3.91202300542815
LN(EXP(3))  ⇒ 3
LN(1)       ⇒ 0
LN(NULL)    ⇒ NULL
```

LOCATE(*par1*, *par2*, *par3*)

Description: This function returns the starting position of the second alphanumeric value within the first alphanumeric value. If the first alphanumeric value does not occur within the second, the LOCATE function returns the length of the first parameter plus one.

Data type: Numeric

```
LOCATE('Database', 'bas') ⇒ 5
LOCATE('Database', 'a')  ⇒ 2
LOCATE('Database', 'Rick') ⇒ 9
```

LOG(*par1*)

Description: This function returns the logarithm to the base value e of the parameter. The value of the parameter should be greater than zero. See also the LN function.

Data type: Numeric

```
LOG(50)      ⇒ 3.91202300542815
LOG(EXP(3))  ⇒ 3
LOG(1)       ⇒ 0
LOG(NULL)    ⇒ NULL
```

LONG_BYTE(*par1*, *par2*)

Description: This function transforms the value of the first parameter to binary data.

Data type: long_byte

```
BYTE('Database') ⇒ 4461746162617365
BYTE(4)           ⇒ 0400
```

LONG_VARCHAR(*par1*)

Description: This function transforms the parameter to an alphanumeric value with a variable length. The data type of the parameter may be alphanumeric, temporal, or numeric. The result always has an alphanumeric data type with a variable length.

Data type: Alphanumeric

```
LONG_VARCHAR('Database')           ⇒ 'Database'
LENGTH(LONG_VARCHAR(CAST('Database' AS VARCHAR(10)))) ⇒ 8
```

Examples of the LONG_VARCHAR function where the data type of the first parameter is equal to integer (the length of this value is equal to the number of digits of which the value consists):

```
LONG_VARCHAR(100)           ⇒ '100'
LONG_VARCHAR(100/10)       ⇒ '10'
LENGTH(LONG_VARCHAR(CAST(100 AS SMALLINT))) ⇒ 3
LENGTH(LONG_VARCHAR(CAST(100 AS INTEGER)))  ⇒ 3
```

Examples of the LONG_VARCHAR function where the data type of the first parameter is equal to decimal:

```
LONG_VARCHAR(123.45) ⇒ '123.45'
LONG_VARCHAR(123.45,1) ⇒ '1'
LONG_VARCHAR(123.45,4) ⇒ '123.'
LONG_VARCHAR(123.45,5) ⇒ '123.4'
```

Examples of the LONG_VARCHAR function where the data type of the first parameter is equal to float:

```
LONG_VARCHAR(123E5) ⇒ '1.23E+007'
LONG_VARCHAR(123E5,5) ⇒ '1.23'
```

Examples of the LONG_VARCHAR function where the data type of the first parameter is equal to interval:

```
LONG_VARCHAR(DATE '2007-05-25') ⇒ '2007-05-25'
LONG_VARCHAR(DATE '2007-05-25',5) ⇒ '2007-'
LONG_VARCHAR(INTERVAL '8' DAY) ⇒ '8 00:00:00'
LONG_VARCHAR(INTERVAL '8' DAY,5) ⇒ '8 00:'
```

LOWER(*par1*)

Description: This function converts all uppercase letters of the value of the parameter to lowercase letters.

Data type: Alphanumeric

```
LOWER('RICK') ⇒ 'rick'
```

LOWERCASE(*par1*)

Description: This function converts all uppercase letters of the value of the parameter to lowercase letters. See also the LOWER function.

Data type: Alphanumeric

```
LOWERCASE('RICK') ⇒ 'rick'
```

LPAD(*par1*, *par2*, *par3*)

Description: The value of the first parameter is filled in the front (the left side) with the value of the last parameter just until the total length of the value is equal to that of the second parameter. If the maximum length is smaller than that of the first parameter, the first parameter is shortened on the left side. If only two parameters exist, the value of the first parameter is filled with spaces.

Data type: Alphanumeric

```
LPAD('data', 16, 'base')    ⇒ 'basebasebasedata'
LENGTH(LPAD('data', 16, 'x')) ⇒ 16
LPAD('data', 6, 'base')    ⇒ 'badata'
LPAD('data', 2, 'base')    ⇒ 'da'
LPAD('data', 10)           ⇒ '      da'
```

LTRIM(*par1*)

Description: This function removes all blanks that appear at the beginning of the parameter.

Data type: Alphanumeric

```
LTRIM('  database') ⇒ 'database'
```

MICROSECOND(*par1*)

Description: This function returns the number of microseconds from a time or timestamp expression. The value of the result is always a whole number between 0 and 999999 inclusive.

Data type: Numeric

```
MICROSECOND(TIMESTAMP '2005-01-01 12:13:14.123456') ⇒ 123456
MICROSECOND(TIME '12:13:14.1')                      ⇒ 100000
MICROSECOND('12:13:14.123456')                      ⇒ 0
```

MINUTE(*par1*)

Description: This function returns the number of minutes from a time or timestamp expression. The value of the result is always a whole number between 0 and 59 inclusive.

Data type: Numeric

```
MINUTE(CURRENT TIME)           ⇒ 28
MINUTE(TIME '12:40:33')       ⇒ 40
MINUTE(TIMESTAMP '2008-05-15 12:22:33') ⇒ 22
```

MOD(*par1*)

Description: This function returns the remainder from the division of two parameters.

Data type: Numeric

```
MOD(15,4)           ⇒ 3
MOD(15.4, 4.4)     ⇒ 3
```

MONEY(*par1*)

Description: This function transforms a numeric value to a value with the money data type.

Data type: money

```
MONEY(100)           ⇒ $100.00
MONEY(1234.5678)    ⇒ $1234.57
```

MONTH(*par1*)

Description: This function returns the number of the month from a date or timestamp expression. The value of the result is always a whole number between 1 and 12 inclusive.

Data type: Numeric

```
MONTH(DATE '2008-07-29')       ⇒ 7
MONTH(TIMESTAMP '2008-11-29 09:40:50') ⇒ 11
```

NCHAR(*par1*, *par2*)

Description: This function transforms the first parameter into an alphanumeric value with the data type NCHAR and with the Unicode character set. The data type of the parameter may be alphanumeric, temporal, or numeric. The result always has an alphanumeric data type with a fixed length. A second parameter may be specified indicating what the maximum length of the result is. The length of the result is equal to the length of the alphanumeric value of the first parameter if it is smaller than the value of the maximum length.

Data type: alphanumeric

```
NCHAR('Database')    ⇒ 'Database'
NCHAR('Database',4)  ⇒ 'Data'
NCHAR('Database',10) ⇒ 'Database'
```

Examples of the NCHAR function where the data type of the first parameter is equal to integer (the length of this value is equal to the number of digits of which the value exists):

```
NCHAR(100)           ⇒ '100'
NCHAR(100,1)        ⇒ '1'
NCHAR(100/10)       ⇒ '10'
LENGTH(NCHAR(CAST(100 AS SMALLINT))) ⇒ 3
LENGTH(NCHAR(CAST(100 AS INTEGER)))  ⇒ 3
```

Examples of the NCHAR function where the data type of the first parameter is equal to decimal:

```
NCHAR(123.45)       ⇒ '123.45'
NCHAR(123.45,1)    ⇒ '1'
NCHAR(123.45,4)    ⇒ '123.'
NCHAR(123.45,5)    ⇒ '123.4'
```

Examples of the NCHAR function where the data type of the first parameter is equal to float:

```
NCHAR(123E5)        ⇒ '1.23E+007'
NCHAR(123E5,5)      ⇒ '1.23'
```

Examples of the NCHAR function where the data type of the first parameter is equal to interval:

```
NCHAR(DATE '2007-05-25') ⇒ '2007-05-25'
NCHAR(DATE '2007-05-25',5) ⇒ '2007-'
NCHAR(INTERVAL '8' DAY) ⇒ '8 00:00:00'
NCHAR(INTERVAL '8' DAY,5) ⇒ '8 00:'
```

NOTRIM(*part*)

Description: This function makes sure that spaces at the end of an alphanumeric value (the parameter) remain if they are placed in a column with a variable length. This function only works in embedded SQL.

Data type: Alphanumeric

```
NOTRIM('database   ') ⇒ 'database   '
```

NULLIF(*par1*, *par2*)

Description: If the value of the first parameter is not equal to that of the second parameter, the result of the function is equal to the null value; otherwise, it is equal to the first parameter. The specification

```
NULLIF(E1, E2)
```

in which E₁ and E₂ are two expressions, is equal to the following case expression:

```
CASE
  WHEN E1 = E2 THEN NULL
  ELSE E1
END
```

Data type: Depends on the parameters

```
NULLIF(CAST(NULL AS CHAR(1)), 'John') ⇒ NULL
NULLIF('John', 'Jim')                ⇒ 'John'
NULLIF('John', 'John')                ⇒ NULL
```

NVARCHAR(*par1*, *par2*)

Description: This function transforms the first parameter in an alphanumeric value with a variable length. The data type of the result is NVARCHAR with Unicode as character set. The data type of the parameter may be alphanumeric, temporal or numeric. The result always has an alphanumeric data type with a variable length. A second parameter may be specified indicating the maximum length of the result. The length of the result is equal to the length of the alphanumeric value of the first parameter if it is smaller than the value of the maximum length.

Data type: Alphanumeric

```
NVARCHAR('Database')    ⇒ 'Database'
NVARCHAR('Database',4)  ⇒ 'Data'
NVARCHAR('Database',10) ⇒ 'Database'
LENGTH(NVARCHAR(CAST('Database' AS VARCHAR(10)),10)) ⇒ 8
```

Examples of the NVARCHAR function where the data type of the first parameter is equal to integer (the length of this value is equal to the number of digits of which the value consists):

```
NVARCHAR(100)           ⇒ '100'
NVARCHAR(100,1)        ⇒ '1'
NVARCHAR(100/10)       ⇒ '10'
LENGTH(NVARCHAR(CAST(100 AS SMALLINT))) ⇒ 3
LENGTH(NVARCHAR(CAST(100 AS INTEGER)))  ⇒ 3
```


Examples of the VARCHAR function where the data type of the first parameter is equal to decimal:

```
NVARCHAR(123.45)    ⇒ '123.45'
NVARCHAR(123.45,1) ⇒ '1'
NVARCHAR(123.45,4) ⇒ '123.'
NVARCHAR(123.45,5) ⇒ '123.4'
```

Examples of the NVARCHAR function where the data type of the first parameter is equal to float:

```
NVARCHAR(123E5)    ⇒ '1.23E+007'
NVARCHAR(123E5,5) ⇒ '1.23'
```

Examples of the NVARCHAR function where the data type of the first parameter is equal to interval:

```
NVARCHAR(DATE '2007-05-25')    ⇒ '2007-05-25'
NVARCHAR(DATE '2007-05-25',5)  ⇒ '2007-'
NVARCHAR(INTERVAL '8' DAY)     ⇒ '8 00:00:00'
NVARCHAR(INTERVAL '8' DAY,5)   ⇒ '8 00:'
```

OBJECT_KEY(*part*)

Description: This function transforms the value of the parameter to an object_key. We can use this value for columns that are defined as table_key of object_key. The parameter must have an alphanumeric data type.

Data type: object_key

```
OBJECT_KEY('0100000000000000') ⇒ '0100000000000000'
OBJECT_KEY('4141414141414141') ⇒ '4141414141414141'
```

PAD(*part*)

Description: The value of the parameter is extended with spaces at the back (right-hand side) just until the total length of the value is equal to the length that the value actually should have. This function is useful when values with a variable length are used. Imagine that a column has the data type VARCHAR(10), and that it contains the value 'data'. By using the function PAD, the value is converted to the value 'data '.

Data type: Alphanumeric

```
PAD('data')                ⇒ 'data'
PAD(CAST('data' AS VARCHAR(10))) ⇒ 'data      '
LENGTH(PAD(CAST('data' AS VARCHAR(10)))) ⇒ 10
```

PI()

Description: This function returns the well-known number *pi*.

Data type: Numeric

```
PI()                ⇒ 3.14159265358979
PI()*100000         ⇒ 314159.265358979
```

POSITION(*par1* IN *par2*)

Description: This function returns the starting position of the first alphanumeric value within the second alphanumeric value. The function returns the value 0 if the first alphanumeric value does not appear within the second. Note that the term IN must be specified between the two parameters.

Data type: Numeric

```
POSITION('bas' IN 'database') ⇒ 5
POSITION('bas' IN 'system')  ⇒ 0
```

POWER(*par1*, *par2*)

Description: The value of the first expression is raised to a specific power. The second parameter indicates the power.

Data type: Numeric

```
POWER(4,3)    ⇒ 64
POWER(2.5,3)  ⇒ 15.625
POWER(4, 0.3) ⇒ 1.5157165665104
POWER(4, -2)  ⇒ 0.0625
```

QUARTER(*par1*)

Description: This function returns the quarter from a date or timestamp expression. The value of the result is always a whole number between 1 and 4 inclusive.

Data type: Numeric

```
QUARTER(DATE '2008-07-29')    ⇒ 3
QUARTER(TIMESTAMP '2005-12-31 23:59:59') ⇒ 4
QUARTER(CURRENT_DATE)        ⇒ 1
```

RANDOM()

Description: This function returns a random number with the integer data type.

Data type: Integer

```
RANDOM() ⇒ 14473873
RANDOM()  ⇒ 1372733
```

Important for this function is the system parameter RANDOM_SEED. It determines the starting point for the calculation of a random number. The SET RANDOM_SEED statement can change the value of this parameter. A global and a local RANDOM_SEED system parameter exists. The global applies to each user and each application until the mentioned SET statement changes it. The result of the RANDOM function is always the same when the following two statement are executed repeatedly, and that is 8621309.

```
SET RANDOM_SEED 1
SELECT RANDOM()
```

If no value for `RANDOM_SEED` has been specified, Ingres takes the ID process and multiplies it with the number of seconds that has elapsed since 1 January 1970.

RANDOM(par1, par2)

Description: This function returns a random number with the integer data type. The value is between the values of the two parameters. The system parameter `RANDOM_SEED` is also relevant for this function; see also the `RANDOM` function without parameters.

Data type: Integer

```
RANDOM(0,10)      ⇒ 6
RANDOM(1000,2000) ⇒ 1606
```

RANDOMF()

Description: This function works the same as the `RANDOM` function, except that the result is not integer but a floating point between 0 and 1. The system parameter `RANDOM_SEED` is also relevant for this function; see also the `RANDOM` function without parameters.

Data type: Float

```
RANDOMF() ⇒ 0.92695047007874
RANDOMF() ⇒ 0.16721244156361
```

RANDOMF(par1, par2)

Description: This function returns a random number of which the value is between the values of the two parameters. If both parameters are integer, the result is integer as well. If both parameters have the float data type, the result will have that too. An if one of the two is integer, and the other float, the result will be float. The system parameter `RANDOM_SEED` is also relevant for this function; see also the `RANDOM` function without parameters.

Data type: depends on the parameters

```
RANDOMF(0,10)      ⇒ 7
RANDOMF(0E0, 1E1) ⇒ 3.49839523434639
RANDOMF(0, 1E1)    ⇒ 4.13499001413584
```

REPLACE(par1, par2, par3)

Description: This function replaces parts of the value of an alphanumeric expression with another value. The second parameter indicates the value to be replaced and the third parameter indicates the new value.

Data type: Alphanumeric

```
REPLACE('database','a','e')      ⇒ 'detebase'
REPLACE('database','ba','warehou') ⇒ 'datawarehouse'
REPLACE('data base',' ','')      ⇒ 'database'
```

RIGHT(*par1*, *par2*)

Description: This function returns the right part of an alphanumeric value (the first parameter). The second parameter indicates the length of the part used.

Data type: Alphanumeric

```
RIGHT('database', 4)      ⇒ 'base'
RIGHT('database', 0)      ⇒ ''
RIGHT('database', 10)     ⇒ 'database'
RIGHT('database', NULL)   ⇒ NULL
LENGTH(RIGHT('database', 0)) ⇒ 0
LENGTH(RIGHT('database', 10)) ⇒ 8
LENGTH(RIGHT('database', NULL)) ⇒ NULL
```

ROUND(*par1*, *par2*)

Description: This function rounds numbers to a specified number of decimal places. Note that the data type of the result is equal to that of the first parameter.

Data type: Numeric

```
ROUND(123.456,2) ⇒ 123.460
ROUND(123.456,1) ⇒ 123.500
ROUND(123.456,0) ⇒ 123.000
ROUND(123.456,-1) ⇒ 120.000
ROUND(123.456,-2) ⇒ 100.000
ROUND(123.456,5) ⇒ 123.456
ROUND(1234,-2) ⇒ 1200
ROUND(123456,10) ⇒ 123456
```

RPAD(*par1*, *par2*, *par3*)

Description: The value of the first parameter is filled at the back (the right side) with the value of the third parameter just until the total length of the value is equal to that of the second parameter. If the maximum length is smaller than that of the first parameter, the first parameter is shortened on the right side.

Data type: Alphanumeric

```
RPAD('data', 16, 'base') ⇒ 'databasebasebase'
RPAD('data', 6, 'base') ⇒ 'databa'
RPAD('data', 2, 'base') ⇒ 'da'
```

RTRIM(*par1*)

Description: This function removes all blanks from the end of the value of the parameter.

Data type: Alphanumeric

```
RTRIM('database ') ⇒ 'database'
CONCAT(RTRIM('data '), 'base') ⇒ 'database'
```

SECOND(*par1*)

Description: This function returns the number of seconds from a time or timestamp expression. The value of the result is always a whole number between 0 and 59 inclusive.

Data type: Numeric

```
SECOND(TIME '12:40:33')           ⇒ 33
SECOND(TIMESTAMP '2008-01-01 14:40:25') ⇒ 25
SECOND(CURRENT_TIME)             ⇒ 5
```

SHIFT(*par1*, *par2*)

Description: This function moves the characters in the first parameter to the left or to the right. The second parameter indicates how many positions the characters must be moved. If the value of the second parameter is greater than 0, the characters are moved to the right, and otherwise to the left. If the value has a fixed length, the result has the same fixed length. If it has a variable length, the value can become as large as the defined length.

Data type: Alphanumeric

```
SHIFT('database',4)              ⇒ '   data'
SHIFT('database',-4)             ⇒ 'base'
SHIFT(CAST('database' AS VARCHAR(10)),4) ⇒ '   databa'
SHIFT(CAST('database' AS VARCHAR(10)),-4) ⇒ 'base'
```

SIGN(*par1*)

Description: This function returns the character of a numeric value. If the value of the parameter is negative, then the result is equal to -1, if it is greater than zero, then it is +1, and if the value is equal to 0, the result of the function will also be equal to 0.

Data type: Numeric

```
SIGN(50)   ⇒ 1
SIGN(0)    ⇒ 0
SIGN(-50)  ⇒ -1
```

SIN(*par1*)

Description: This function returns, in radians, the sine value of any angle size. The result is always a value between -1 and 1 inclusive.

Data type: Numeric

```
SIN(0)           ⇒ 0
SIN(PI()/2)     ⇒ 1
SIN(PI())       ⇒ 1.22460635382238E-016
SIN(1)-COS((PI()/2) - 1) ⇒ 0
```

SIZE(*par1*, *par2*)

Description: This function returns the maximum length in characters of an alphanumeric value.

Data type: Numeric

```
SIZE('database')           ⇒ 8
SIZE(CAST('database' AS CHAR(10))) ⇒ 10
SIZE(CAST('database' AS VARCHAR(10))) ⇒ 10
```

SOUNDEX(*part*)

Description: This function returns the SOUNDEX code of the alphanumeric parameter. A SOUNDEX code consists of four characters. Alphanumeric values that sound roughly the same are converted to identical SOUNDEX codes. The SOUNDEX code is specified according to the following rules:

- All blanks at the beginning of the parameter are removed.
- All the following letters are removed from the parameter, provided that they do not appear on the first position: a e h i o u w y.
- The following values are assigned to the remaining letters:

```
b f p v      = 1
c g j k q s x z = 2
d t          = 3
l            = 4
m n          = 5
r            = 6
```

- If two linked letters have the same value, the second is removed.
- The code is broken after the fourth character.
- If the remaining code consists of less than four characters, it is filled with zeroes.
- Characters appearing after a blank are skipped.
- If the value of the parameter does not begin with a letter, the result is equal to 0000.

Data type: Alphanumeric

```
SOUNDEX('Smith')      ⇒ 'S530'
SOUNDEX('Smythe')     ⇒ 'S530'
SOUNDEX('Bill')       ⇒ 'B400'
SOUNDEX(' Bill')      ⇒ 'B400'
SOUNDEX('Billy')      ⇒ 'B400'
```

SQUEEZE(*part*)

Description: This function removes all the so-called *white spaces* at the beginning and end of an alphanumeric value and replaces all the white spaces that do not appear at the beginning or end with spaces. With white spaces, we mean the characters spaces, null (X'00'), newline (X'0A'), carriage return (X'0D'), horizontal tab (X'09') and end of text (X'04'). A string of white spaces in the middle is replaced by one space.

Data type: Numeric

```
'X' || SQUEEZE(' A A ') || 'X'  ⇒ 'XA AX'
SQUEEZE('A' || X'0A' || 'A')    ⇒ 'A A'
SQUEEZE('A' || X'000D09' || 'A') ⇒ 'A A'
```

SQRT(*par1*)

Description: This function returns the square root of the value of the parameter.

Data type: Numeric

```
SQRT(225) ⇒ 15
SQRT(200) ⇒ 14.14
```

SUBSTR(*par1*, *par2*, *par3*)

Description: This function extracts a part of the alphanumeric value of the first parameter. The second parameter identifies the starting position, and the third identifies its number of characters. If the second parameter is negative, counting characters starts from position 1. If the third parameter is not specified, up to the last character is included.

Data type: Alphanumeric

```
SUBSTR('database',5)    ⇒ 'base'
SUBSTR('database',10)   ⇒ ''
SUBSTR('database',5,2)  ⇒ 'ba'
SUBSTR('database',5,10) ⇒ 'base'
SUBSTR('database',-6)   ⇒ 'database'
```

SUBSTRING(*par1*, *par2*, *par3*)

Description: This function extracts a part of the alphanumeric value of the first parameter. The second parameter identifies the starting position, and the third identifies its number of characters. If the second parameter is negative, counting characters starts from position 1. If the third parameter is not specified, up to the last character is included.

Data type: Alphanumeric

```
SUBSTRING('database',5)    ⇒ 'base'
SUBSTRING('database',10)   ⇒ ''
SUBSTRING('database',5,2)  ⇒ 'ba'
SUBSTRING('database',5,10) ⇒ 'base'
SUBSTRING('database',-6)   ⇒ 'database'
```

SUBSTRING(*par1* FROM *par2* FOR *par3*)

Description: This function extracts a part of the alphanumeric value of the first parameter. The second parameter identifies the starting position, and the third identifies its number of characters. If the second parameter is negative, counting characters starts from position 1. If the third parameter is not specified, up to the last character is included.

Data type: Alphanumeric

```

SUBSTRING('database' FROM 5)      ⇒ 'base'
SUBSTRING('database' FROM 10)     ⇒ ''
SUBSTRING('database' FROM 5 FOR 2) ⇒ 'ba'
SUBSTRING('database' FROM 5 FOR 10) ⇒ 'base'
SUBSTRING('database' FROM -6)     ⇒ 'tabase'

```

TABLE_KEY(*par1*)

Description: This function transforms the value of the parameter to a table_key. We can use this value for columns that are defined as table_key of object_key. The parameter must have an alphanumeric data type.

Data type: table_key

```

TABLE_KEY('0100000000000000') ⇒ '01000000'
TABLE_KEY('4141414141414141') ⇒ '41414141'

```

TAN(*par1*)

Description: This function returns, in radians, the tangent value of any angle size.

Data type: Numeric

```

TAN(0)      ⇒ 0
TAN(PI())   ⇒ -1.22460635382238E-016
TAN(PI()/4) ⇒ 1
TAN(1)      ⇒ 1.5574077246549

```

TEXT(*par1*, *par2*)

Description: This function transforms the first parameter to an alphanumeric value with the data type TEXT. The data type of the parameter may be alphanumeric, temporal or numeric. The result always has an alphanumeric data type with a fixed length. A second parameter may be specified indicating the maximum length of the result. The length of the result is equal to the length of the alphanumeric value of the first parameter if it is smaller than the value of the maximum length.

Data type: Alphanumeric

```

TEXT('Database')      ⇒ 'Database'
TEXT('Database',4)    ⇒ 'Data'
TEXT('Database',10)   ⇒ 'Database'
LENGTH(TEXT(CAST('Database' AS VARCHAR(10)),10)) ⇒ 8

```

Examples of the TEXT function where the data type of the first parameter is equal to integer (the length of this value is equal to the number of digits of which the value consists):

```

TEXT(100)      ⇒ '100'
TEXT(100,1)    ⇒ '1'
TEXT(100/10)   ⇒ '10'
LENGTH(TEXT(CAST(100 AS SMALLINT))) ⇒ 3
LENGTH(TEXT(CAST(100 AS INTEGER))) ⇒ 3

```

Examples of the TEXT function where the data type of the first parameter is equal to decimal:


```
TEXT(123.45)    ⇒ '123.45'
TEXT(123.45,1) ⇒ '1'
TEXT(123.45,4) ⇒ '123.'
TEXT(123.45,5) ⇒ '123.4'
```

Examples of the TEXT function where the data type of the first parameter is equal to float:

```
TEXT(123E5)    ⇒ '1.23E+007'
TEXT(123E5,5) ⇒ '1.23'
```

Examples of the TEXT function where the data type of the first parameter is equal to interval:

```
TEXT(DATE '2007-05-25') ⇒ '2007-05-25'
TEXT(DATE '2007-05-25',5) ⇒ '2007-'
TEXT(INTERVAL '8' DAY) ⇒ '8 00:00:00'
TEXT(INTERVAL '8' DAY,5) ⇒ '8 00:'
```

TIME(*part*)

Description: This function returns the time part of a time, timestamp, or alphanumeric expression. Any time zone will be removed.

Data type: Time

```
TIME(TIMESTAMP '2005-12-08 12:00:00')    ⇒ '12:00:00.000000000'
TIME(TIMESTAMP '2005-12-08 12:00:00.123456') ⇒ '12:00:00.123456000'
TIME(TIME '12:00:00')                    ⇒ '12:00:00.000000000'
TIME(TIME '12:00:00+03:00')              ⇒ '12:00:00.000000000'
TIME('12:13:14')                        ⇒ '12:13:14.000000000'
```

TIME(*part*)

Description: This function returns the time when a number of seconds have elapsed since January 1, 1970 GMT at 00:00 hours. The parameter indicates the number of seconds. The result is a time with the pattern HH:MM.

Data type: Alphanumeric

```
TIME(10000)          ⇒ '03:46'
TIME(60*60*24*365*38) ⇒ '01:00'
```

TIME_LOCAL()

Description: This function transforms a time, timestamp or alphanumeric expression to a value with the data type time. If a time zone has been specified, the specified time is converted to the local time. The following examples assume that the expressions are executed in the time zone GMT+1:00. For example, in the expression TIME_LOCAL(TIME '12:00:00+03:00') the time zone +03:00 occurs. This refers to the time zone 3 hours east of Greenwich. If 3 hours east of Greenwich it is 12:00, it is 10:00 o'clock in the morning in the local time zone (1 hour east of Greenwich).

Data type: Time

```

TIME_LOCAL(TIME '12:00:00')           ⇒ '12:00:00.000000000'
TIME_LOCAL(TIME '12:00:00+00:00')    ⇒ '13:00:00.000000000'
TIME_LOCAL(TIME '12:00:00+01:00')    ⇒ '12:00:00.000000000'
TIME_LOCAL(TIME '12:00:00.1234-08:30') ⇒ '21:30:00.123400000'
TIME_LOCAL('12:13:14')               ⇒ '12:13:14.000000000'
TIME_LOCAL(TIMESTAMP '2008-12-08 12:13:14') ⇒ '12:13:14.000000000'

```

TIMESTAMP(*par1*, *par2*)

Description: This function transforms the first parameter into a timestamp value (including the second fraction). Any time zone will be removed.

Data type: Timestamp

```

TIMESTAMP('2008-12-08')              ⇒ '2008-12-08 00:00:00.000000000'
TIMESTAMP(TIMESTAMP '2008-12-08 12:00:00') ⇒ '2008-12-08 12:00:00.000000000'
TIMESTAMP(TIMESTAMP '2008-12-08 12:00:00+03:00') ⇒ '2008-12-08 12:00:00.000000000'
TIMESTAMP(DATE '2008-12-08')         ⇒ '2008-12-08 00:00:00.000000000'
TIMESTAMP(TIME '12:00:00')           ⇒ '2008-01-05 12:00:00.000000000'
TIMESTAMP(CURRENT_DATE)              ⇒ '2008-01-05 00:00:00.000000000'

```

TIMESTAMP_LOCAL()

Description: This function transforms a date, time, timestamp or alphanumeric expression to a value with the data type timestamp. If a time zone has been specified, the specified time is converted to the local time. The following examples assume that the expressions are executed in the time zone GMT+1:00. For example, in the expression `TIMESTAMP_LOCAL(TIME '2008-01-01 12:00:00+03:00')` the time zone +03:00 occurs. This refers to the time zone 3 hours east of Greenwich. If 3 hours east of Greenwich it is 12:00, it is 10:00 o'clock in the morning in the local time zone (1 hour east of Greenwich).

Data type: Timestamp

```

TIMESTAMP_LOCAL(TIMESTAMP '2008-12-08 12:13:14') ⇒ '2008-12-08 12:13:14.000000000'
TIMESTAMP_LOCAL(TIMESTAMP '2008-12-08 12:13:14+03:00') ⇒ '2008-12-08 10:13:14.000000000'
TIMESTAMP_LOCAL(TIME '12:00:00') ⇒ '2008-01-28 12:00:00.000000000'
TIMESTAMP_LOCAL(DATE '2008-01-01') ⇒ '2008-01-01 00:00:00.000000000'
TIMESTAMP_LOCAL('2008-10-01 12:13:14.1234') ⇒ '2008-10-01 12:13:14.123400000'

```

TIMESTAMP_WITH_TZ()

Description: This function transforms a date, time, timestamp or alphanumeric expression to a value with the data type timestamp. The result always contains a time zone and is the GMT time. The specified time is a local time. The following examples assume that the expressions are executed in the time zone GMT+1:00. For example, the value of the expression `TIMESTAMP_WITH_TZ(TIMESTAMP '2008-01-01 12:00:00')` is '2008-01-01 12:00:00+01:00'.

Data type: Timestamp

```

TIMESTAMP_WITH_TZ (TIMESTAMP '2008-12-08 12:13:14')      ⇨ '2008-12-08 12:13:14.000000000'
TIMESTAMP_WITH_TZ (TIMESTAMP '2008-12-08 12:13:14+03:00') ⇨ '2008-12-08 10:13:14.000000000'
TIMESTAMP_WITH_TZ (TIME '12:00:00')                      ⇨ '2008-01-28 12:00:00.000000000'
TIMESTAMP_WITH_TZ (DATE '2008-01-01')                   ⇨ '2008-01-01 00:00:00.000000000'
TIMESTAMP_WITH_TZ ('2008-10-01 12:13:14.1234')         ⇨ '2008-10-01 12:13:14.123400000'

```

TIMESTAMP_WO_TZ(*par1*, *par2*)

Description: This function transforms the first parameter to a timestamp value, including the seconds fraction. Any time zone will be removed. This function is equal to the `TIMESTAMP` function.

Data type: Timestamp

```

TIMESTAMP_WO_TZ('2008-12-08')                            ⇨ '2008-12-08 00:00:00.000000000'
TIMESTAMP_WO_TZ(TIMESTAMP '2008-12-08 12:00:00')       ⇨ '2008-12-08 12:00:00.000000000'
TIMESTAMP_WO_TZ(TIMESTAMP '2008-12-08 12:00:00+03:00') ⇨ '2008-12-08 12:00:00.000000000'
TIMESTAMP_WO_TZ(DATE '2008-12-08')                     ⇨ '2008-12-08 00:00:00.000000000'
TIMESTAMP_WO_TZ(TIME '12:00:00')                       ⇨ '2008-01-05 12:00:00.000000000'
TIMESTAMP_WO_TZ(CURRENT_DATE)                          ⇨ '2008-01-05 00:00:00.000000000'

```

TIME_WITH_TZ()

Description: This function transforms a time, timestamp or alphanumeric expression to a value with the data type time. The result always contains a time zone and is the GMT time. The specified time is a local time. The following examples assume that the expressions are executed in the time zone GMT+1:00. For example, the value of the expression `TIMESTAMP_ TIME_WITH_TZ(TIME '12:00:00')` is '12:00:00+01:00'.

Data type: Time

```

TIME_WITH_TZ(TIME '12:00:00')                            ⇨ '12:00:00.000000000+01:00'
TIME_WITH_TZ(TIME '12:00:00+01:00')                    ⇨ '12:00:00.000000000+01:00'
TIME_WITH_TZ('12:13:14')                              ⇨ '12:13:14.000000000+01:00'
TIME_WITH_TZ(TIMESTAMP '2008-12-08 12:13:14')         ⇨ '12:13:14.000000000+01:00'

```

TIME_WO_TZ()

Description: This function returns the time part of a time, timestamp, or alphanumeric expression. Any time zone will be removed.

Data type: Time

```

TIME_WO_TZ(TIMESTAMP '2005-12-08 12:00:00')            ⇨ '12:00:00.000000000'
TIME_WO_TZ (TIMESTAMP '2005-12-08 12:00:00.123456')   ⇨ '12:00:00.123456000'
TIME_WO_TZ (TIME '12:00:00')                          ⇨ '12:00:00.000000000'
TIME_WO_TZ (TIME '12:00:00+03:00')                    ⇨ '12:00:00.000000000'
TIME_WO_TZ ('12:13:14')                              ⇨ '12:13:14.000000000'

```

TRIM(*par1*)

Description: This function removes all blanks from end of an alphanumeric value (the parameter). Blanks in the middle are not removed.

Data type: Alphanumeric

```
TRIM('database ') ⇒ 'database'
TRIM(' da ta ') ⇒ ' da ta'
```

TRIM([LEADING | TRAILIN | BOTH] *par1* FROM *par2*)

Description: If an alphanumeric value (the first parameter) appears at the start or end of another alphanumeric value (the second parameter), it is removed. Before the first parameter, you may specify the terms LEADING, TRAILING, or BOTH. Adding BOTH has no effect on the result. If LEADING is specified, only values at the start are removed; with TRAILING, only the values at the end are removed. For the value of the first parameter only the first character counts.

Data type: Alphanumeric

```
TRIM(' ' FROM ' data base ') ⇒ 'data base'
TRIM('a' FROM 'database') ⇒ 'database'
TRIM('x' FROM 'xxdatabasexx') ⇒ 'database'
TRIM('xy' FROM 'xydatabasexx') ⇒ 'ydatabase'
TRIM(LEADING ' ' FROM ' data base ') ⇒ 'data base '
TRIM(TRAILING 'da' FROM ' data base ') ⇒ ' data base'
```

TRUNC(*par1*, *par2*)

Description: This function truncates numbers to a specified number of decimal places. See also the TRUNCATE function.

Data type: Numeric

```
TRUNC(123.567, -1) ⇒ 120.000
TRUNC(123.567, 1) ⇒ 123.500
TRUNC(123.567, 5) ⇒ 123.567
```

TRUNCATE(*par1*, *par2*)

Description: This function truncates numbers to a specified number of decimal places.

Data type: Numeric

```
TRUNCATE(123.567, -1) ⇒ 120.000
TRUNCATE(123.567, 1) ⇒ 123.500
TRUNCATE(123.567, 5) ⇒ 123.567
```

UNHEX(*par1*)

Description: This function returns the hexadecimal representation of the parameter. Each pair of characters is converted to the corresponding character.

Data type: Alphanumeric

```
UNHEX('334538') ⇒ '3E8'
UNHEX('E7') ⇒ 'ç'
UNHEX(HEX('SQL')) ⇒ 'SQL'
```

UPPER(*par1*)

Description: This function converts all lowercase letters of the value of the parameter to uppercase letters.

Data type: Alphanumeric

```
UPPER('Database') ⇒ 'DATABASE'
```

UPPERCASE(*par1*)

Description: This function converts all lowercase letters of the value of the parameter to uppercase letters. See also the UPPER function.

Data type: Alphanumeric

```
UPPERCASE('Database') ⇒ 'DATABASE'
```

UUID_COMPARE(*par1, par2*)

Description: This function compares two UUID codes; see also the description of the UUID_CREATE function. The result of the function is 0 if the two are equal, 1 if the first value is greater than the second, and -1 if the first value is smaller.

Data type: Integer

```
UUID_COMPARE('D195AE00BD14914AB488760EE9C8C7C4', '74745742851611489262013D7088858B') ⇒ 1
UUID_COMPARE('74745742851611489262013D7088858B', 'D195AE00BD14914AB488760EE9C8C7C4') ⇒ -1
```

UUID_CREATE()

Description: This function generates an 16-byte wide unique code. The abbreviation UUID stands for *Universal Unique Identifier*. The first three parts of this code are derived from the system time. The fourth part must make sure that the codes are unique, in case duplicate values arise because of time zones. The fifth part identifies the server in a certain way. Generating unique values is not guaranteed; but is likely.

Data type: Hexadecimal

```
UUID_CREATE() ⇒ X'D195AE00BD14914AB488760EE9C8C7C4'
UUID_CREATE() ⇒ X'74745742851611489262013D7088858B'
```

UUID_FROM_CHAR()

Description: This function transforms a generated UUID with an alphanumeric data type to an internal value with a hexadecimal data type; see also the description of the `UUID_CREATE` function.

Data type: Hexadecimal

```
UUID_FROM_CHAR(UUID_TO_CHAR(UUID_CREATE()))
⇒ '0399D1B375CDEA468EDC272B25EDFCD3'
UUID_FROM_CHAR('5641aae5-f0a0-4061-ac42-f2ddece18371')
⇒ X'E5AA4156A0F06140AC42F2DDECE18371'
```

UUID_TO_CHAR()

Description: This function transforms a generated UUID (with a hexadecimal data type) to an alphanumeric value; see also the description of the `UUID_CREATE` function.

Data type: Alphanumeric

```
UUID_TO_CHAR(UUID_CREATE())           ⇒ '5641aae5-f0a0-4061-ac42-f2ddece18371'
UUID_FROM_CHAR(UUID_TO_CHAR(UUID_CREATE())) ⇒ X'0399D1B375CDEA468EDC272B25EDFCD3'
```

VARBYTE(*par1*, *par2*)

Description: This function transforms the value of the first parameter to binary data. The difference with the `BYTE` function is that the result of the `VARBYTE` function has a variable and not a fixed length. The second parameter is not mandatory and indicates the maximum length of the result. If the value of the second parameter is greater than the length of the first parameter value, the length of the latter is used.

Data type: Byte

```
VARBYTE('Database')   ⇒ 4461746162617365
VARBYTE('Database',4) ⇒ 44617461
VARBYTE(4)             ⇒ 0400
```

VARCHAR(*par1*, *par2*)

Description: This function transforms the first parameter to an alphanumeric value with a variable length. The data type of the parameter may be alphanumeric, temporal, or numeric. The result always has an alphanumeric data type with a variable length. A second parameter may be specified indicating the maximum length of the result. The length of the result is equal to the length of the alphanumeric value of the first parameter if it is smaller than the value of the maximum length.

Data type: Alphanumeric

```
VARCHAR('Database')           ⇒ 'Database'
VARCHAR('Database',4)        ⇒ 'Data'
VARCHAR('Database',10)       ⇒ 'Database'
LENGTH(CAST('Database' AS VARCHAR(10))) ⇒ 8
LENGTH(VARCHAR(CAST('Database' AS VARCHAR(10)),10)) ⇒ 8
```

Examples of the VARCHAR function where the data type of the first parameter is equal to integer (the length of this value is equal to the number of digits of which the value consists):

```

VARCHAR(100)           ⇒ '100'
VARCHAR(100,1)        ⇒ '1'
VARCHAR(100/10)       ⇒ '10'
LENGTH(VARCHAR(CAST(100 AS SMALLINT))) ⇒ 3
LENGTH(VARCHAR(CAST(100 AS INTEGER)))  ⇒ 3

```

Examples of the VARCHAR function where the data type of the first parameter is equal to decimal:

```

VARCHAR(123.45)   ⇒ '123.45'
VARCHAR(123.45,1) ⇒ '1'
VARCHAR(123.45,4) ⇒ '123.'
VARCHAR(123.45,5) ⇒ '123.4'

```

Examples of the VARCHAR function where the data type of the first parameter is equal to float:

```

VARCHAR(123E5)   ⇒ '1.23E+007'
VARCHAR(123E5,5) ⇒ '1.23'

```

Examples of the VARCHAR function where the data type of the first parameter is equal to interval:

```

VARCHAR(DATE '2007-05-25')   ⇒ '2007-05-25'
VARCHAR(DATE '2007-05-25',5) ⇒ '2007-'
VARCHAR(INTERVAL '8' DAY)    ⇒ '8 00:00:00'
VARCHAR(INTERVAL '8' DAY,5)  ⇒ '8 00:'

```

WEEK(*part*)

Description: This function returns the week from a date or timestamp expression. The days before the first Monday of the year, are considered to belong to week 0. In other words, week 1 begins on the first Monday of the year. The value of the result is always a whole number between 0 and 53 inclusive.

Data type: Numeric

```

WEEK('1988-07-29') ⇒ 30
WEEK('1997-01-01') ⇒ 1
WEEK('2000-12-31') ⇒ 53
WEEK(CURDATE())    ⇒ 7

```

WEEK_ISO(*part*)

Description: This function calculates a week out of a date or timestamp expression. The week number is calculated according to the ISO 8601 definition for week numbers. This means that week 1 begins on the first Thursday of the year. The value of the result is always a number greater than or equal to 0, and smaller than or equal to 53.

Data type: Numeric

```
WEEK_ISO (DATE '1988-07-29')      ⇒ 30
WEEK_ISO (TIMESTAMP '1997-01-01') ⇒ 1
WEEK_ISO ('12-31-2000')          ⇒ 52
WEEK_ISO (CURRENT_DATE)          ⇒ 3
```

YEAR(*part*)

Description: This function returns the number of the year from a date, time, or timestamp expression. The result is always a number greater than or equal 0.

Data type: Numeric

```
YEAR (DATE '2008-12-31')          ⇒ 2008
YEAR (TIMESTAMP '2005-01-01 12:13:14') ⇒ 2005
YEAR (TIME '12:13:14')           ⇒ 0
YEAR (DATE '2002-12-31' + INTERVAL '5' YEAR) ⇒ 200
```