the data types associated with that variable will result in an exact match. This distinction impacts function selection as described in "Function Resolution" on page 144.

There may be additional functions available because user-defined functions may be created in different schemas using one of these function signatures as a source (see "CREATE FUNCTION" on page 589 for details) or users may create external functions using their own programs.

Note:

- Built-in functions are provided with the database manager, providing a single result value, and they are identified as part of the SYSIBM schema. Examples of such functions include column functions such as AVG, operator functions such as "+", casting functions such as DECIMAL, and others such as SUBSTR.
- User-defined functions are functions that are registered to a database in SYSCAT.FUNCTIONS (using the CREATE FUNCTION statement). User-defined functions are never part of the SYSIBM schema. One such set of functions is provided with the database manager in a schema called SYSFUN.

Table 15. Supported Functions

Function name	**Schema**	**Description**	
	Input Parameters		**Returns**
ABS or ABSVAL	SYSFUN	Returns the absolute value of the argument.	
	SMALLINT		SMALLINT
	INTEGER		INTEGER
	BIGINT		BIGINT
	DOUBLE		DOUBLE
ACOS	SYSFUN	Returns the arccosine of the argument as an angle expressed in radians.	
	DOUBLE		DOUBLE
ASCII	SYSFUN	Returns the ASCII code value of the leftmost character of the argument as an integer.	
	CHAR		INTEGER
	VARCHAR(4000)		INTEGER
	CLOB(1M)		INTEGER
ASIN	SYSFUN	Returns the arcsine of the argument as an angle, expressed in radians.	
	DOUBLE		DOUBLE
ATAN	SYSFUN	Returns the arctangent of the argument as an angle, expressed in radians.	
	DOUBLE		DOUBLE

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
ATAN2	SYSFUN	Returns the arctangent of *x* and *y* coordinates, specified by the first and second arguments respectively, as an angle, expressed in radians.	
	DOUBLE, DOUBLE		DOUBLE
AVG	SYSIBM	Returns the average of a set of numbers (column function).	
	numeric-type [4]		*numeric-type* [1]
BIGINT	SYSIBM	Returns a 64 bit integer representation of a number or character string in the form of an integer constant.	
	numeric-type		BIGINT
	VARCHAR		BIGINT
BLOB	SYSIBM	Casts from source type to BLOB, with optional length.	
	string-type		BLOB
	string-type, INTEGER		BLOB
CEIL or CEILING	SYSFUN	Returns the smallest integer greater than or equal to the argument.	
	SMALLINT		SMALLINT
	INTEGER		INTEGER
	BIGINT		BIGINT
	DOUBLE		DOUBLE
CHAR	SYSIBM	Returns a string representation of the source type.	
	character-type		CHAR
	character-type, INTEGER		CHAR(*integer*)
	datetime-type		CHAR
	datetime-type, *keyword* [2]		CHAR
	SMALLINT		CHAR(6)
	INTEGER		CHAR(11)
	BIGINT		CHAR(20)
	DECIMAL		CHAR(2+*precision*)
	DECIMAL, VARCHAR		CHAR(2+*precision*)
CHAR	SYSFUN	Returns a character string representation of a floating-point number.	
	DOUBLE		CHAR(24)
CHR	SYSFUN	Returns the character that has the ASCII code value specified by the argument. The value of the argument should be between 0 and 255; otherwise, the return value is null.	
	INTEGER		CHAR(1)
CLOB	SYSIBM	Casts from source type to CLOB, with optional length.	
	character-type		CLOB
	character-type, INTEGER		CLOB

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
COALESCE [3]	SYSIBM	Returns the first non-null argument in the set of arguments.	
	any-type, any-union-compatible-type, ...		*any-type*
CONCAT or ||	SYSIBM	Returns the concatenation of 2 string arguments.	
	string-type, compatible-string-type		*max string-type*
CORRELATION or CORR	SYSIBM	Returns the coefficient of correlation of a set of number pairs.	
	numeric-type, numeric-type		DOUBLE
COS	SYSFUN	Returns the cosine of the argument, where the argument is an angle expressed in radians.	
	DOUBLE		DOUBLE
COT	SYSFUN	Returns the cotangent of the argument, where the argument is an angle expressed in radians.	
	DOUBLE		DOUBLE
COUNT	SYSIBM	Returns the count of the number of rows in a set of rows or values (column function).	
	any-builtin-type [4]		INTEGER
COUNT_BIG	SYSIBM	Returns the number of rows or values in a set of rows or values (column function). Result can be greater than the maximum value of integer.	
	any-builtin-type [4]		DECIMAL(31,0)
COVARIANCE or COVAR	SYSIBM	Returns the covariance of a set of number pairs.	
	numeric-type, numeric-type		DOUBLE
DATE	SYSIBM	Returns a date from a single input value.	
	DATE		DATE
	TIMESTAMP		DATE
	DOUBLE		DATE
	VARCHAR		DATE
DAY	SYSIBM	Returns the day part of a value.	
	VARCHAR		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
DAYNAME	SYSFUN	Returns a mixed case character string containing the name of the day (e.g. Friday) for the day portion of the argument based on what the locale was when db2start was issued.	
	VARCHAR(26)		VARCHAR(100)
	DATE		VARCHAR(100)
	TIMESTAMP		VARCHAR(100)

Table 15. Supported Functions (continued)

Function name	Schema	Description
	Input Parameters	**Returns**
DAYOFWEEK	SYSFUN	Returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Sunday.
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
DAYOFWEEK_ISO	SYSFUN	Returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Monday.
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
DAYOFYEAR	SYSFUN	Returns the day of the year in the argument as an integer value in the range 1-366.
	VARCHAR(26)	INTEGER
	DATE	INTEGER
	TIMESTAMP	INTEGER
DAYS	SYSIBM	Returns an integer representation of a date.
	VARCHAR	INTEGER
	TIMESTAMP	INTEGER
	DATE	INTEGER
DBCLOB	SYSIBM	Casts from source type to DBCLOB, with optional length.
	graphic-type	DBCLOB
	graphic-type, INTEGER	DBCLOB
DECIMAL or DEC	SYSIBM	Returns decimal representation of a number, with optional precision and scale.
	numeric-type	DECIMAL
	numeric-type, INTEGER	DECIMAL
	numeric-type INTEGER, INTEGER	DECIMAL
DECIMAL or DEC	SYSIBM	Returns decimal representation of a character string, with optional precision, scale, and decimal-character.
	VARCHAR	DECIMAL
	VARCHAR, INTEGER	DECIMAL
	VARCHAR, INTEGER, INTEGER	DECIMAL
	VARCHAR, INTEGER, INTEGER, VARCHAR	DECIMAL
DEGREES	SYSFUN	Returns the number of degrees converted from the argument in expressed in radians.
	DOUBLE	DOUBLE

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
DEREF	SYSIBM	Returns an instance of the target type of the reference type argument.	
	REF(*any-structured-type*) *with defined scope*		*any-structured-type (same as input target type)*
DIFFERENCE	SYSFUN	Returns the difference between the sounds of the words in the two argument strings as determined using the SOUNDEX function. A value of 4 means the strings sound the same.	
	VARCHAR(4000), VARCHAR(4000)		INTEGER
DIGITS	SYSIBM	Returns the character string representation of a number.	
	DECIMAL		CHAR
DLCOMMENT	SYSIBM	Returns the comment attribute of a datalink value.	
	DATALINK		VARCHAR(254)
DLLINKTYPE	SYSIBM	Returns the link type attribute of a datalink value.	
	DATALINK		VARCHAR(4)
DLURLCOMPLETE	SYSIBM	Returns the complete URL (including access token) of a datalink value.	
	DATALINK		VARCHAR
DLURLPATH	SYSIBM	Returns the path and file name (including access token) of a datalink value.	
	DATALINK		VARCHAR
DLURLPATHONLY	SYSIBM	Returns the path and file name (without any access token) of a datalink value.	
	DATALINK		VARCHAR
DLURLSCHEME	SYSIBM	Returns the scheme from the URL attribute of a datalink value.	
	DATALINK		VARCHAR
DLURLSERVER	SYSIBM	Returns the server from the URL attribute of a datalink value.	
	DATALINK		VARCHAR
DLVALUE	SYSIBM	Builds a datalink value from a data-location argument, link type argument and optional comment-string argument.	
	VARCHAR		DATALINK
	VARCHAR, VARCHAR		DATALINK
	VARCHAR, VARCHAR, VARCHAR		DATALINK
DOUBLE or DOUBLE_PRECISION	SYSIBM	Returns the floating-point representation of a number.	
	numeric-type		DOUBLE
DOUBLE	SYSFUN	Returns the floating-point number corresponding to the character string representation of a number. Leading and trailing blanks in *argument* are ignored.	
	VARCHAR		DOUBLE
EVENT_MON_STATE	SYSIBM	Returns the operational state of particular event monitor.	
	VARCHAR		INTEGER

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
EXP	SYSFUN	Returns the exponential function of the argument.	
	DOUBLE		DOUBLE
FLOAT	SYSIBM	Same as DOUBLE.	
FLOOR	SYSFUN	Returns the largest integer less than or equal to the argument.	
	SMALLINT		SMALLINT
	INTEGER		INTEGER
	BIGINT		BIGINT
	DOUBLE		DOUBLE
GENERATE_UNIQUE	SYSIBM	Returns a bit data character string that is unique compared to any other execution of the same function.	
	no argument		CHAR(13) FOR BIT DATA
GRAPHIC	SYSIBM	Cast from source type to GRAPHIC, with optional length.	
	graphic-type		GRAPHIC
	graphic-type, INTEGER		GRAPHIC
GROUPING	SYSIBM	Used with grouping-sets and super-groups to indicate sub-total rows generated by a grouping set (column function). The value returned is: **1** The value of the argument in the returned row is a null value and the row was generated for a grouping set. This generated row provides a sub-total for a grouping set. **0** otherwise.	
	any-type		SMALLINT
HEX	SYSIBM	Returns the hexadecimal representation of a value.	
	any-builtin-type		VARCHAR
HOUR	SYSIBM	Returns the hour part of a value.	
	VARCHAR		INTEGER
	TIME		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
INSERT	SYSFUN	Returns a string where *argument3* bytes have been deleted from *argument1* beginning at *argument2* and where *argument4* has been inserted into *argument1* beginning at *argument2*.	
	VARCHAR(4000), INTEGER, INTEGER, VARCHAR(4000)		VARCHAR(4000)
	CLOB(1M), INTEGER, INTEGER, CLOB(1M)		CLOB(1M)
	BLOB(1M), INTEGER, INTEGER, BLOB(1M)		BLOB(1M)

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
INTEGER or INT	SYSIBM	Returns the integer representation of a number.	
	numeric-type		INTEGER
	VARCHAR		INTEGER
JULIAN_DAY	SYSFUN	Returns an integer value representing the number of days from January 1, 4712 B.C. (the start of the Julian date calendar) to the date value specified in the *argument*.	
	VARCHAR(26)		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
LCASE or LOWER	SYSIBM	Returns a string in which all the characters have been converted to lower case characters.	
	CHAR		CHAR
	VARCHAR		VARCHAR
LCASE	SYSFUN	Returns a string in which all the characters have been converted to lower case characters. LCASE will only handle characters in the invariant set. Therefore, LCASE(UCASE(string)) will not necessarily return the same result as LCASE(string).	
	VARCHAR(4000)		VARCHAR(4000)
	CLOB(1M)		CLOB(1M)
LEFT	SYSFUN	Returns a string consisting of the leftmost *argument2* bytes in *argument1*.	
	VARCHAR(4000), INTEGER		VARCHAR(4000)
	CLOB(1M), INTEGER		CLOB(1M)
	BLOB(1M), INTEGER		BLOB(1M)
LENGTH	SYSIBM	Returns the length of the operand in bytes (except for double byte string types which return the length in characters).	
	any-builtin-type		INTEGER
LN	SUSFUN	Returns the natural logarithm of the argument (same as LOG).	
	DOUBLE		DOUBLE
LOCATE	SYSFUN	Returns the starting position of the first occurrence of *argument1* within *argument2*. If the optional third argument is specified, it indicates the character position in *argument2* at which the search is to begin. If *argument1* is not found within *argument2*, the value 0 is returned.	
	VARCHAR(4000), VARCHAR(4000)		INTEGER
	VARCHAR(4000), VARCHAR(4000), INTEGER		INTEGER
	CLOB(1M), CLOB(1M)		INTEGER
	CLOB(1M), CLOB(1M), INTEGER		INTEGER
	BLOB(1M), BLOB(1M)		INTEGER
	BLOB(1M), BLOB(1M), INTEGER		INTEGER

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
LOG	SYSFUN	Returns the natural logarithm of the argument (same as LN).	
	DOUBLE		DOUBLE
LOG10		Returns the base 10 logarithm of the argument.	
	DOUBLE		DOUBLE
LONG_VARCHAR	SYSIBM	Returns a long string.	
	character-type		LONG VARCHAR
LONG_VARGRAPHIC	SYSIBM	Casts from source type to LONG_VARGRAPHIC.	
	graphic-type		LONG VARGRAPHIC
LTRIM	SYSIBM	Returns the characters of the argument with leading blanks removed.	
	CHAR		VARCHAR
	VARCHAR		VARCHAR
	GRAPHIC		VARGRAPHIC
	VARGRAPHIC		VARGRAPHIC
LTRIM	SYSFUN	Returns the characters of the argument with leading blanks removed.	
	VARCHAR(4000)		VARCHAR(4000)
	CLOB(1M)		CLOB(1M)
MAX	SYSIBM	Returns the maximum value in a set of values (column function).	
	any-builtin-type [5]		*same as input type*
MICROSECOND	SYSIBM	Returns the microsecond (time-unit) part of a value.	
	VARCHAR		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
MIDNIGHT_SECONDS	SYSFUN	Returns an integer value in the range 0 to 86 400 representing the number of seconds between midnight and time value specified in the *argument*.	
	VARCHAR(26)		INTEGER
	TIME		INTEGER
	TIMESTAMP		INTEGER
MIN	SYSIBM	Returns the minimum value in a set of values (column function).	
	any-builtin-type [5]		*same as input type*
MINUTE	SYSIBM	Returns the minute part of a value.	
	VARCHAR		INTEGER
	TIME		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
MOD	SYSFUN	Returns the remainder (modulus) of *argument1* divided by *argument2*. The result is negative only if *argument1* is negative.	
	SMALLINT, SMALLINT		SMALLINT
	INTEGER, INTEGER		INTEGER
	BIGINT, BIGINT		BIGINT
MONTH	SYSIBM	Returns the month part of a value.	
	VARCHAR		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
MONTHNAME	SYSFUN	Returns a mixed case character string containing the name of month (e.g. January) for the month portion of the argument that is a date or timestamp, based on what the locale was when the database was started.	
	VARCHAR(26)		VARCHAR(100)
	DATE		VARCHAR(100)
	TIMESTAMP		VARCHAR(100)
NODENUMBER [3]	SYSIBM	Returns the node number of the row. The argument is a column name within a table.	
	any-type		INTEGER
NULLIF [3]	SYSIBM	Returns NULL if the arguments are equal, else returns the first argument.	
	any-type [5], *any-comparable-type*[5]		*any-type*
PARTITION [3]	SYSIBM	Returns the partitioning map index (0 to 4095) of the row. The argument is a column name within a table.	
	any-type		INTEGER
POSSTR	SYSIBM	Returns the position at which one string is contained in another.	
	string-type, *compatible-string-type*		INTEGER
POWER	SYSFUN	Returns the value of *argument1* to the power of *argument2*.	
	INTEGER, INTEGER		INTEGER
	BIGINT, BIGINT		BIGINT
	DOUBLE, INTEGER		DOUBLE
	DOUBLE, DOUBLE		DOUBLE
QUARTER	SYSFUN	Returns an integer value in the range 1 to 4 representing the quarter of the year for the date specified in the argument.	
	VARCHAR(26)		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER

Table 15. Supported Functions (continued)

Function name	Schema	Description
	Input Parameters	**Returns**
RADIANS	SYSFUN	Returns the number of radians converted from argument which is expressed in degrees.
	DOUBLE	DOUBLE
RAISE_ERROR[3]	SYSIBM	Raises an error in the SQLCA. The sqlstate returned is indicated by *argument1*. The second argument contains any text to be returned.
	VARCHAR, VARCHAR	*any-type* [6]
RAND	SYSFUN	Returns a random floating point value between 0 and 1 using the argument as the optional seed value.
	no argument required	DOUBLE
	INTEGER	DOUBLE
REAL	SYSIBM	Returns the single-precision floating-point representation of a number.
	numeric-type	REAL
REGR_AVGX	SYSIBM	Returns quantities used to compute diagnostic statistics.
	numeric-type, numeric-type	DOUBLE
REGR_AVGY	SYSIBM	Returns quantities used to compute diagnostic statistics.
	numeric-type, numeric-type	DOUBLE
REGR_COUNT	SYSIBM	Returns the the number of non-null number pairs used to fit the regression line.
	numeric-type, numeric-type	INTEGER
REGR_INTERCEPT or REGR_ICPT	SYSIBM	Returns the y-intercept of the regression line.
	numeric-type, numeric-type	DOUBLE
REGR_R2	SYSIBM	Returns the coefficient of determination for the regression.
	numeric-type, numeric-type	DOUBE
REGR_SLOPE	SYSIBM	Returns the slope of the line.
	numeric-type, numeric-type	DOUBLE
REGR_SXX	SYSIBM	Returns quantities used to compute diagnostic statistics.
	numeric-type, numeric-type	DOUBLE
REGR_SXY	SYSIBM	Returns quantities used to compute diagnostic statistics.
	numeric-type, numeric-type	DOUBLE
REGR_SYY	SYSIBM	Returns quantities used to compute diagnostic statistics.
	numeric-type, numeric-type	DOUBLE
REPEAT	SYSFUN	Returns a character string composed of *argument1* repeated *argument2* times.
	VARCHAR(4000), INTEGER	VARCHAR(4000)
	CLOB(1M), INTEGER	CLOB(1M)
	BLOB(1M), INTEGER	BLOB(1M)

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
REPLACE	SYSFUN	Replaces all occurrences of *argument2* in *argument1* with *argument3*.	
	VARCHAR(4000), VARCHAR(4000), VARCHAR(4000)		VARCHAR(4000)
	CLOB(1M), CLOB(1M), CLOB(1M)		CLOB(1M)
	BLOB(1M), BLOB(1M), BLOB(1M)		BLOB(1M)
RIGHT	SYSFUN	Returns a string consisting of the rightmost *argument2* bytes in *argument1*.	
	VARCHAR(4000), INTEGER		VARCHAR(4000)
	CLOB(1M), INTEGER		CLOB(1M)
	BLOB(1M), INTEGER		BLOB(1M)
ROUND	SYSFUN	Returns the first argument rounded to *argument2* places **right** of the decimal point. If *argument2* is negative, *argument1* is rounded to the absolute value of *argument2* places to the **left** of the decimal point.	
	INTEGER, INTEGER		INTEGER
	BIGINT, INTEGER		BIGINT
	DOUBLE, INTEGER		DOUBLE
RTRIM	SYSIBM	Returns the characters of the argument with trailing blanks removed.	
	CHAR		VARCHAR
	VARCHAR		VARCHAR
	GRAPHIC		VARGRAPHIC
	VARGRAPHIC		VARGRAPHIC
RTRIM	SYSFUN	Returns the characters of the argument with trailing blanks removed.	
	VARCHAR(4000)		VARCHAR(4000)
	CLOB(1M)		CLOB(1M)
SECOND	SYSIBM	Returns the second (time-unit) part of a value.	
	VARCHAR		INTEGER
	TIME		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
SIGN	SYSFUN	Returns an indicator of the sign of the argument. If the argument is less than zero, -1 is returned. If argument equals zero, 0 is returned. If argument is greater than zero, 1 is returned.	
	SMALLINT		SMALLINT
	INTEGER		INTEGER
	BIGINT		BIGINT
	DOUBLE		DOUBLE

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
SIN	SYSFUN	Returns the sine of the argument, where the argument is an angle expressed in radians.	
	DOUBLE		DOUBLE
SMALLINT	SYSIBM	Returns the small integer representation of a number.	
	numeric-type		SMALLINT
	VARCHAR		SMALLINT
SOUNDEX	SYSFUN	Returns a 4 character code representing the sound of the words in the argument. The result can be used to compare with the sound of other strings. See also DIFFERENCE.	
	VARCHAR(4000)		CHAR(4)
SPACE	SYSFUN	Returns a character string consisting of *argument1* blanks.	
	INTEGER		VARCHAR(4000)
SQLCACHE_SNAPSHOT	SYSFUN	Returns a table of the snapshot of the db2 dynamic SQL statement cache.	
	Refer to "SQLCACHE_SNAPSHOT" on page 390.		
SQRT	SYSFUN	Returns the square root of the argument.	
	DOUBLE		DOUBLE
STDDEV	SYSIBM	Returns the standard deviation of a set of numbers (column function).	
	DOUBLE		DOUBLE
SUBSTR	SYSIBM	Returns a substring of a string *argument1* starting at *argument2* for *argument3* characters. If *argument3* is not specified, the remainder of the string is assumed.	
	string-type, INTEGER		*string-type*
	string-type, INTEGER, INTEGER		*string-type*
SUM	SYSIBM	Returns the sum of a set of numbers (column function).	
	numeric-type [4]		*max-numeric-type* [1]
TABLE_NAME	SYSIBM	Returns an unqualified name of a table or view based on the object name given in *argument1* and the optional schema name given in *argument2*. It is used to resolve aliases.	
	VARCHAR		VARCHAR(128)
	VARCHAR, VARCHAR		VARCHAR(128)
TABLE_SCHEMA	SYSIBM	Returns the schema name portion of the two part table or view name given by the object name in *argument1* and the optional schema name in *argument2*. It is used to resolve aliases.	
	VARCHAR		VARCHAR(128)
	VARCHAR, VARCHAR		VARCHAR(128)
TAN	SYSFUN	Returns the tangent of the argument, where the argument is an angle expressed in radians.	
	DOUBLE		DOUBLE

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
TIME	SYSIBM	Returns a time from a value.	
	TIME		TIME
	TIMESTAMP		TIME
	VARCHAR		TIME
TIMESTAMP	SYSIBM	Returns a timestamp from a value or a pair of values.	
	TIMESTAMP		TIMESTAMP
	VARCHAR		TIMESTAMP
	VARCHAR, VARCHAR		TIMESTAMP
	VARCHAR, TIME		TIMESTAMP
	DATE, VARCHAR		TIMESTAMP
	DATE, TIME		TIMESTAMP
TIMESTAMP_ISO	SYSFUN	Returns a timestamp value based on a date, time, or timestamp argument. If the argument is a date, it inserts zero for all the time elements. If the argument is a time, it inserts the value of CURRENT DATE for the date elements and zero for the fractional time element.	
	DATE		TIMESTAMP
	TIME		TIMESTAMP
	TIMESTAMP		TIMESTAMP
	VARCHAR(26)		TIMESTAMP
TIMESTAMPDIFF	SYSFUN	Returns an estimated number of intervals of type *argument1* based on the difference between two timestamps. The second argument is the result of subtracting two timestamp types and converting the result to CHAR. Valid values of interval (*argument1*) are: **1** Fractions of a second **2** Seconds **4** Minutes **8** Hours **16** Days **32** Weeks **64** Months **128** Quarters **256** Years	
	INTEGER, CHAR(22)		INTEGER

Table 15. Supported Functions (continued)

Function name	Schema	Description
	Input Parameters	**Returns**
TRANSLATE	SYSIBM	Returns a string in which one or more characters may have been translated into other characters.
	CHAR	CHAR
	VARCHAR	VARCHAR
	CHAR, VARCHAR, VARCHAR	CHAR
	VARCHAR, VARCHAR, VARCHAR	VARCHAR
	CHAR, VARCHAR, VARCHAR, VARCHAR	CHAR
	VARCHAR, VARCHAR, VARCHAR, VARCHAR	VARCHAR
	GRAPHIC, VARGRAPHIC, VARGRAPHIC	GRAPHIC
	VARGRAPHIC, VARGRAPHIC, VARGRAPHIC	VARGRAPHIC
	GRAPHIC, VARGRAPHIC, VARGRAPHIC, VARGRAPHIC	GRAPHIC
	VARGRAPHIC, VARGRAPHIC, VARGRAPHIC, VARGRAPHIC	VARGRAPHIC
TRUNC or TRUNCATE	SYSFUN	Returns *argument1* truncated to *argument2* places **right** of the decimal point. If *argument2* is negative, *argument1* is truncated to the absolute value of *argument2* places to the **left** of the decimal point.
	INTEGER, INTEGER	INTEGER
	BIGINT, INTEGER	BIGINT
	DOUBLE, INTEGER	DOUBLE
TYPE_ID [3]	SYSIBM	Returns the internal data type identifier of the dynamic data type of the argument. Note that the result of this function is not portable across databases.
	any-structured-type	INTEGER
TYPE_NAME [3]	SYSIBM	Returns the unqualified name of the dynamic data type of the argument.
	any-structured-type	VARCHAR(18)
TYPE_SCHEMA [3]	SYSIBM	Returns the schema name of the dynamic type of the argument.
	any-structured-type	VARCHAR(128)
UCASE or UPPER	SYSIBM	Returns a string in which all the characters have been converted to upper case characters.
	CHAR	CHAR
	VARCHAR	VARCHAR
UCASE	SYSFUN	Returns a string in which all the characters have been converted to upper case characters.
	VARCHAR	VARCHAR
VALUE [3]	SYSIBM	Same as COALESCE.

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
VARCHAR	SYSIBM	Returns a VARCHAR representation of the first argument. If a second argument is present, it specifies the length of the result.	
	character-type		VARCHAR
	character-type, INTEGER		VARCHAR
	datetime-type		VARCHAR
VARGRAPHIC	SYSIBM	Returns a VARGRAPHIC representation of the first argument. If a second argument is present, it specifies the length of the result.	
	graphic-type		VARGRAPHIC
	graphic-type, INTEGER		VARGRAPHIC
	VARCHAR		VARGRAPHIC
VARIANCE or VAR	SYSIBM	Returns the variance of a set of numbers (column function).	
	DOUBLE		DOUBLE
WEEK	SYSFUN	Returns the week of the year in of the argument as an integer value in the range of 1-54.	
	VARCHAR(26)		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
WEEK_ISO	SYSFUN	Returns the week of the year in of the argument as an integer value in the range of 1-53. The first day of a week is Monday. Week 1 is the first week of the year to contain a Thursday.	
	VARCHAR(26)		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
YEAR	SYSIBM	Returns the year part of a value.	
	VARCHAR		INTEGER
	DATE		INTEGER
	TIMESTAMP		INTEGER
	DECIMAL		INTEGER
"+"	SYSIBM	Adds two numeric operands.	
	numeric-type, *numeric-type*		*max numeric-type*
"+"	SYSIBM	Unary plus operator.	
	numeric-type		*numeric-type*

Table 15. Supported Functions (continued)

Function name	Schema	Description	
	Input Parameters		**Returns**
"+"	SYSIBM	Datetime plus operator.	
	DATE, DECIMAL(8,0)		DATE
	TIME, DECIMAL(6,0)		TIME
	TIMESTAMP, DECIMAL(20,6)		TIMESTAMP
	DECIMAL(8,0), DATE		DATE
	DECIMAL(6,0), TIME		TIME
	DECIMAL(20,6), TIMESTAMP		TIMESTAMP
	datetime-type, DOUBLE, *labeled-duration-code*		*datetime-type*
"-"	SYSIBM	Subtracts two numeric operands.	
	numeric-type, *numeric-type*		*max numeric-type*
"-"	SYSIBM	Unary minus operator.	
	numeric-type		*numeric-type* [1]
"-"	SYSIBM	Datetime minus operator.	
	DATE, DATE		DECIMAL(8,0)
	TIME, TIME		DECIMAL(6,0)
	TIMESTAMP, TIMESTAMP		DECIMAL(20,6)
	DATE, VARCHAR		DECIMAL(8,0)
	TIME, VARCHAR		DECIMAL(6,0)
	TIMESTAMP, VARCHAR		DECIMAL(20,6)
	VARCHAR, DATE		DECIMAL(8,0)
	VARCHAR, TIME		DECIMAL(6,0)
	VARCHAR, TIMESTAMP		DECIMAL(20,6)
	DATE, DECIMAL(8,0)		DATE
	TIME, DECIMAL(6,0)		TIME
	TIMESTAMP, DECIMAL(20,6)		TIMESTAMP
	datetime-type, DOUBLE, *labeled-duration-code*		*datetime-type*
"*"	SYSIBM	Multiplies two numeric operands.	
	numeric-type, *numeric-type*		*max numeric-type*
"/"	SYSIBM	Divides two numeric operands.	
	numeric-type, *numeric-type*		*max numeric-type*
"||"	SYSIBM	Same as CONCAT.	

Notes

- References to string data types that are not qualified by a length should be assumed to support the maximum length for the data type
- References to a DECIMAL data type without precision and scale should be assumed to allow any supported precision and scale.

Key to Table

any-builtin-type Any data type that is not a distinct type.

any-type Any type defined to the database.

any-structured-type
Any user-defined structured type defined to the database.

any-comparable-type
Any type that is comparable with other argument types as defined in "Assignments and Comparisons" on page 94.

any-union-compatible-type
Any type that is compatible with other argument types as defined in "Rules for Result Data Types" on page 107.

character-type Any of the character string types: CHAR, VARCHAR, LONG VARCHAR, CLOB.

compatible-string-type
A string type that comes from the same grouping as the other argument (e.g. if one argument is a *character-type* the other must also be a *character-type*).

datetime-type Any of the datetime types: DATE, TIME, TIMESTAMP.

graphic-type Any of the double byte character string types: GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, DBCLOB.

labeled-duration-code
As a type this is a SMALLINT. If the function is invoked using the infix form of the plus or minus operator, labeled-durations as defined in "Labeled Durations" on page 164 can be used. For a source function that does not use the plus or minus operator character as the name, the following values must be used for the labeled-duration-code argument when invoking the function.

1	YEAR or YEARS
2	MONTH or MONTHS
3	DAY or DAYS
4	HOUR or HOURS
5	MINUTE or MINUTES
6	SECOND or SECONDS
7	MICROSECOND or MICROSECONDS

LOB-type Any of the large object types: BLOB, CLOB, DBCLOB.

max-numeric-type The maximum numeric type of the arguments where maximum is defined as the rightmost *numeric-type*.

max-string-type The maximum string type of the arguments where maximum is defined as the rightmost *character-type* or *graphic-type*. If arguments are BLOB, the *max-string-type* is BLOB.

numeric-type Any of the numeric types: SMALLINT, INTEGER, BIGINT, DECIMAL, REAL, DOUBLE.

string-type Any type from *character type*, *graphic-type* or BLOB.

Table Footnotes

1. When the input parameter is SMALLINT, the result type is INTEGER. When the input parameter is REAL, the result type is DOUBLE.
2. Keywords allowed are ISO, USA, EUR, JIS, and LOCAL. This function signature is not supported as a sourced function.
3. This function cannot be used as a source function.
4. The keyword ALL or DISTINCT may be used before the first parameter. If DISTINCT is specified, the use of user-defined structured types, long string types or a DATALINK type is not supported.
5. The use of user-defined structured types, long string types or a DATALINK type is not supported.
6. The type returned by RAISE_ERROR depends upon the context of its use. RAISE_ERROR, if not cast to a particular type, will return a type appropriate to its invocation within a CASE expression.

Column Functions

The argument of a column function is a set of values derived from an expression. The expression may include columns but cannot include a *scalar-fullselect* or another column function (SQLSTATE 42607). The scope of the set is a group or an intermediate result table as explained in "Chapter 5. Queries" on page 393.

If a GROUP BY clause is specified in a query and the intermediate result from the FROM, WHERE, GROUP BY and HAVING clauses is the empty set; then the column functions are not applied, the result of the query is the empty set, the SQLCODE is set to +100 and the SQLSTATE is set to '02000'.

If a GROUP BY clause is not specified in a query and the intermediate result is of the FROM, WHERE, and HAVING clauses is the empty set, then the column functions are applied to the empty set.

For example, the result of the following SELECT statement is the number of distinct values of JOBCODE for employees in department D01:

```
SELECT COUNT(DISTINCT JOBCODE)
  FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT = 'D01'
```

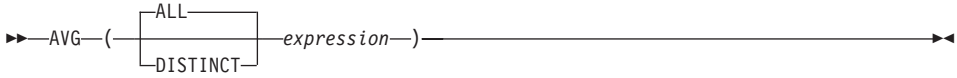
The keyword DISTINCT is not considered an argument of the function, but rather a specification of an operation that is performed before the function is applied. If DISTINCT is specified, duplicate values are eliminated. If ALL is implicitly or explicitly specified, duplicate values are not eliminated.

Expressions can be used in column functions, for example:

```
SELECT MAX(BONUS + 1000)
  INTO :TOP_SALESREP_BONUS
  FROM EMPLOYEE
  WHERE COMM > 5000
```

The column functions that follow are in the SYSIBM schema and may be qualified with the schema name (for example, SYSIBM.COUNT(*)).

AVG



The schema is SYSIBM.

The AVG function returns the average of a set of numbers.

The argument values must be numbers and their sum must be within the range of the data type of the result. The result can be null.

The data type of the result is the same as the data type of the argument values, except that:

- The result is a large integer if the argument values are small integers.
- The result is double-precision floating point if the argument values are single-precision floating point.

If the data type of the argument values is decimal with precision *p* and scale *s*, the precision of the result is 31 and the scale is 31-*p*+*s*.

The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are eliminated.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the average value of the set.

If the type of the result is integer, the fractional part of the average is lost.

Examples:

- Using the PROJECT table, set the host variable AVERAGE (decimal(5,2)) to the average staffing level (PRSTAFF) of projects in department (DEPTNO) 'D11'.

```
SELECT AVG(PRSTAFF)
  INTO :AVERAGE
  FROM PROJECT
  WHERE DEPTNO = 'D11'
```

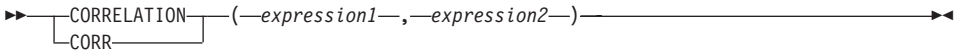
  Results in AVERAGE being set to 4.25 (that is 17/4) when using the sample table.

- Using the PROJECT table, set the host variable ANY_CALC (decimal(5,2)) to the average of each unique staffing level value (PRSTAFF) of projects in department (DEPTNO) 'D11'.

```
SELECT AVG(DISTINCT PRSTAFF)
  INTO :ANY_CALC
  FROM PROJECT
  WHERE DEPTNO = 'D11'
```

  Results in ANY_CALC being set to 4.66 (that is 14/3) when using the sample table.

CORRELATION



The schema is SYSIBM.

The CORRELATION function returns the coefficient of correlation of a set of number pairs.

The argument values must be numbers.

The data type of the result is double-precision floating point. The result can be null. When not null, the result is between 0 and 1.

The function is applied to the set of (*expression1*, *expression2*) pairs derived from the argument values by the elimination of all pairs for which either *expression1* or *expression2* is null.

If the function is applied to an empty set, or if either STDDEV(*expression1*) or STDDEV(*expression2*) is equal to zero, the result is a null value. Otherwise, the result is the correlation coefficient for the value pairs in the set. The result is equivalent to the following expression:

```
COVARIANCE(expression1,expression2)/(STDDEV(expression1)*STDDEV(expression2))
```

The order in which the values are aggregated is undefined, but every intermediate result must be within the range of the result data type.

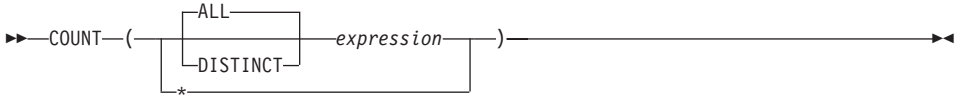
Example:

- Using the EMPLOYEE table, set the host variable CORRLN (double precision floating point) to the correlation between salary and bonus for those employees in department (WORKDEPT) 'A00'.

  ```
  SELECT CORRELATION(SALARY, BONUS)
    INTO :CORRLN
    FROM EMPLOYEE
    WHERE WORKDEPT = 'A00'
  ```

  CORRLN is set to approximately 9.99853953399538E-001 when using the sample table.

COUNT



The schema is SYSIBM.

The COUNT function returns the number of rows or values in a set of rows or values.

If DISTINCT is used, the resulting data type of *expression* must not have a length greater than 255 for a character column or 127 for a graphic column. The data type of *expression* cannot be a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

The result of the function is a large integer. The result cannot be null.

The argument of COUNT(*) is a set of rows. The result is the number of rows in the set. A row that includes only NULL values is included in the count.

The argument of COUNT(DISTINCT *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null and duplicate values. The result is the number of different non-null values in the set.

The argument of COUNT(*expression*) or COUNT(ALL *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null values. The result is the number of non-null values in the set, including duplicates.

Examples:

- Using the EMPLOYEE table, set the host variable FEMALE (int) to the number of rows where the value of the SEX column is 'F'.

```
SELECT COUNT(*)
  INTO :FEMALE
  FROM EMPLOYEE
  WHERE SEX = 'F'
```

  Results in FEMALE being set to 13 when using the sample table.
- Using the EMPLOYEE table, set the host variable FEMALE_IN_DEPT (int) to the number of departments (WORKDEPT) that have at least one female as a member.

```
SELECT COUNT(DISTINCT WORKDEPT)
  INTO :FEMALE_IN_DEPT
  FROM EMPLOYEE
  WHERE SEX = 'F'
```

Results in FEMALE_IN_DEPT being set to 5 when using the sample table. (There is at least one female in departments A00, C01, D11, D21, and E11.)

COUNT_BIG

The schema is SYSIBM.

The COUNT_BIG function returns the number of rows or values in a set of rows or values. It is similar to COUNT except that the result can be greater than the maximum value of integer.

If DISTINCT is used, the resulting data type of *expression* must not have a length greater than 255 for a character column or 127 for a graphic column. The resulting data type of *expression* cannot be a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

The result of the function is a decimal with precision 31 and scale 0. The result cannot be null.

The argument of COUNT_BIG(*) is a set of rows. The result is the number of rows in the set. A row that includes only NULL values is included in the count.

The argument of COUNT_BIG(DISTINCT *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null and duplicate values. The result is the number of different non-null values in the set.

The argument of COUNT_BIG(*expression*) or COUNT_BIG(ALL *expression*) is a set of values. The function is applied to the set of values derived from the argument values by the elimination of null values. The result is the number of non-null values in the set, including duplicates.

Examples:

- Refer to COUNT examples and substitute COUNT_BIG for occurrences of COUNT. The results are the same except for the data type of the result.
- Some applications may require the use of COUNT but need to support values larger than the largest integer. This can be achieved by use of sourced user-defined functions and setting the SQL path. The following series of statements shows how to create a sourced function to support COUNT(*) based on COUNT_BIG and returning a decimal value with a

precision of 15. The SQL path is set such that the sourced function based on COUNT_BIG is used in subsequent statements such as the query shown.

```
CREATE FUNCTION RICK.COUNT() RETURNS DECIMAL(15,0)
       SOURCE SYSIBM.COUNT_BIG();
SET CURRENT FUNCTION PATH RICK, SYSTEM PATH;
SELECT COUNT(*) FROM EMPLOYEE;
```

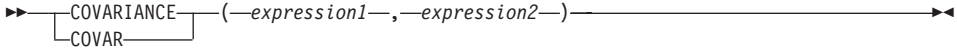
Note how the sourced function is defined with no parameters to support COUNT(*). This only works if you name the function COUNT and do not qualify the function with the schema name when it is used. To get the same effect as COUNT(*) with a name other than COUNT, invoke the function with no parameters. Thus, if RICK.COUNT had been defined as RICK.MYCOUNT instead, the query would have to be written as follows:

```
SELECT MYCOUNT() FROM EMPLOYEE;
```

If the count is taken on a specific column, the sourced function must specify the type of the column. The following statements created a sourced function that will take any CHAR column as a argument and use COUNT_BIG to perform the counting.

```
CREATE FUNCTION RICK.COUNT(CHAR()) RETURNS DOUBLE
       SOURCE SYSIBM.COUNT_BIG(CHAR());
SELECT COUNT(DISTINCT WORKDEPT) FROM EMPLOYEE;
```

COVARIANCE



The schema is SYSIBM.

The COVARIANCE function returns the (population) covariance of a set of number pairs.

The argument values must be numbers.

The data type of the result is double-precision floating point. The result can be null.

The function is applied to the set of (*expression1*,*expression2*) pairs derived from the argument values by the elimination of all pairs for which either *expression1* or *expression2* is null.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the covariance of the value pairs in the set. The result is equivalent to the following:

1. Let avgexp1 be the result of AVG(*expression1*) and let avgexp2 be the result of AVG(*expression2*).
2. The result of COVARIANCE(*expression1*, *expression2*) is AVG((*expression1* - avgexp1) * (*expression2* - avgexp2)

The order in which the values are aggregated is undefined, but every intermediate result must be within the range of the result data type.

Example:

- Using the EMPLOYEE table, set the host variable COVARNCE (double precision floating point) to the covariance between salary and bonus for those employees in department (WORKDEPT) 'A00'.

```
SELECT COVARIANCE(SALARY, BONUS)
  INTO :COVARNCE
  FROM EMPLOYEE
  WHERE WORKDEPT = 'A00'
```

  COVARNCE is set to approximately 1.68888888888889E+006 when using the sample table.

GROUPING

The schema is SYSIBM.

Used in conjunction with grouping-sets and super-groups (see "group-by-clause" on page 409 for details), the GROUPING function returns a value which indicates whether or not a row returned in a GROUP BY answer set is a row generated by a grouping set that excludes the column represented by *expression*.

The argument can be of any type, but must be an item of a GROUP BY clause.

The result of the function is a small integer. It is set to one of the following values:

1 The value of *expression* in the returned row is a null value, and the row was generated by the super-group. This generated row can be used to provide sub-total values for the GROUP BY expression.

0 The value is other than the above.

Example:

The following query:

```
SELECT SALES_DATE,
       SALES_PERSON,
       SUM(SALES) AS UNITS_SOLD,
       GROUPING(SALES_DATE) AS DATE_GROUP,
       GROUPING(SALES_PERSON) AS  SALES_GROUP
  FROM SALES
  GROUP BY CUBE (SALES_DATE, SALES_PERSON)
  ORDER BY SALES_DATE, SALES_PERSON
```

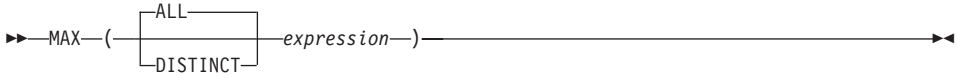
results in:

```
SALES_DATE SALES_PERSON    UNITS_SOLD  DATE_GROUP  SALES_GROUP
---------- --------------- ----------- ----------- -----------
12/31/1995 GOUNOT                    1           0           0
12/31/1995 LEE                       6           0           0
12/31/1995 LUCCHESSI                 1           0           0
12/31/1995 -                         8           0           1
03/29/1996 GOUNOT                   11           0           0
03/29/1996 LEE                      12           0           0
03/29/1996 LUCCHESSI                 4           0           0
03/29/1996 -                        27           0           1
```

```
03/30/1996 GOUNOT                21          0          0
03/30/1996 LEE                   21          0          0
03/30/1996 LUCCHESSI              4          0          0
03/30/1996 -                     46          0          1
03/31/1996 GOUNOT                 3          0          0
03/31/1996 LEE                   27          0          0
03/31/1996 LUCCHESSI              1          0          0
03/31/1996 -                     31          0          1
04/01/1996 GOUNOT                14          0          0
04/01/1996 LEE                   25          0          0
04/01/1996 LUCCHESSI              4          0          0
04/01/1996 -                     43          0          1
-          GOUNOT                50          1          0
-          LEE                   91          1          0
-          LUCCHESSI             14          1          0
-          -                    155          1          1
```

An application can recognize a SALES_DATE sub-total row by the fact that the value of DATE_GROUP is 0 and the value of SALES_GROUP is 1. A SALES_PERSON sub-total row can be recognized by the fact that the value of DATE_GROUP is 1 and the value of SALES_GROUP is 0. A grand total row can be recognized by the value 1 for both DATE_GROUP and SALES_GROUP.

MAX



The schema is SYSIBM.

The MAX function returns the maximum value in a set of values.

The argument values can be of any built-in type other than a long string or DATALINK.

If DISTINCT is used, the resulting data type of *expression* must not have a length greater than 255 for a character column or 127 for a graphic column. The resulting data type of *expression* cannot be a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

The data type, length and code page of the result are the same as the data type, length and code page of the argument values. The result is considered to be a derived value and can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the maximum value in the set.

The specification of DISTINCT has no effect on the result and therefore is not recommended. It is included for compatibility with other relational systems.

Examples:

- Using the EMPLOYEE table, set the host variable MAX_SALARY (decimal(7,2)) to the maximum monthly salary (SALARY/12) value.

  ```
  SELECT MAX(SALARY) / 12
    INTO :MAX_SALARY
    FROM EMPLOYEE
  ```

  Results in MAX_SALARY being set to 4395.83 when using the sample table.

- Using the PROJECT table, set the host variable LAST_PROJ(char(24)) to the project name (PROJNAME) that comes last in the collating sequence.

  ```
  SELECT MAX(PROJNAME)
    INTO :LAST_PROJ
    FROM PROJECT
  ```

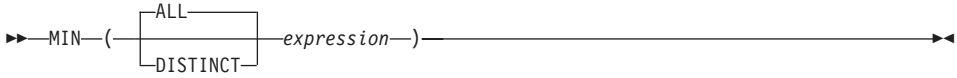
Results in LAST_PROJ being set to 'WELD LINE PLANNING' when using the sample table.

- Similar to the previous example, set the host variable LAST_PROJ (char(40)) to the project name that comes last in the collating sequence when a project name is concatenated with the host variable PROJSUPP. PROJSUPP is '_Support'; it has a char(8) data type.

```
SELECT MAX(PROJNAME CONCAT PROJSUPP)
  INTO :LAST_PROJ
  FROM PROJECT
```

  Results in LAST_PROJ being set to 'WELD LINE PLANNING_SUPPORT' when using the sample table.

MIN



The schema is SYSIBM.

The MIN function returns the minimum value in a set of values.

The argument values can be of any built-in type other than a long string or DATALINK.

If DISTINCT is used, the resulting data type of *expression* must not have a length greater than 255 for a character column or 127 for a graphic column. The resulting data type of *expression* cannot be a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

The data type, length, and code page of the result are the same as the data type, length, and code page of the argument values. The result is considered to be a derived value and can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values.

If this function is applied to an empty set, the result of the function is a null value. Otherwise, the result is the minimum value in the set.

The specification of DISTINCT has no effect on the result and therefore is not recommended. It is included for compatibility with other relational systems.

Examples:

- Using the EMPLOYEE table, set the host variable COMM_SPREAD (decimal(7,2)) to the difference between the maximum and minimum commission (COMM) for the members of department (WORKDEPT) 'D11'.

  ```
  SELECT MAX(COMM) - MIN(COMM)
    INTO :COMM_SPREAD
    FROM EMPLOYEE
    WHERE WORKDEPT = 'D11'
  ```

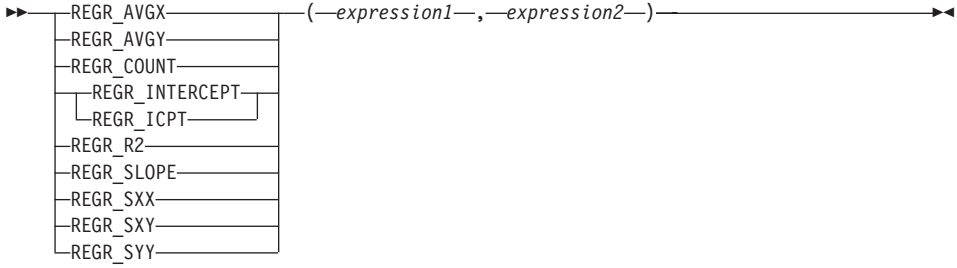
  Results in COMM_SPREAD being set to 1118 (that is, 2580 - 1462) when using the sample table.

- Using the PROJECT table, set the host variable (FIRST_FINISHED (char(10)) to the estimated ending date (PRENDATE) of the first project scheduled to be completed.

```
SELECT MIN(PRENDATE)
  INTO :FIRST_FINISHED
  FROM PROJECT
```

  Results in FIRST_FINISHED being set to '1982-09-15' when using the sample table.

REGRESSION Functions



The schema is SYSIBM.

The regression functions support the fitting of an ordinary-least-squares regression line of the form y = a * x + b to a set of number pairs. The first element of each pair (*expression1*) is interpreted as a value of the dependent variable (i.e., a ″y value″). The second element of each pair (*expression2*) is interpreted as a value of the independent variable (i.e., an ″x value″).

The function REGR_COUNT returns the number of non-null number pairs used to fit the regression line (see below).

The function REGR_INTERCEPT (the short form is REGR_ICPT) returns the y-intercept of the regression line (″b″ in the above equation)

The function REGR_R2 returns the coefficient of determination (also called ″R-squared″ or ″goodness-of-fit″) for the regression.

The function REGR_SLOPE returns the slope of the line (the parameter ″a″ in the above equation).

The functions REGR_AVGX, REGR_AVGY, REGR_SXX, REGR_SYY, and REGR_SXY return quantities that can be used to compute various diagnostic statistics needed for the evaluation of the quality and statistical validity of the regression model (see below).

The argument values must be numbers.

The data type of the result of REGR_COUNT is integer. For the remaining functions, the data type of the result is double-precision floating point. The result can be null. When not null, the result of REGR_R2 is between 0 and 1 and the result of both REGR_SXX and REGR_SYY is non-negative.

Each function is applied to the set of (*expression1*, *expression2*) pairs derived from the argument values by the elimination of all pairs for which either *expression1* or e*xpression2* is null.

If the set is not empty and VARIANCE(*expression2*) is positive, REGR_COUNT returns the number of non-null pairs in the set, and the remaining functions return results that are defined as follows:

```
REGR_SLOPE(expression1,expression2) =
 COVARIANCE(expression1,expression2)/VARIANCE(expression2)

REGR_INTERCEPT(expression1, expression2) =
  AVG(expression1) - REGR_SLOPE(expression1, expression2) * AVG(expression2)

REGR_R2(expression1, expression2) =
  POWER(CORRELATION(expression1, expression2), 2) if VARIANCE(expression1)>0
  REGR_R2(expression1, expression2) = 1 if VARIANCE(expression1)=0

REGR_AVGX(expression1, expression2) = AVG(expression2)

REGR_AVGY(expression1, expression2) = AVG(expression1)

REGR_SXX(expression1, expression2) =
  REGR_COUNT(expression1, expression2) * VARIANCE(expression2)

REGR_SYY(expression1, expression2) =
  REGR_COUNT(expression1, expression2) * VARIANCE(expression1)

REGR_SXY(expression1, expression2) =
  REGR_COUNT(expression1, expression2) * COVARIANCE(expression1, expression2)
```

If the set is not empty and VARIANCE(*expression2*) is equal to zero, then the regression line either has infinite slope or is undefined. In this case, the functions REGR_SLOPE, REGR_INTERCEPT, and REGR_R2 each return a null value, and the remaining functions return values as defined above. If the set is empty, REGR_COUNT returns zero and the remaining functions return a null value.

The order in which the values are aggregated is undefined, but every intermediate result must be within the range of the result data type.

The regression functions are all computed simultaneously during a single pass through the data. In general, it is more efficient to use the regression functions to compute the statistics needed for a regression analysis than to perform the equivalent computations using ordinary column functions such as AVERAGE, VARIANCE, COVARIANCE, and so forth.

The usual diagnostic statistics that accompany a linear-regression analysis can be computed in terms of the above functions. For example:

Adjusted R2

```
1 - ( (1 - REGR_R2) * ((REGR_COUNT - 1) / (REGR_COUNT - 2)) )
```

Standard error
SQRT((REGR_SYY-
(POWER(REGR_SXY,2)/REGR_SXX))/(REGR_COUNT-2))

Total sum of squares
REGR_SYY

Regression sum of squares
POWER(REGR_SXY,2) / REGR_SXX

Residual sum of squares
(Total sum of squares)-(Regression sum of squares)

t statistic for slope
REGR_SLOPE * SQRT(REGR_SXX) / (Standard error)

t statistic for y-intercept
REGR_INTERCEPT/((Standard error) *
SQRT((1/REGR_COUNT)+(POWER(REGR_AVGX,2)/REGR_SXX))

Example:

- Using the EMPLOYEE table, compute an ordinary-least-squares regression line that expresses the bonus of an employee in department (WORKDEPT) 'A00' as a linear function of the employee's salary. Set the host variables SLOPE, ICPT, RSQR (double precision floating point) to the slope, intercept, and coefficient of determination of the regression line, respectively. Also set the host variables AVGSAL and AVGBONUS to the average salary and average bonus, respectively, of the employees in department 'A00', and set the host variable CNT (integer) to the number of employees in department 'A00' for whom both salary and bonus data are available. Store the remaining regression statistics in host variables SXX, SYY, and SXY.

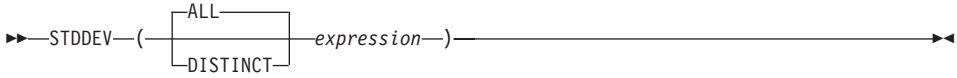
```
SELECT REGR_SLOPE(BONUS,SALARY), REGR_INTERCEPT(BONUS,SALARY),
   REGR_R2(BONUS,SALARY), REGR_COUNT(BONUS,SALARY),
   REGR_AVGX(BONUS,SALARY), REGR_AVGY(BONUS,SALARY),
   REGR_SXX(BONUS,SALARY), REGR_SYY(BONUS,SALARY),
   REGR_SXY(BONUS,SALARY)
   INTO :SLOPE, :ICPT,
        :RSQR, :CNT,
        :AVGSAL, :AVGBONUS,
        :SXX, :SYY,
        :SXY
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
```

When using the sample table, the host variables are set to the following approximate values:

```
SLOPE:  +1.71002671916749E-002
ICPT:  +1.00871888623260E+002
RSQR:  +9.99707928128685E-001
```

```
CNT:  3
AVGSAL:  +4.28333333333333E+004
AVGBONUS:  +8.33333333333333E+002
SXX:  +2.96291666666667E+008
SYY:  +8.66666666666667E+004
SXY:  +5.06666666666667E+006
```

STDDEV



The schema is SYSIBM.

The STDDEV function returns the standard deviation of a set of numbers.

The argument values must be numbers.

The data type of the result is double-precision floating point. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are eliminated.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the standard deviation of the values in the set.

The order in which the values are aggregated is undefined, but every intermediate result must be within the range of the result data type.

Example:

- Using the EMPLOYEE table, set the host variable DEV (double precision floating point) to the standard deviation of the salaries for those employees in department (WORKDEPT) 'A00'.

```
SELECT STDDEV(SALARY)
  INTO :DEV
  FROM EMPLOYEE
  WHERE WORKDEPT = 'A00'
```

  Results in DEV being set to approximately 9938.00 when using the sample table.

SUM

The schema is SYSIBM.

The SUM function returns the sum of a set of numbers.

The argument values must be numbers (built-in types only) and their sum must be within the range of the data type of the result.

The data type of the result is the same as the data type of the argument values except that:

- The result is a large integer if the argument values are small integers.
- The result is double-precision floating point if the argument values are single-precision floating point.

If the data type of the argument values is decimal, the precision of the result is 31 and the scale is the same as the scale of the argument values. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are also eliminated.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the sum of the values in the set.

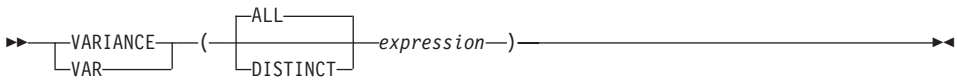
Example:

- Using the EMPLOYEE table, set the host variable JOB_BONUS (decimal(9,2)) to the total bonus (BONUS) paid to clerks (JOB='CLERK').

```
SELECT SUM(BONUS)
  INTO :JOB_BONUS
  FROM EMPLOYEE
  WHERE JOB = 'CLERK'
```

  Results in JOB_BONUS being set to 2800 when using the sample table.

VARIANCE



The schema is SYSIBM.

The VARIANCE function returns the variance of a set of numbers.

The argument values must be numbers.

The data type of the result is double-precision floating point. The result can be null.

The function is applied to the set of values derived from the argument values by the elimination of null values. If DISTINCT is specified, duplicate values are eliminated.

If the function is applied to an empty set, the result is a null value. Otherwise, the result is the variance of the values in the set.

The order in which the values are added is undefined, but every intermediate result must be within the range of the result data type.

Example:

- Using the EMPLOYEE table, set the host variable VARNCE (double precision floating point) to the variance of the salaries for those employees in department (WORKDEPT) 'A00'.

  ```
  SELECT VARIANCE(SALARY)
    INTO :VARNCE
    FROM EMPLOYEE
    WHERE WORKDEPT = 'A00'
  ```

  Results in VARNCE being set to approximately 98763888.88 when using the sample table.

Scalar Functions

A scalar function can be used wherever an expression can be used. However, the restrictions that apply to the use of expressions and column functions also apply when an expression or column function is used within a scalar function. For example, the argument of a scalar function can be a column function only if a column function is allowed in the context in which the scalar function is used.

The restrictions on the use of column functions do not apply to scalar functions because a scalar function is applied to a single value rather than a set of values.

Example: The result of the following SELECT statement has as many rows as there are employees in department D01:

```
SELECT EMPNO, LASTNAME, YEAR(CURRENT DATE - BRTHDATE)
  FROM EMPLOYEE
  WHERE WORKDEPT = 'D01'
```

The scalar functions that follow may be qualified with the schema name (for example, SYSIBM.CHAR(123)).

ABS or ABSVAL

The schema is SYSFUN.

Returns the absolute value of the argument.

The argument can be of any built-in numeric data type. If it is of type DECIMAL or REAL, it is converted to a double-precision floating-point number for processing by the function.

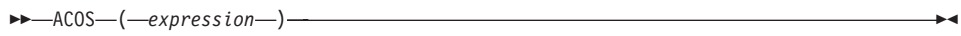
The result of the function is:

- SMALLINT if the argument is SMALLINT
- INTEGER if the argument is INTEGER
- BIGINT if the argument is BIGINT
- DOUBLE if the argument is DOUBLE, DECIMAL or REAL [39].

The result can be null; if the argument is null, the result is the null value.

39. Also returns DOUBLE when the argument is the smallest value of BIGINT, –9 223 372 036 854 775 808.

ACOS



The schema is SYSFUN.

Returns the arccosine of the argument as an angle expressed in radians.

The argument can be of any built-in numeric data type. It is converted to a double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

ASCII

The schema is SYSFUN.

Returns the ASCII code value of the leftmost character of the argument as an integer.

The argument can be of any built-in character string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB the maximum length is 1 048 576 bytes. LONG VARCHAR is converted to CLOB for processing by the function.

The result of the function is always INTEGER.

The result can be null; if the argument is null, the result is the null value.

ASIN

The schema is SYSFUN.

Returns the arcsine on the argument as an angle expressed in radians.

The argument can be of any built-in numeric type. It is converted to a double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

ATAN

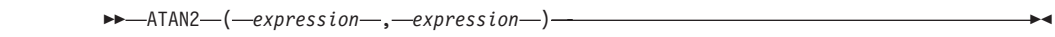
The schema is SYSFUN.

Returns the arctangent of the argument as an angle expressed in radians.

The argument can be of any built-in numeric data type. It is converted to a double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

ATAN2



The schema is SYSFUN.

Returns the arctangent of x and y coordinates as an angle expressed in radians. The x and y coordinates are specified by the first and second arguments respectively.

The first and the second arguments can be of any built-in numeric data type. Both are converted to a double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if any argument is null, the result is the null value.

BIGINT

The schema is SYSIBM.

The BIGINT function returns a 64 bit integer representation of a number or character string in the form of an integer constant.

numeric-expression
An expression that returns a value of any built-in numeric data type.

If the argument is a *numeric-expression*, the result is the same number that would occur if the argument were assigned to a big integer column or variable. If the whole part of the argument is not within the range of integers, an error occurs. The decimal part of the argument is truncated if present.

character-expression
An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading and trailing blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant (SQLSTATE 22018). The character string cannot be a long string.

If the argument is a *character-expression*, the result is the same number that would occur if the corresponding integer constant were assigned to a big integer column or variable.

The result of the function is a big integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Examples:

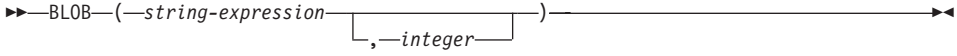
- From ORDERS_HISTORY table, count the number of orders and return the result as a big integer value.

```
SELECT BIGINT (COUNT_BIG(*) )
  FROM ORDERS_HISTORY
```

- Using the EMPLOYEE table, select the EMPNO column in big integer form for further processing in the application.

```
SELECT BIGINT(EMPNO) FROM EMPLOYEE
```

BLOB



The schema is SYSIBM.

The BLOB function returns a BLOB representation of a string of any type.

string-expression
A *string-expression* whose value can be a character string, graphic string, or a binary string.

integer
An integer value specifying the length attribute of the resulting BLOB data type. If *integer* is not specified, the length attribute of the result is the same as the length of the input, except where the input is graphic. In this case, the length attribute of the result is twice the length of the input.

The result of the function is a BLOB. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Examples

- Given a table with a BLOB column named TOPOGRAPHIC_MAP and a VARCHAR column named MAP_NAME, locate any maps that contain the string 'Pellow Island' and return a single binary string with the map name concatenated in front of the actual map.

```
SELECT BLOB(MAP_NAME  || ':  ') || TOPOGRAPHIC_MAP
  FROM ONTARIO_SERIES_4
  WHERE TOPOGRAPIC_MAP LIKE BLOB('%Pellow Island%')
```

CEILING or CEIL

The schema is SYSFUN.

Returns the smallest integer value greater than or equal to the argument.

The argument can be of any built-in numeric type. If the argument is of type DECIMAL or REAL, it is converted to a double-precision floating-point number for processing by the function. If the argument is of type SMALLINT or INTEGER, the argument value is returned.

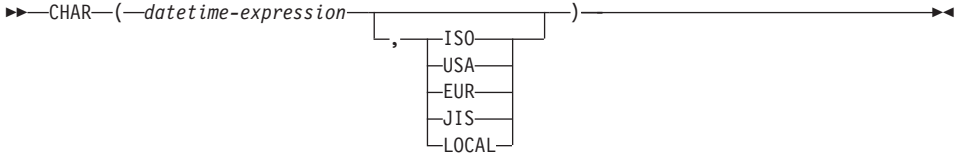
The result of the function is:

- SMALLINT if the argument is SMALLINT
- INTEGER if the argument is INTEGER
- BIGINT if the argument is BIGINT
- DOUBLE if the argument is DECIMAL, REAL or DOUBLE. Decimal values with more than 15 digits to the left of the decimal will not return the desired integer value due to loss of precision in the conversion to DOUBLE.

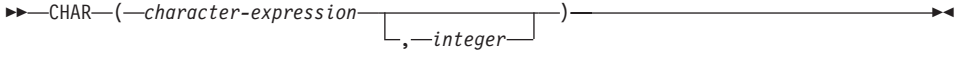
The result can be null; if the argument is null, the result is the null value.

CHAR

Datetime to Character:

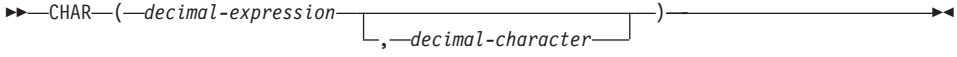


Character to Character:



Integer to Character:

Decimal to Character:



Floating-point to Character:

The schema is SYSIBM. However, the schema for CHAR(floating-point-expression) is SYSFUN.

The CHAR function returns a character-string representation of a:

- Datetime value if the first argument is a date, time or timestamp
- Character string value if the first argument is any type of character string
- Integer number if the first argument is a SMALLINT, INTEGER or BIGINT
- Decimal number if the first argument is a decimal number
- Double-precision floating-point number if the first argument is a DOUBLE or REAL.

The result of the function is a fixed-length character string. If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

Datetime to Character

datetime-expression
An expression that is one of the following three data types

date The result is the character string representation of the date in the format specified by the second argument. The length of the result is 10. An error occurs if the second argument is specified and is not a valid value (SQLSTATE 42703).

time The result is the character string representation of the time in the format specified by the second argument. The length of the result is 8. An error occurs if the second argument is specified and is not a valid value (SQLSTATE 42703).

timestamp
The second argument is not applicable and must not be specified. SQLSTATE 42815 The result is the character string representation of the timestamp. The length of the result is 26.

The code page of the string is the code page of the database at the application server.

Character to Character

character-expression
An expression that returns a value that is CHAR, VARCHAR, LONG VARCHAR, or CLOB data type.

integer
the length attribute for the resulting fixed length character string. The value must be between 0 and 254.

If the length of the character-expression is less than the length attribute of the result, the result is padded with blanks up to the length of the result. If the length of the character-expression is greater than the length attribute of the result, truncation is performed. A warning is returned (SQLSTATE 01004) unless the truncated characters were all blanks and the character-expression was not a long string (LONG VARCHAR or CLOB).

Integer to Character

integer-expression
An expression that returns a value that is an integer data type (either SMALLINT, INTEGER or BIGINT).

The result is the character string representation of the argument in the form of an SQL integer constant. The result consists of n characters that are the significant digits that represent the value of the argument with a preceding minus sign if the argument is negative. It is left justified.

- If the first argument is a small integer:

  The length of the result is 6. If the number of characters in the result is less than 6, then the result is padded on the right with blanks to length 6.
- If the first argument is a large integer:

  The length of the result is 11. If the number of characters in the result is less than 11, then the result is padded on the right with blanks to length 11.
- If the first argument is a big integer:

  The length of the result is 20. If the number of characters in the result is less than 20, then the result is padded on the right with blanks to length 20.

The code page of the string is the code page of the database at the application server.

Decimal to Character

decimal-expression
An expression that returns a value that is a decimal data type. If a different precision and scale is desired, the DECIMAL scalar function can be used first to make the change.

decimal-character
Specifies the single-byte character constant that is used to delimit the decimal digits in the result character string. The character cannot be a digit, plus ('+'), minus ('-') or blank. (SQLSTATE 42815). The default is the period ('.') character

The result is the fixed-length character-string representation of the argument. The result includes a decimal character and *p* digits, where *p* is the precision of the *decimal-expression* with a preceding minus sign if the argument is negative. The length of the result is 2+*p*, where *p* is the precision of the *decimal-expression*. This means that a positive value will always include one trailing blank.

The code page of the string is the code page of the database at the application server.

Floating-point to Character

floating-point-expression
An expression that returns a value that is a floating-point data type (DOUBLE or REAL).

The result is the fixed-length character-string representation of the argument in the form of a floating-point constant. The length of the result is 24. If the argument is negative, the first character of the result is a minus sign. Otherwise, the first character is a digit. If the argument value is zero, the result is 0E0. Otherwise, the result includes the smallest number of characters that can represent the value of the argument such that the mantissa consists of a single digit other than zero followed by a period and a sequence of digits. If the number of characters in the result is less than 24, then the result is padded on the right with blanks to length 24.

The code page of the string is the code page of the database at the application server.

Examples:

- Assume the column PRSTDATE has an internal value equivalent to 1988-12-25.

  ```
  CHAR(PRSTDATE, USA)
  ```

  Results in the value '12/25/1988'.
- Assume the column STARTING has an internal value equivalent to 17.12.30, the host variable HOUR_DUR (decimal(6,0)) is a time duration with a value of 050000. (that is, 5 hours).

  ```
  CHAR(STARTING, USA)
  ```

  Results in the value '5:12 PM'.

  ```
  CHAR(STARTING + :HOUR_DUR, USA)
  ```

  Results in the value '10:12 PM'.
- Assume the column RECEIVED (timestamp) has an internal value equivalent to the combination of the PRSTDATE and STARTING columns.

  ```
  CHAR(RECEIVED)
  ```

  Results in the value '1988-12-25-17.12.30.000000'.

- Use the CHAR function to make the type fixed length character and reduce the length of the displayed results to 10 characters for the LASTNAME column (defined as VARCHAR(15)) of the EMPLOYEE table.

  ```
  SELECT CHAR(LASTNAME,10) FROM EMPLOYEE
  ```

  For rows having a LASTNAME with a length greater than 10 characters (excluding trailing blanks), a warning that the value is truncated is returned.
- Use the CHAR function to return the values for EDLEVEL (defined as smallint) as a fixed length character string.

  ```
  SELECT CHAR(EDLEVEL) FROM EMPLOYEE
  ```

  An EDLEVEL of 18 would be returned as the CHAR(6) value '18 ' (18 followed by four blanks).
- Assume that STAFF has a SALARY column defined as decimal with precision of 9 and scale of 2. The current value is 18357.50 and it is to be displayed with a comma as the decimal character (18357,50).

  ```
  CHAR(SALARY, ',')
  ```

  returns the value '00018357,50 '.
- Assume the same SALARY column subtracted from 20000.25 is to be displayed with the default decimal character.

  ```
  CHAR(20000.25 - SALARY)
  ```

  returns the value '-0001642.75'.
- Assume a host variable, SEASONS_TICKETS, has an integer data type and a 10000 value.

  ```
  CHAR(DECIMAL(:SEASONS_TICKETS,7,2))
  ```

  Results in the character value '10000.00 '.
- Assume a host variable, DOUBLE_NUM has a double data type and a value of -987.654321E-35.

  ```
  CHAR(:DOUBLE_NUM)
  ```

  Results in the character value of '-9.87654321E-33 '. Since the result data type is CHAR(24), there are 9 trailing blanks in the result.

CHR

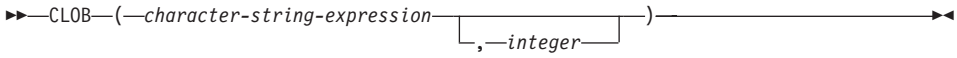
The schema is SYSFUN.

Returns the character that has the ASCII code value specified by the argument.

The argument can be either INTEGER or SMALLINT. The value of the argument should be between 0 and 255; otherwise, the return value is null.

The result of the function is CHAR(1). The result can be null; if the argument is null, the result is the null value.

CLOB



The schema is SYSIBM.

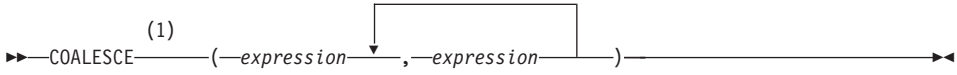
The CLOB function returns a CLOB representation of a character string type.

character-string-expression
An *expression* that returns a value that is a character string.

integer
An integer value specifying the length attribute of the resulting CLOB data type. The value must be between 0 and 2 147 483 647. If *integer* is not specified, the length of the result is the same as the length of the first argument.

The result of the function is a CLOB. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

COALESCE



Notes:

1 VALUE is a synonym for COALESCE.

The schema is SYSIBM.

COALESCE returns the first argument that is not null.

The arguments are evaluated in the order in which they are specified, and the result of the function is the first argument that is not null. The result can be null only if all the arguments can be null, and the result is null only if all the arguments are null. The selected argument is converted, if necessary, to the attributes of the result.

The arguments must be compatible. See "Rules for Result Data Types" on page 107 for what data types are compatible and the attributes of the result. They can be of either a built-in or user-defined data type. [40]

Examples:

- When selecting all the values from all the rows in the DEPARTMENT table, if the department manager (MGRNO) is missing (that is, null), then return a value of 'ABSENT'.

```
SELECT DEPTNO, DEPTNAME, COALESCE(MGRNO, 'ABSENT'), ADMRDEPT
  FROM DEPARTMENT
```

- When selecting the employee number (EMPNO) and salary (SALARY) from all the rows in the EMPLOYEE table, if the salary is missing (that is, null), then return a value of zero.

```
SELECT EMPNO, COALESCE(SALARY, 0)
  FROM EMPLOYEE
```

40. This function may not be used as a source function when creating a user-defined function. Since it accepts any compatible data types as arguments, it is not necessary to create additional signatures to support user-defined distinct types.

CONCAT

Notes:

1 || may be used as a synonym for CONCAT.

The schema is SYSIBM.

Returns the concatenation of two string arguments. The two arguments must be compatible types.

The result of the function is a string. Its length is sum of the lengths of the two arguments. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

See "With the Concatenation Operator" on page 158 for more information.

COS

The schema is SYSFUN.

Returns the cosine of the argument, where the argument is an angle expressed in radians.

The argument can be of any built-in numeric type. It is converted to a double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

COT

The schema is SYSFUN.

Returns the cotangent of the argument, where the argument is an angle expressed in radians.

The argument can be of any built-in numeric type. It is converted to a double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

DATE

The schema is SYSIBM.

The DATE function returns a date from a value.

The argument must be a date, timestamp, a positive number less than or equal to 3 652 059, a valid character string representation of a date or timestamp, or a character string of length 7 that is neither a CLOB nor a LONG VARCHAR.

If the argument is a character string of length 7, it must represent a valid date in the form *yyyynnn*, where *yyyy* are digits denoting a year, and *nnn* are digits between 001 and 366, denoting a day of that year.

The result of the function is a date. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a date, timestamp, or valid string representation of a date or timestamp:
  - The result is the date part of the value.
- If the argument is a number:
  - The result is the date that is *n*-1 days after January 1, 0001, where *n* is the integral part of the number.
- If the argument is a character string with a length of 7:
  - The result is the date represented by the character string.

Examples:

- Assume that the column RECEIVED (timestamp) has an internal value equivalent to '1988-12-25-17.12.30.000000'.

  ```
  DATE(RECEIVED)
  ```

  Results in an internal representation of '1988-12-25'.
- This example results in an internal representation of '1988-12-25'.

  ```
  DATE('1988-12-25')
  ```
- This example results in an internal representation of '1988-12-25'.

  ```
  DATE('25.12.1988')
  ```
- This example results in an internal representation of '0001-02-04'.

```
DATE(35)
```

DAY

The schema is SYSIBM.

The DAY function returns the day part of a value.

The argument must be a date, timestamp, date duration, timestamp duration, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a date, timestamp, or valid string representation of a date or timestamp:
  - The result is the day part of the value, which is an integer between 1 and 31.
- If the argument is a date duration or timestamp duration:
  - The result is the day part of the value, which is an integer between –99 and 99. A nonzero result has the same sign as the argument.

Examples:

- Using the PROJECT table, set the host variable END_DAY (smallint) to the day that the WELD LINE PLANNING project (PROJNAME) is scheduled to stop (PRENDATE).

  ```
  SELECT DAY(PRENDATE)
    INTO :END_DAY
    FROM PROJECT
    WHERE PROJNAME = 'WELD LINE PLANNING'
  ```

  Results in END_DAY being set to 15 when using the sample table.
- Assume that the column DATE1 (date) has an internal value equivalent to 2000-03-15 and the column DATE2 (date) has an internal value equivalent to 1999-12-31.

  ```
  DAY(DATE1 - DATE2)
  ```

  Results in the value 15.

DAYNAME

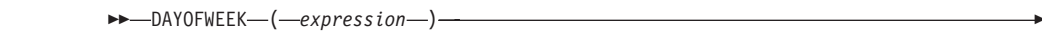
The schema is SYSFUN.

Returns a mixed case character string containing the name of the day (e.g. Friday) for the day portion of the argument based on the locale when the database was started.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is VARCHAR(100). The result can be null; if the argument is null, the result is the null value.

DAYOFWEEK



The schema is SYSFUN.

Returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Sunday.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

DAYOFWEEK_ISO

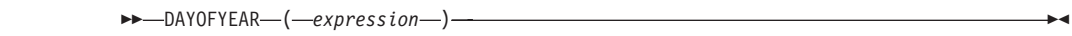
The schema is SYSFUN.

Returns the day of the week in the argument as an integer value in the range 1-7, where 1 represents Monday.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

DAYOFYEAR



The schema is SYSFUN.

Returns the day of the year in the argument as an integer value in the range 1-366.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

DAYS

The schema is SYSIBM.

The DAYS function returns an integer representation of a date.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is 1 more than the number of days from January 1, 0001 to *D*, where *D* is the date that would occur if the DATE function were applied to the argument.

Examples:

- Using the PROJECT table, set the host variable EDUCATION_DAYS (int) to the number of elapsed days (PRENDATE - PRSTDATE) estimated for the project (PROJNO) 'IF2000'.

  ```
  SELECT DAYS(PRENDATE) - DAYS(PRSTDATE)
    INTO :EDUCATION_DAYS
    FROM PROJECT
    WHERE PROJNO = 'IF2000'
  ```

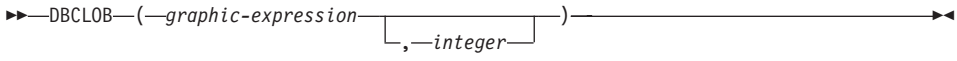
  Results in EDUCATION_DAYS being set to 396 when using the sample table.

- Using the PROJECT table, set the host variable TOTAL_DAYS (int) to the sum of elapsed days (PRENDATE - PRSTDATE) estimated for all projects in department (DEPTNO) 'E21'.

  ```
  SELECT SUM(DAYS(PRENDATE) - DAYS(PRSTDATE))
    INTO :TOTAL_DAYS
    FROM PROJECT
    WHERE DEPTNO = 'E21'
  ```

  Results in TOTAL_DAYS being set to 1584 when using the sample table.

DBCLOB



The schema is SYSIBM.

The DBCLOB function returns a DBCLOB representation of a graphic string type.

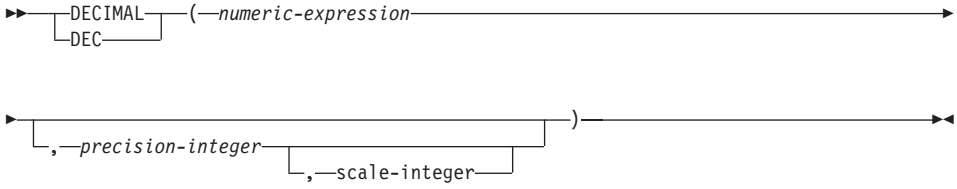
graphic-expression
An *expression* that returns a value that is a graphic string.

integer
An integer value specifying the length attribute of the resulting DBCLOB data type. The value must be between 0 and 1 073 741 823. If *integer* is not specified, the length of the result is the same as the length of the first argument.

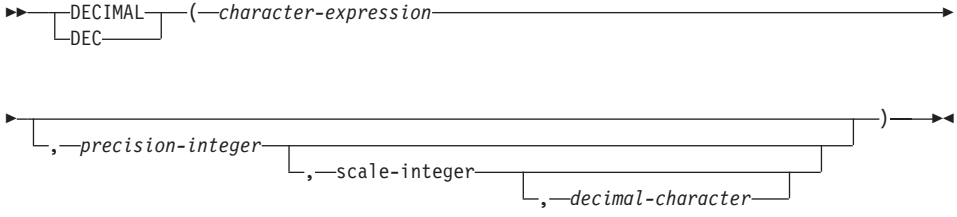
The result of the function is a DBCLOB. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

DECIMAL

Numeric to Decimal:



Character to Decimal:



The schema is SYSIBM.

The DECIMAL function returns a decimal representation of

- A number
- A character string representation of a decimal number
- A character string representation of a integer number.

The result of the function is a decimal number with precision of *p* and scale of *s*, where *p* and *s* are the second and third arguments. If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

Numeric to Decimal

numeric-expression
An expression that returns a value of any numeric data type.

precision-integer
An integer constant with a value in the range of 1 to 31.

The default for the *precision-integer* depends on the data type of the *numeric-expression*:

- 15 for floating-point and decimal
- 19 for big integer

- 11 for large integer
- 5 for small integer.

scale-integer
An integer constant in the range of 0 to the *precision-integer* value. The default is zero.

The result is the same number that would occur if the first argument were assigned to a decimal column or variable with a precision of *p* and a scale of *s*, where *p* and *s* are the second and third arguments. An error occurs if the number of significant decimal digits required to represent the whole part of the number is greater than *p-s*.

Character to Decimal

character-expression
An *expression* that returns a value that is a character string with a length not greater than the maximum length of a character constant (4 000 bytes). It cannot have a CLOB or LONG VARCHAR data type. Leading and trailing blanks are eliminated from the string. The resulting substring must conform to the rules for forming an SQL integer or decimal constant (SQLSTATE 22018).

The *character-expression* is converted to the database code page if required to match the code page of the constant *decimal-character*.

precision-integer
An integer constant with a value in the range 1 to 31 that specifies the precision of the result. If not specified, the default is 15.

scale-integer
An integer constant with a value in the range 0 to *precision-integer* that specifies the scale of the result. If not specified, the default is 0.

decimal-character
Specifies the single byte character constant that is used to delimit the decimal digits in *character-expression* from the whole part of the number. The character cannot be a digit plus ('+'), minus ('-') or blank and can appear at most once in *character-expression* (SQLSTATE 42815).

The result is a decimal number with precision *p* and scale *s* where *p* and *s* are the second and third arguments. Digits are truncated from the end if the number of digits right of the decimal character is greater than the scale *s*. An error occurs if the number of significant digits left of the decimal character (the whole part of the number) in *character-expression* is greater than *p-s* (SQLSTATE 22003). The default

decimal character is not valid in the substring if the *decimal-character* argument is specified (SQLSTATE 22018).

Examples:

- Use the DECIMAL function in order to force a DECIMAL data type (with a precision of 5 and a scale of 2) to be returned in a select-list for the EDLEVEL column (data type = SMALLINT) in the EMPLOYEE table. The EMPNO column should also appear in the select list.

```
SELECT EMPNO, DECIMAL(EDLEVEL,5,2)
  FROM EMPLOYEE
```

- Assume the host variable PERIOD is of type INTEGER. Then, in order to use its value as a date duration it must be ″cast″ as decimal(8,0).

```
SELECTPRSTDATE + DECIMAL(:PERIOD,8)
  FROM PROJECT
```

- Assume that updates to the SALARY column are input through a window as a character string using comma as a decimal character (for example, the user inputs 21400,50). Once validated by the application, it is assigned to the host variable newsalary which is defined as CHAR(10).

```
UPDATE STAFF
  SET SALARY = DECIMAL(:newsalary, 9, 2, ',')
  WHERE ID = :empid;
```

  The value of newsalary becomes 21400.50.

- Add the default decimal character (.) to a value.

```
DECIMAL('21400,50', 9, 2, '.')
```

  This fails because a period (.) is specified as the decimal character but a comma (,) appears in the first argument as a delimiter.

DEGREES

The schema is SYSFUN.

Returns the number of degrees converted from the argument expressed in radians.

The argument can be of any built-in numeric type. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

DEREF

The schema is SYSIBM.

The DEREF function returns an instance of the target type of the argument.

The argument can be any value with a reference data type that has a defined scope (SQLSTATE 428DT).

The static data type of the result is the target type of the argument. The dynamic data type of the result is a subtype of the target type of the argument. The result can be null. The result is the null value if *expression* is a null value or if *expression* is a reference that has no matching OID in the target table.

The result is an instance of the subtype of the target type of the reference. The result is determined by finding the row of the target table or target view of the reference that has an object identifier that matches the reference value. The type of this row determines the dynamic type of the result. Since the type of the result can be based on a row of a subtable or subview of the target table or target view, the authorization ID of the statement must have SELECT privilege on the target table and all of its subtables or the target view and all of its subviews (SQLSTATE 42501).

Examples:

Assume that EMPLOYEE is a table of type EMP, and that its object identifier column is named EMPID. Then the following query returns an object of type EMP (or one of its subtypes), for each row of the EMPLOYEE table (and its subtables). This query requires SELECT privilege on EMPLOYEE and all its subtables.

```
SELECT DEREF(EMPID) FROM EMPLOYEE
```

For additional examples, see "TYPE_NAME" on page 378.

DIFFERENCE

The schema is SYSFUN.

Returns a value from 0 to 4 representing the difference between the sounds of two strings based on applying the SOUNDEX function to the strings. A value of 4 is the best possible sound match.

The arguments can be character strings that are either CHAR or VARCHAR up to 4 000 bytes.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

Example:

```
VALUES (DIFFERENCE('CONSTRAINT','CONSTANT'),SOUNDEX('CONSTRAINT'),
        SOUNDEX('CONSTANT')),
        (DIFFERENCE('CONSTRAINT','CONTRITE'),SOUNDEX('CONSTRAINT'),
        SOUNDEX('CONTRITE'))
```

This example returns the following.

```
1           2    3
----------- ---- ----
          4 C523 C523
          2 C523 C536
```

In the first row, the words have the same result from SOUNDEX while in the second row the words have only some similarity.

DIGITS

The schema is SYSIBM.

The DIGITS function returns a character-string representation of a number.

The argument must be an expression that returns a value of type SMALLINT, INTEGER, BIGINT or DECIMAL.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result of the function is a fixed-length character string representing the absolute value of the argument without regard to its scale. The result does not include a sign or a decimal character. Instead, it consists exclusively of digits, including, if necessary, leading zeros to fill out the string. The length of the string is:

- 5 if the argument is a small integer
- 10 if the argument is a large integer
- 19 if the argument is a big integer
- *p* if the argument is a decimal number with a precision of *p*.

Examples:

- Assume that a table called TABLEX contains an INTEGER column called INTCOL containing 10-digit numbers. List all distinct four digit combinations of the first four digits contained in column INTCOL.

  ```
  SELECT DISTINCT SUBSTR(DIGITS(INTCOL),1,4)
    FROM TABLEX
  ```

- Assume that COLUMNX has the DECIMAL(6,2) data type, and that one of its values is -6.28. Then, for this value:

  ```
  DIGITS(COLUMNX)
  ```

  returns the value '000628'.

  The result is a string of length six (the precision of the column) with leading zeros padding the string out to this length. Neither sign nor decimal point appear in the result.

DLCOMMENT

The schema is SYSIBM.

The DLCOMMENT function returns the comment value, if it exists, from a DATALINK value.

The argument must be an expression that results in a value with data type of DATALINK.

The result of the function is VARCHAR(254). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example:

- Prepare a statement to select the date, the description and the comment from the link to the ARTICLES column from the HOCKEY_GOALS table. The rows to be selected are those for goals scored by either of the Richard brothers (Maurice or Henri).

```
stmtvar = "SELECT DATE_OF_GOAL, DESCRIPTION, DLCOMMENT(ARTICLES)
             FROM HOCKEY_GOALS
             WHERE BY_PLAYER = 'Maurice Richard'
             OR BY_PLAYER = 'Henri Richard' ";
EXEC SQL PREPARE HOCKEY_STMT FROM :stmtvar;
```

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

```
DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','A comment')
```

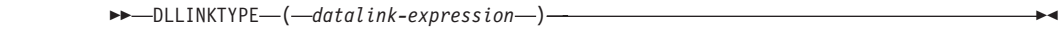
  then the following function operating on that value:

```
DLCOMMENT(COLA)
```

  will return the value:

```
A comment
```

DLLINKTYPE



The schema is SYSIBM.

The DLLINKTYPE function returns the linktype value from a DATALINK value.

The argument must be an expression that results in a value with data type DATALINK.

The result of the function is VARCHAR(4). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example:

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

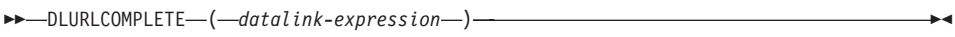
  then the following function operating on that value:

  ```
  DLLINKTYPE(COLA)
  ```

  will return the value:

  ```
  URL
  ```

DLURLCOMPLETE



The schema is SYSIBM.

The DLURLCOMPLETE function returns the data location attribute from a DATALINK value with a link type of URL. When appropriate, the value includes a file access token.

The argument must be an expression that results in a value with data type DATALINK.

The result of the function is VARCHAR(254). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

If the DATALINK value only includes the comment the result returned is a zero length string.

Example:

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

  then the following function operating on that value:

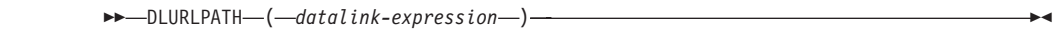
  ```
  DLURLCOMPLETE(COLA)
  ```

  will return the value:

  ```
  HTTP://DLFS.ALMADEN.IBM.COM/x/y/**********;a.b
  ```

  (where ********** represents the access token)

DLURLPATH



The schema is SYSIBM.

The DLURLPATH function returns the path and file name necessary to access a file within a given server from a DATALINK value with a linktype of URL. When appropriate, the value includes a file access token.

The argument must be an expression that results in a value with data type DATALINK.

The result of the function is VARCHAR(254). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

If the DATALINK value only includes the comment the result returned is a zero length string.

Example:

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

  then the following function operating on that value:

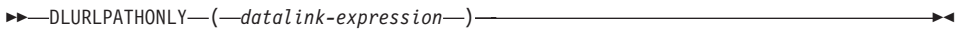
  ```
  DLURLPATH(COLA)
  ```

  will return the value:

  ```
  /x/y/**********;a.b
  ```

  (where ********** represents the access token)

DLURLPATHONLY



The schema is SYSIBM.

The DLURLPATHONLY function returns the path and file name necessary to access a file within a given server from a DATALINK value with a linktype of URL. The value returned NEVER includes a file access token.

The argument must be an expression that results in a value with data type DATALINK.

The result of the function is VARCHAR(254). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

If the DATALINK value only includes the comment the result returned is a zero length string.

Example:

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

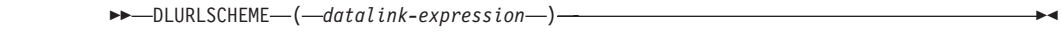
  then the following function operating on that value:

  ```
  DLURLPATHONLY(COLA)
  ```

  will return the value:

  ```
  /x/y/a.b
  ```

DLURLSCHEME



The schema is SYSIBM.

The DLURLSCHEME function returns the scheme from a DATALINK value with a linktype of URL. The value will always be in upper case.

The argument must be an expression that results in a value with data type DATALINK.

The result of the function is VARCHAR(20). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

If the DATALINK value only includes the comment the result returned is a zero length string.

Example:

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

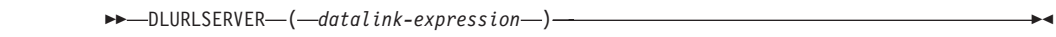
  then the following function operating on that value:

  ```
  DLURLSCHEME(COLA)
  ```

  will return the value:

  ```
  HTTP
  ```

DLURLSERVER



The schema is SYSIBM.

The DLURLSERVER function returns the file server from a DATALINK value with a linktype of URL. The value will always be in upper case.

The argument must be an expression that results in a value with data type DATALINK.

The result of the function is VARCHAR(254). If the argument can be null, the result can be null; if the argument is null, the result is the null value.

If the DATALINK value only includes the comment the result returned is a zero length string.

Example:

- Given a DATALINK value that was inserted into column COLA of a row in table TBLA using the scalar function:

  ```
  DLVALUE('http://dlfs.almaden.ibm.com/x/y/a.b','URL','a comment')
  ```

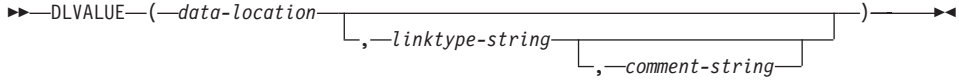
  then the following function operating on that value:

  ```
  DLURLSERVER(COLA)
  ```

  will return the value:

  ```
  DLFS.ALMADEN.IBM.COM
  ```

DLVALUE



The schema is SYSIBM.

The DLVALUE function returns a DATALINK value. When the function is on the right hand side of a SET clause in an UPDATE statement or is in a VALUES clause in an INSERT statement, it usually also creates a link to a file. However, if only a comment is specified (in which case the data-location is a zero-length string), the DATALINK value is created with empty linkage attributes so there is no file link.

data-location
If the link type is URL, then this is an expression that yields a varying length character string containing a complete URL value.

linktype-string
An optional VARCHAR expression that specifies the link type of the DATALINK value. The only valid value is 'URL' (SQLSTATE 428D1).

comment-string
An optional VARCHAR(254) value that provides a comment or additional location information.

The result of the function is a DATALINK value. If any argument of the DLVALUE function can be null, the result can be null; If the *data-location* is null, the result is the null value.

When defining a DATALINK value using this function, consider the maximum length of the target of the value. For example, if a column is defined as DATALINK(200), then the maximum length of the *data-location* plus the *comment* is 200 bytes.

Example:

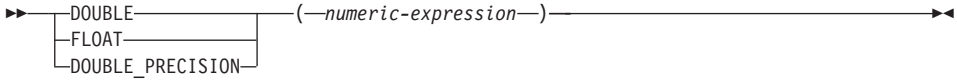
- Insert a row into the table. The URL values for the first two links are contained in the variables named url_article and url_snapshot. The variable named url_snapshot_comment contains a comment to accompany the snapshot link. There is, as yet, no link for the movie, only a comment in the variable named url_movie_comment.

```
EXEC SQL  INSERT INTO HOCKEY_GOALS
            VALUES('Maurice Richard',
                   'Montreal Canadien',
                   '?',
                   'Boston Bruins,
```

```
'1952-04-24',
'Winning goal in game 7 of Stanley Cup final',
 DLVALUE(:url_article),
 DLVALUE(:url_snapshot, 'URL', :url_snapshot_comment),
 DLVALUE('', 'URL', :url_movie_comment) );
```

DOUBLE

Numeric to Double:



Character String to Double:

The schema is SYSIBM. However, the schema for DOUBLE(*string-expression*) is SYSFUN.

The DOUBLE function returns a floating-point number corresponding to a:

- number if the argument is a numeric expression
- character string representation of a number if the argument is a string expression.

Numeric to Double

numeric-expression
The argument is an expression that returns a value of any built-in numeric data type.

The result of the function is a double-precision floating-point number. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is the same number that would occur if the argument were assigned to a double-precision floating-point column or variable.

Character String to Double

string-expression
The argument can be of type CHAR or VARCHAR in the form of a numeric constant. Leading and trailing blanks in argument are ignored.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

The result is the same number that would occur if the string was considered a constant and assigned to a double-precision floating-point column or variable.

Example:

Using the EMPLOYEE table, find the ratio of salary to commission for employees whose commission is not zero. The columns involved (SALARY and COMM) have DECIMAL data types. To eliminate the possibility of out-of-range results, DOUBLE is applied to SALARY so that the division is carried out in floating point:

```
SELECT EMPNO, DOUBLE(SALARY)/COMM
  FROM EMPLOYEE
  WHERE COMM > 0
```

EVENT_MON_STATE

The schema is SYSIBM.

The EVENT_MON_STATE function returns the current state of an event monitor.

The argument is a string expression with a resulting type of CHAR or VARCHAR and a value that is the name of an event monitor. If the named event monitor does not exist in the SYSCAT.EVENTMONITORS catalog table, SQLSTATE 42704 will be returned.

The result is an integer with one of the following values:

•

0 The event monitor is inactive.

1 The event monitor is active.

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example:

- The following example selects all of the defined event monitors, and indicates whether each is active or inactive:

```
SELECT EVMONNAME,
   CASE
      WHEN EVENT_MON_STATE(EVMONNAME) = 0 THEN 'Inactive'
      WHEN EVENT_MON_STATE(EVMONNAME) = 1 THEN 'Active'
   END
FROM SYSCAT.EVENTMONITORS
```

EXP

The schema is SYSFUN.

Returns the exponential function of the argument.

The argument can be of any built-in numeric data type. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

FLOAT

The schema is SYSIBM.

The FLOAT function returns a floating-point representation of a number.

FLOAT is a synonym for DOUBLE. See "DOUBLE" on page 296 for details.

FLOOR

The schema is SYSFUN.

Returns the largest integer value less than or equal to the argument.

The argument can be of any built-in numeric type. If the argument is of type DECIMAL or REAL, it is converted to a double-precision floating-point number for processing by the function. If the argument is of type SMALLINT, INTEGER or BIGINT the argument value is returned.

The result of the function is:

- SMALLINT if the argument is SMALLINT
- INTEGER if the argument is INTEGER
- BIGINT if the argument is BIGINT
- DOUBLE if the argument is DOUBLE, DECIMAL or REAL. Decimal values with more than 15 digits to the left of the decimal will not return the desired integer value due to loss of precision in the conversion to DOUBLE.

The result can be null; if the argument is null, the result is the null value.

GENERATE_UNIQUE

The schema is SYSIBM.

The GENERATE_UNIQUE function returns a bit data character string 13 bytes long (CHAR(13) FOR BIT DATA) that is unique compared to any other execution of the same function.[41] The function is defined as not-deterministic.

There are no arguments to this function (the empty parentheses must be specified).

The result of the function is a unique value that includes the internal form of the Universal Time, Coordinated (UTC) and the partition number where the function was processed. The result cannot be null.

The result of this function can be used to provide unique values in a table. Each successive value will be greater than the previous value, providing a sequence that can be used within a table. The value includes the partition number where the function executed so that a table partitioned across multiple partitions also has unique values in some sequence. The sequence is based on the time the function was executed.

This function differs from using the special register CURRENT TIMESTAMP in that a unique value is generated for each row of a multiple row insert statement or an insert statement with a fullselect.

The timestamp value that is part of the result of this function can be determined using the TIMESTAMP scalar function with the result of GENERATE_UNIQUE as an argument.

Examples:

- Create a table that includes a column that is unique for each row. Populate this column using the GENERATE_UNIQUE function. Notice that the UNIQUE_ID column has "FOR BIT DATA" specified to identify the column as a bit data character string.

```
CREATE TABLE EMP_UPDATE
(UNIQUE_ID CHAR(13) FOR BIT DATA,
 EMPNO CHAR(6),
```

41. The system clock is used to generate the internal Universal Time, Coordinated (UTC) timestamp along with the partition number on which the function executes. Adjustments that move the actual system clock backward could result in duplicate values.

```
  TEXT  VARCHAR(1000))

INSERT INTO EMP_UPDATE
          VALUES (GENERATE_UNIQUE(), '000020', 'Update entry...'),
                 (GENERATE_UNIQUE(), '000050', 'Update entry...')
```

This table will have a unique identifier for each row provided that the UNIQUE_ID column is always set using GENERATE_UNIQUE. This can be done by introducing a trigger on the table.

```
CREATE TRIGGER EMP_UPDATE_UNIQUE
NO CASCADE BEFORE INSERT ON EMP_UPDATE
REFERENCING NEW AS NEW_UPD
FOR EACH ROW MODE DB2SQL
SET NEW_UPD.UNIQUE_ID = GENERATE_UNIQUE()
```

With this trigger defined, the previous INSERT statement could be issued without the first column as follows.

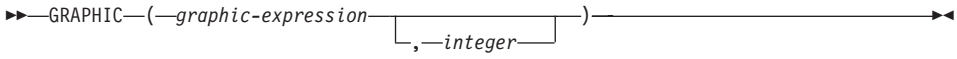
```
INSERT INTO EMP_UPDATE (EMPNO, TEXT)
            VALUES ('000020', 'Update entry 1...'),
                   ('000050', 'Update entry 2...')
```

The timestamp (in UTC) for when a row was added to EMP_UPDATE can be returned using:

```
       SELECT TIMESTAMP (UNIQUE_ID), EMPNO, TEXT  FROM EMP_UPDATE
```

Therefore, there is no need to have a timestamp column in the table to record when a row is inserted.

GRAPHIC



The schema is SYSIBM.

The GRAPHIC function returns a GRAPHIC representation of a graphic string type.

graphic-expression
An *expression* that returns a value that is a graphic string.

integer
An integer value specifying the length attribute of the resulting GRAPHIC data type. The value must be between 1 and 127. If *integer* is not specified, the length of the result is the same as the length of the first argument.

The result of the function is a GRAPHIC. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

HEX

The schema is SYSIBM.

The HEX function returns a hexadecimal representation of a value as a character string.

The argument can be an expression that is a value of any built-in data type with a maximum length of 16 336 bytes.

The result of the function is a character string. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The code page is the database code page.

The result is a string of hexadecimal digits. The first two represent the first byte of the argument, the next two represent the second byte of the argument, and so forth. If the argument is a datetime value or a numeric value the result is the hexadecimal representation of the internal form of the argument. The hexadecimal representation that is returned may be different depending on the application server where the function is executed. Cases where differences would be evident include:

- Character string arguments when the HEX function is performed on an ASCII client with an EBCDIC server or on an EBCDIC client with an ASCII server.
- Numeric arguments (in some cases) when the HEX function is performed where client and server systems have different byte orderings for numeric values.

The type and length of the result vary based on the type and length of character string arguments.

- Character string
  - Fixed length not greater than 127
    - Result is a character string of fixed length twice the defined length of the argument.
  - Fixed length greater than 127
    - Result is a character string of varying length twice the defined length of the argument.
  - Varying length

    - Result is a character string of varying length with maximum length twice the defined maximum length of the argument.
- Graphic string
  - Fixed length not greater than 63
    - Result is a character string of fixed length four times the defined length of the argument.
- Fixed length greater than 63
  - Result is a character string of varying length four times the defined length of the argument.
- Varying length
  - Result is a character string of varying length with maximum length four times the defined maximum length of the argument.

Examples:

Assume the use of a DB2 for AIX application server for the following examples.

- Using the DEPARTMENT table set the host variable HEX_MGRNO (char(12)) to the hexadecimal representation of the manager number (MGRNO) for the 'PLANNING' department (DEPTNAME).

```
SELECT HEX(MGRNO)
  INTO :HEX_MGRNO
  FROM DEPARTMENT
  WHERE DEPTNAME = 'PLANNING'
```

  HEX_MGRNO will be set to '303030303230' when using the sample table (character value is '000020').
- Suppose COL_1 is a column with a data type of char(1) and a value of 'B'. The hexadecimal representation of the letter 'B' is X'42'. HEX(COL_1) returns a two-character string '42'.
- Suppose COL_3 is a column with a data type of decimal(6,2) and a value of 40.1. An eight-character string '0004010C' is the result of applying the HEX function to the internal representation of the decimal value, 40.1.

HOUR

The schema is SYSIBM.

The HOUR function returns the hour part of a value.

The argument must be a time, timestamp, time duration, timestamp duration or a valid character string representation of a time or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

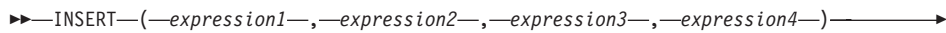
- If the argument is a time, timestamp or valid string representation of a time or timestamp:
  - The result is the hour part of the value, which is an integer between 0 and 24.
- If the argument is a time duration or timestamp duration:
  - The result is the hour part of the value, which is an integer between –99 and 99. A nonzero result has the same sign as the argument.

Example:

Using the CL_SCHED sample table, select all the classes that start in the afternoon.

```
SELECT * FROM CL_SCHED
  WHERE HOUR(STARTING) BETWEEN 12 AND 17
```

INSERT



The schema is SYSFUN.

Returns a string where *expression3* bytes have been deleted from *expression1* beginning at *expression2* and where *expression4* has been inserted into *expression1* beginning at *expression2*. If the length of the result string exceeds the maximum for the return type, an error occurs (SQLSTATE 38552).

The first argument is a character string or a binary string type. The second and third arguments must be a numeric value with a data type of SMALLINT or INTEGER. If the first argument is a character string, then the fourth argument must also be a character string. If the first argument is a binary string, then the fourth argument must be a binary string. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB or a binary string the maximum length is 1 048 576 bytes. For the first and fourth arguments, CHAR is converted to VARCHAR and LONG VARCHAR to CLOB(1M), for second and third arguments SMALLINT is converted to INTEGER for processing by the function.

The result is based on the argument types as follows:

- VARCHAR(4000) if both the first and fourth arguments are VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if either the first or fourth argument is CLOB or LONG VARCHAR
- BLOB(1M) if both first and fourth arguments are BLOB.

The result can be null; if any argument is null, the result is the null value.

Example:

- Delete one character from the word 'DINING' and insert 'VID', both beginning at the third character.

  ```
  VALUES CHAR(INSERT('DINING', 3, 1, 'VID'), 10)
  ```

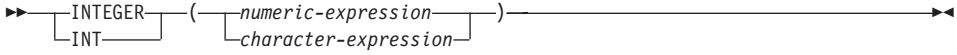
  This example returns the following:

  ```
  1
  ----------
  DIVIDING
  ```

  As mentioned, the output of the INSERT function is VARCHAR(4000). For the above example the function CHAR has been used to limit the output of

INSERT to 10 bytes. The starting location of a particular string can be found using LOCATE. Refer to "LOCATE" on page 318 for more information.

INTEGER



The schema is SYSIBM.

The INTEGER function returns an integer representation of a number or character string in the form of an integer constant.

numeric-expression
An expression that returns a value of any built-in numeric data type.

If the argument is a *numeric-expression*, the result is the same number that would occur if the argument were assigned to a large integer column or variable. If the whole part of the argument is not within the range of integers, an error occurs. The decimal part of the argument is truncated if present.

character-expression
An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading and trailing blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant (SQLSTATE 22018). The character string cannot be a long string.

If the argument is a *character-expression*, the result is the same number that would occur if the corresponding integer constant were assigned to a large integer column or variable.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Examples:

- Using the EMPLOYEE table, select a list containing salary (SALARY) divided by education level (EDLEVEL). Truncate any decimal in the calculation. The list should also contain the values used in the calculation and employee number (EMPNO). The list should be in descending order of the calculated value.

```
SELECT INTEGER (SALARY / EDLEVEL), SALARY, EDLEVEL, EMPNO
  FROM EMPLOYEE
  ORDER BY 1 DESC
```

- Using the EMPLOYEE table, select the EMPNO column in integer form for further processing in the application.

```
SELECT INTEGER(EMPNO) FROM EMPLOYEE
```

JULIAN_DAY

The schema is SYSFUN.

Returns an integer value representing the number of days from January 1,4712 B.C. (the start of Julian date calendar) to the date value specified in the argument.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

LCASE or LOWER

The schema is SYSIBM. [42]

The LCASE or LOWER function returns a string in which all the SBCS characters have been converted to lowercase characters (that is, the characters A-Z will be translated to the characters a-z, and characters with diacritical marks will be translated to their lower case equivalents if they exist. For example, in code page 850, É maps to é). Since not all characters are translated, LCASE(UCASE(*string-expression*)) does not necessarily return the same result as LCASE(*string-expression*).

The argument must be an expression whose value is a CHAR or VARCHAR data type.

The result of the function has the same data type and length attribute of the argument. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

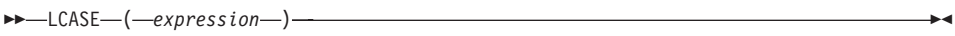
Example: Ensure that the characters in the value of column JOB in the EMPLOYEE table are returned in lowercase characters.

```
SELECT LCASE(JOB)
  FROM EMPLOYEE WHERE EMPNO = '000020';
```

The result is the value 'manager'.

42. The SYSFUN version of this function continues to be available with support for LONG VARCHAR and CLOB arguments. See "LCASE (SYSFUN schema)" on page 313 for a description.

LCASE (SYSFUN schema)



The schema is SYSFUN.

Returns a string in which all the characters A-Z have been converted to the characters a-z (characters with diacritical marks are not converted). Note that LCASE(UCASE(string)) will therefore not necessarily return the same result as LCASE(string).

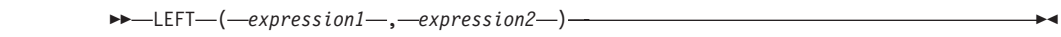
The argument can be of any built-in character string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB the maximum length is 1 048 576 bytes.

The result of the function is:

- VARCHAR(4000) if the argument is VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if the argument is CLOB or LONG VARCHAR

The result can be null; if the argument is null, the result is the null value.

LEFT



The schema is SYSFUN.

Returns a string consisting of the leftmost *expression2* bytes in *expression1*. The *expression1* value is effectively padded on the right with the necessary number of blank characters so that the specified substring of *expression1* always exists.

The first argument is a character string or binary string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB or a binary string the maximum length is 1 048 576 bytes. The second argument must be of INTEGER or SMALLINT datatype.

The result of the function is:

- VARCHAR(4000) if the argument is VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if the argument is CLOB or LONG VARCHAR
- BLOB(1M) if the argument is BLOB.

The result can be null; if any argument is null, the result is the null value.

LENGTH

The schema is SYSIBM.

The LENGTH function returns the length of a value.

The argument can be an expression that returns a value of any built-in data type.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The result is the length of the argument. The length does not include the null indicator byte of column arguments that allow null values. The length of strings includes blanks but does not include the length control field of varying-length strings. The length of a varying-length string is the actual length, not the maximum length.

The length of a graphic string is the number of DBCS characters. The length of all other values is the number of bytes used to represent the value:

- 2 for small integer
- 4 for large integer
- (p/2)+1 for decimal numbers with precision p
- The length of the string for binary strings
- The length of the string for character strings
- 4 for single-precision floating-point
- 8 for double-precision floating-point
- 4 for date
- 3 for time
- 10 for timestamp

Examples:

- Assume the host variable ADDRESS is a varying length character string with a value of '895 Don Mills Road'.

  ```
  LENGTH(:ADDRESS)
  ```

  Returns the value 18.
- Assume that START_DATE is a column of type DATE.

  ```
  LENGTH(START_DATE)
  ```

Returns the value 4.

- This example returns the value 10.

```
LENGTH(CHAR(START_DATE, EUR))
```

LN

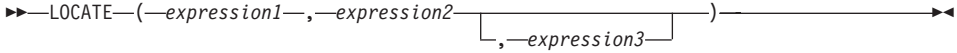
The schema is SYSFUN.

Returns the natural logarithm of the argument (same as LOG).

The argument can be of any built-in numeric data type. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

LOCATE



The schema is SYSFUN.

Returns the starting position of the first occurrence of *expression1* within *expression2*. If the optional *expression3* is specified, it indicates the character position in *expression2* at which the search is to begin. If *expression1* is not found within *expression2*, the value 0 is returned.

If the first argument is a character string, then the second argument must be a character string. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB the maximum length is 1 048 576 bytes. If the first argument is a binary string, then the second argument must be a binary string with a maximum length of 1 048 576 bytes. The third argument must be is INTEGER or SMALLINT.

The result of the function is INTEGER. The result can be null; if any argument is null, the result is the null value.

Example:

- Find the location of the letter 'N' (first occurrence) in the word 'DINING'.

  ```
  VALUES LOCATE ('N', 'DINING')
  ```

  This example returns the following:

  ```
  1
  -----------
            3
  ```

LOG

The schema is SYSFUN.

Returns the natural logarithm of the argument (same as LN).

The argument can be of any built-in numeric data type. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

LOG10

The schema is SYSFUN.

Returns the base 10 logarithm of the argument.

The argument can be of any built-in numeric type. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

LONG_VARCHAR

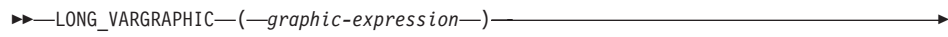
The schema is SYSIBM.

The LONG_VARCHAR function returns a LONG VARCHAR representation of a character string data type.

character-string-expression
An *expression* that returns a value that is a character string with a maximum length of 32 700 bytes.

The result of the function is a LONG VARCHAR. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

LONG_VARGRAPHIC



The schema is SYSIBM.

The LONG_VARGRAPHIC function returns a LONG VARGRAPHIC representation of a double-byte character string.

graphic-expression
An *expression* that returns a value that is a graphic string with a maximum length of 16 350 double byte characters.

The result of the function is a LONG VARGRAPHIC. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

LTRIM

The schema is SYSIBM. [43]

The LTRIM function removes blanks from the beginning of *string-expression*.

The argument can be a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type.

- If the argument is a graphic string in a DBCS or EUC database, then the leading double byte blanks are removed.
- If the argument is a graphic string in a Unicode database, then the leading UCS-2 blanks are removed.
- Otherwise, the leading single byte blanks are removed.

The result data type of the function is:

- VARCHAR if the data type of *string-expression* is VARCHAR or CHAR
- VARGRAPHIC if the data type of *string-expression* is VARGRAPHIC or GRAPHIC

The length parameter of the returned type is the same as the length parameter of the argument data type.

The actual length of the result for character strings is the length of *string-expression* minus the number of bytes removed for blank characters. The actual length of the result for graphic strings is the length (in number of double byte characters) of *string-expression* minus the number of double byte blank characters removed. If all of the characters are removed, the result is an empty, varying-length string (length is zero).

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HELLO is defined as CHAR(9) and has a value of ' Hello'.

```
VALUES LTRIM(:HELLO)
```

The result is 'Hello'.

43. The SYSFUN version of this function continues to be available with support for LONG VARCHAR and CLOB arguments. See "LTRIM (SYSFUN schema)" on page 324 for a description.

LTRIM (SYSFUN schema)

The schema is SYSFUN.

Returns the characters of the argument with leading blanks removed.

The argument can be of any built-in character string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB the maximum length is 1 048 576 bytes.

The result of the function is:

- VARCHAR(4000) if the argument is VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if the argument is CLOB or LONG VARCHAR.

The result can be null; if the argument is null, the result is the null value.

MICROSECOND

The schema is SYSIBM.

The MICROSECOND function returns the microsecond part of a value.

The argument must be a timestamp, timestamp duration or a valid character string representation of a timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

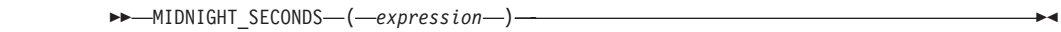
- If the argument is a timestamp or a valid string representation of a timestamp:
  - The integer ranges from 0 through 999 999.
- If the argument is a duration:
  - The result reflects the microsecond part of the value which is an integer between –999 999 through 999 999. A nonzero result has the same sign as the argument.

Example:

- Assume a table TABLEA contains two columns, TS1 and TS2, of type TIMESTAMP. Select all rows in which the microseconds portion of TS1 is not zero and the seconds portion of TS1 and TS2 are identical.

```
SELECT * FROM TABLEA
  WHERE MICROSECOND(TS1) <> 0 AND
  SECOND(TS1) = SECOND(TS2)
```

MIDNIGHT_SECONDS



The schema is SYSFUN.

Returns an integer value in the range 0 to 86 400 representing the number of seconds between midnight and the time value specified in the argument.

The argument must be a time, timestamp, or a valid character string representation of a time or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

Example:

- Find the number of seconds between midnight and 00:10:10, and midnight and 13:10:10.

  ```
  VALUES (MIDNIGHT_SECONDS('00:10:10'), MIDNIGHT_SECONDS('13:10:10'))
  ```

  This example returns the following:

  ```
  1           2
  ----------- -----------
          610       47410
  ```

  Since a minute is 60 seconds, there are 610 seconds between midnight and the specified time. The same follows for the second example. There are 3600 seconds in an hour, and 60 seconds in a minute, resulting in 47410 seconds between the specified time and midnight.

- Find the number of seconds between midnight and 24:00:00, and midnight and 00:00:00.

  ```
  VALUES (MIDNIGHT_SECONDS('24:00:00'), MIDNIGHT_SECONDS('00:00:00'))
  ```

  This example returns the following:

  ```
  1           2
  ----------- -----------
        86400           0
  ```

  Note that these two values represent the same point in time, but return different MIDNIGHT_SECONDS values.

MINUTE

The schema is SYSIBM.

The MINUTE function returns the minute part of a value.

The argument must be a time, timestamp, time duration, timestamp duration or a valid character string representation of a time or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a time, timestamp or valid string representation of a time or timestamp:
  - The result is the minute part of the value, which is an integer between 0 and 59.
- If the argument is a time duration or timestamp duration:
  - The result is the minute part of the value, which is an integer between –99 and 99. A nonzero result has the same sign as the argument.

Example:

- Using the CL_SCHED sample table, select all classes with a duration less than 50 minutes.

```
SELECT * FROM CL_SCHED
  WHERE HOUR(ENDING - STARTING) = 0 AND
  MINUTE(ENDING - STARTING) < 50
```

MOD

The schema is SYSFUN.

Returns the remainder of the first argument divided by the second argument. The result is negative only if first argument is negative.

The result of the function is:

- SMALLINT if both arguments are SMALLINT
- INTEGER if one argument is INTEGER and the other is INTEGER or SMALLINT
- BIGINT if one argument is BIGINT and the other argument is BIGINT, INTEGER or SMALLINT.

The result can be null; if any argument is null, the result is the null value.

MONTH

The schema is SYSIBM.

The MONTH function returns the month part of a value.

The argument must be a date, timestamp, date duration, timestamp duration or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a date, timestamp, or a valid string representation of a date or timestamp:
  - The result is the month part of the value, which is an integer between 1 and 12.
- If the argument is a date duration or timestamp duration:
  - The result is the month part of the value, which is an integer between –99 and 99. A nonzero result has the same sign as the argument.

Example:

- Select all rows from the EMPLOYEE table for people who were born (BIRTHDATE) in DECEMBER.

```
SELECT * FROM EMPLOYEE
  WHERE MONTH(BIRTHDATE) = 12
```

MONTHNAME

The schema is SYSFUN.

Returns a mixed case character string containing the name of month (e.g. January) for the month portion of the argument, based on the locale when the database was started.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is VARCHAR(100). The result can be null; if the argument is null, the result is the null value.

NODENUMBER

The schema is SYSIBM.

The NODENUMBER function returns the partition number of the row. For example, if used in a SELECT clause, it returns the partition number for each row of the table that was used to form the result of the SELECT statement.

The partition number returned on transition variables and tables is derived from the current transition values of the partitioning key columns. For example, in a before insert trigger, the function will return the projected partition number given the current values of the new transition variables. However, the values of the partitioning key columns may be modified by a subsequent before insert trigger. Thus, the final partition number of the row when it is inserted into the database may differ from the projected value.

The argument must be the qualified or unqualified name of a column of a table. The column can have any data type. [44] If *column-name* references a column of a view the expression in the view for the column must reference a column of the underlying base table and the view must be deletable. A nested or common table expression follows the same rules as a view. See "Notes" on page 832 for the definition of a deletable view.

The specific row (and table) for which the partition number is returned by the NODENUMBER function is determined from the context of the SQL statement that uses the function.

The data type of the result is INTEGER and is never null. Since row-level information is returned, the results are the same, regardless of which column is specified for the table. If there is no `db2nodes.cfg` file, the result is 0.

The NODENUMBER function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

Examples:

44. This function may not be used as a source function when creating a user-defined function. Since it accepts any data types as an argument, it is not necessary to create additional signatures to support user-defined distinct types.

- Count the number of rows where the row for an EMPLOYEE is on a different partition from the employee's department description in DEPARTMENT.

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E
        WHERE D.DEPTNO=E.WORKDEPT
        AND NODENUMBER(E.LASTNAME) <> NODENUMBER(D.DEPTNO)
```

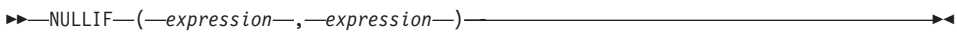
- Join the EMPLOYEE and DEPARTMENT tables where the rows of the two tables are on the same partition.

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E
        WHERE NODENUMBER(E.LASTNAME) = NODENUMBER(D.DEPTNO)
```

- Log the employee number and the projected partition number of the new row into a table called EMPINSERTLOG1 for any insertion of employees by creating a before trigger on the table EMPLOYEE.

```
CREATE TRIGGER EMPINSLOGTRIG1
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH MODE ROW MODE DB2SQL
INSERT INTO EMPINSERTLOG1
VALUES(NEWTABLE.EMPNO, NODENUMBER(NEWTABLE.EMPNO))
```

NULLIF



The schema is SYSIBM.

The NULLIF function returns a null value if the arguments are equal, otherwise it returns the value of the first argument.

The arguments must be comparable (see "Assignments and Comparisons" on page 94). They can be of either a built-in (other than a long string or DATALINK) or distinct data type (other than based on a long string or DATALINK). [45] The attributes of the result are the attributes of the first argument.

The result of using NULLIF(e1,e2) is the same as using the expression

```
CASE WHEN e1=e2 THEN NULL ELSE e1 END
```

Note that when e1=e2 evaluates to unknown (because one or both arguments is NULL), CASE expressions consider this not true. Therefore, in this situation, NULLIF returns the value of the first argument.

Example:

- Assume host variables PROFIT, CASH, and LOSSES have DECIMAL data types with the values 4500.00, 500.00, and 5000.00 respectively:

  ```
  NULLIF (:PROFIT + :CASH , :LOSSES )
  ```

  Returns a null value.

45. This function may not be used as a source function when creating a user-defined function. Since it accepts any compatible data types as arguments, it is not necessary to create additional signatures to support user-defined distinct types.

PARTITION

The schema is SYSIBM.

The PARTITION function returns the partitioning map index of the row obtained by applying the partitioning function on the partitioning key value of the row. For example, if used in a SELECT clause, it returns the partitioning map index for each row of the table that was used to form the result of the SELECT statement.

The partitioning map index returned on transition variables and tables is derived from the current transition values of the partitioning key columns. For example, in a before insert trigger, the function will return the projected partitioning map index given the current values of the new transition variables. However, the values of the partitioning key columns may be modified by a subsequent before insert trigger. Thus, the final partitioning map index of the row when it is inserted into the database may differ from the projected value.

The argument must be the qualified or unqualified name of a column of a table. The column can have any data type. [46] If *column-name* references a column of a view the expression in the view for the column must reference a column of the underlying base table and the view must be deletable. A nested or common table expression follows the same rules as a view. See "Notes" on page 832 for the definition of a deletable view.

The specific row (and table) for which the partitioning map index is returned by the PARTITION function is determined from the context of the SQL statement that uses the function.

The data type of the result is INTEGER in the range 0 to 4095. For a table with no partitioning key, the result is always 0. A null value is never returned. Since row-level information is returned, the results are the same, regardless of which column is specified for the table.

The PARTITION function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

46. This function may not be used as a source function when creating a user-defined function. Since it accepts any data type as an arguments, it is not necessary to create additional signatures to support user-defined distinct types.

Example:

- List the employee numbers (EMPNO) from the EMPLOYEE table for all rows with a partitioning map index of 100.

```
SELECT EMPNO FROM EMPLOYEE
  WHERE PARTITION(PHONENO) = 100
```

- Log the employee number and the projected partitioning map index of the new row into a table called EMPINSERTLOG2 for any insertion of employees by creating a before trigger on the table EMPLOYEE.

```
CREATE TRIGGER EMPINSLOGTRIG2
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH MODE ROW MODE DB2SQL
INSERT INTO EMPINSERTLOG2
  VALUES(NEWTABLE.EMPNO, PARTITION(NEWTABLE.EMPNO))
```

POSSTR

The schema is SYSIBM.

The POSSTR function returns the starting position of the first occurrence of one string (called the *search-string*) within another string (called the *source-string*). Numbers for the *search-string* position start at 1 (not 0).

The result of the function is a large integer. If either of the arguments can be null, the result can be null; if either of the arguments is null, the result is the null value.

source-string
An expression that specifies the source string in which the search is to take place.

The expression can be specified by any one of:

- a constant
- a special register
- a host variable (including a locator variable or a file reference variable)
- a scalar function
- a large object locator
- a column name
- an expression concatenating any of the above

search-string
An expression that specifies the string that is to be searched for.

The expression can be specified by any one of:

- a constant
- a special register
- a host variable
- a scalar function whose operands are any of the above
- an expression concatenating any of the above

with the restrictions that:

- No element in the expression can be of type LONG VARCHAR, CLOB, LONG VARGRAPHIC or DBCLOB. In addition, it cannot be a BLOB file reference variable.
- The actual length of *search-string* cannot be more than 4 000 bytes.

Note that these rules are the same as those for the *pattern-expression* described in "LIKE Predicate" on page 197.

Both *search-string* and *source-string* have zero or more contiguous positions. If the strings are character or binary strings, a position is a byte. If the strings are graphic strings, a position is a graphic (DBCS) character.

The POSSTR function accepts mixed data strings. However, POSSTR operates on a strict byte-count basis, oblivious to changes between single and multi-byte characters.

The following rules apply:

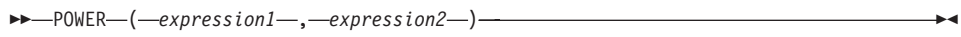
- The data types of *source-string* and *search-string* must be compatible, otherwise an error is raised (SQLSTATE 42884).
  - If *source-string* is a character string, then *search-string* must be a character string, but not a CLOB or LONG VARCHAR, with an actual length of 32 672 bytes or less.
  - If *source-string* is a graphic string, then *search-string* must be a graphic string, but not a DBCLOB or LONG VARGRAPHIC, with an actual length of 16 336 double-byte characters or less.
  - If *source-string* is a binary string, then *search-string* must be a binary string with an actual length of 32 672 bytes or less.
- If *search-string* has a length of zero, the result returned by the function is 1.
- Otherwise:
  - If *source-string* has a length of zero, the result returned by the function is zero.
  - Otherwise:
    - If the value of *search-string* is equal to an identical length substring of contiguous positions from the value of *source-string*, then the result returned by the function is the starting position of the first such substring within the *source-string* value.
    - Otherwise, the result returned by the function is 0.

Example

- Select RECEIVED and SUBJECT columns as well as the starting position of the words 'GOOD BEER' within the NOTE_TEXT column for all entries in the IN_TRAY table that contain these words.

```
SELECT RECEIVED, SUBJECT, POSSTR(NOTE_TEXT, 'GOOD BEER')
  FROM IN_TRAY
  WHERE POSSTR(NOTE_TEXT, 'GOOD BEER') <> 0
```

POWER



The schema is SYSFUN.

Returns the value of *expression1* to the power of *expression2*.

The arguments can be of any built-in numeric data type. DECIMAL and REAL arguments are converted to double-precision floating-point number.

The result of the function is:

- INTEGER if both arguments are INTEGER or SMALLINT
- BIGINT if one argument is BIGINT and the other argument is BIGINT, INTEGER or SMALLINT
- DOUBLE otherwise.

The result can be null; if any argument is null, the result is the null value.

QUARTER

The schema is SYSFUN.

Returns an integer value in the range 1 to 4 representing the quarter of the year for the date specified in the argument.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

RADIANS

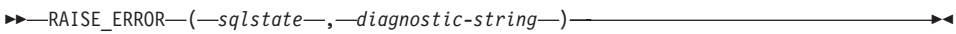
The schema is SYSFUN.

Returns the number of radians converted from argument which is expressed in degrees.

The argument can be of any built-in numeric data types. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

RAISE_ERROR



The schema is SYSIBM.

The RAISE_ERROR function causes the statement that includes the function to return an error with the specified SQLSTATE, SQLCODE -438 and *diagnostic-string*. The RAISE_ERROR function always returns NULL with an undefined data type.

sqlstate
A character string containing exactly 5 characters. It must be of type CHAR defined with a length of 5 or type VARCHAR defined with a length of 5 or greater. The *sqlstate* value must follow the rules for application-defined SQLSTATEs as follows:

- Each character must be from the set of digits ('0' through '9') or non-accented upper case letters ('A' through 'Z')
- The SQLSTATE class (first two characters) cannot be '00', '01' or '02' since these are not error classes.
- If the SQLSTATE class (first two characters) starts with the character '0' through '6' or 'A' through 'H', then the subclass (last three characters) must start with a letter in the range 'I' through 'Z'
- If the SQLSTATE class (first two characters) starts with the character '7', '8', '9' or 'I' though 'Z', then the subclass (last three characters) can be any of '0' through '9' or 'A' through 'Z'.

If the SQLSTATE does not conform to these rules an error occurs (SQLSTATE 428B3).

diagnostic-string
An expression of type CHAR or VARCHAR that returns a character string of up to 70 bytes that describes the error condition. If the string is longer than 70 bytes, it will be truncated.

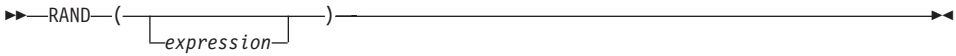
In order to use this function in a context where Rules for Result Data Types do not apply (such as alone in a select list), a cast specification must be used to give the null returned value a data type. A CASE expression is where the RAISE_ERROR function will be most useful.

Example:

List employee numbers and education levels as Post Graduate, Graduate and Diploma. If an education level is greater than 20, raise an error.

```
SELECT EMPNO,
       CASE WHEN EDUCLVL < 16 THEN 'Diploma'
            WHEN EDUCLVL < 18 THEN 'Graduate'
            WHEN EDUCLVL < 21 THEN 'Post Graduate'
            ELSE RAISE_ERROR('70001',
                             'EDUCLVL has a value greater than 20')
       END
     FROM EMPLOYEE
```

RAND



The schema is SYSFUN.

Returns a random floating point value between 0 and 1 using the argument as the optional seed value. The function is defined as not-deterministic.

An argument is not required, but if it is specified it can be either INTEGER or SMALLINT.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

REAL

The schema is SYSIBM.

The REAL function returns a single-precision floating-point representation of a number.

The argument is an expression that returns a value of any built-in numeric data type.

The result of the function is a single-precision floating-point number. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

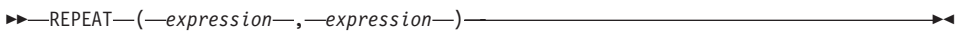
The result is the same number that would occur if the argument were assigned to a single-precision floating-point column or variable.

Example:

Using the EMPLOYEE table, find the ratio of salary to commission for employees whose commission is not zero. The columns involved (SALARY and COMM) have DECIMAL data types. The result is desired in single-precision floating point. Therefore, REAL is applied to SALARY so that the division is carried out in floating point (actually double precision) and then REAL is applied to the complete expression to return the result in single-precision floating point.

```
SELECT EMPNO, REAL(REAL(SALARY)/COMM)
  FROM EMPLOYEE
  WHERE COMM > 0
```

REPEAT



The schema is SYSFUN.

Returns a character string composed of the first argument repeated the number of times specified by the second argument.

The first argument is a character string or binary string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB or a binary string the maximum length is 1 048 576 bytes. The second argument can be SMALLINT or INTEGER.

The result of the function is:

- VARCHAR(4000) if the first argument is VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if the first argument is CLOB or LONG VARCHAR
- BLOB(1M) if the first argument is BLOB.

The result can be null; if any argument is null, the result is the null value.

Example:

- List the phrase 'REPEAT THIS' five times.

  ```
  VALUES CHAR(REPEAT('REPEAT THIS', 5), 60)
  ```

  This example return the following:

  ```
  1
  ------------------------------------------------------------
  REPEAT THISREPEAT THISREPEAT THISREPEAT THISREPEAT THIS
  ```

  As mentioned, the output of the REPEAT function is VARCHAR(4000). For the above example the function CHAR has been used to limit the output of REPEAT to 60 bytes.

REPLACE

The schema is SYSFUN.

Replaces all occurrences of *expression2* in *expression1* with *expression3*.

The first argument can be of any built-in character string or binary string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB or a binary string the maximum length is 1 048 576 bytes. CHAR is converted to VARCHAR and LONG VARCHAR is converted to CLOB(1M). The second and third arguments are identical to the first argument.

The result of the function is:

- VARCHAR(4000) if the first, second and third arguments are VARCHAR or CHAR
- CLOB(1M) if the first, second and third arguments are CLOB or LONG VARCHAR
- BLOB(1M) if the first, second and third arguments are BLOB.

The result can be null; if any argument is null, the result is the null value.

Example:

- Replace all occurrence of the letter 'N' in the word 'DINING' with 'VID'.

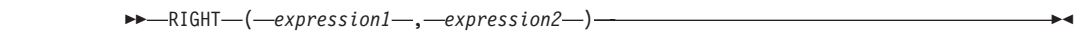
  ```
  VALUES CHAR (REPLACE ('DINING', 'N', 'VID'), 10)
  ```

  This example returns the following:

  ```
  1
  ----------
  DIVIDIVIDG
  ```

  As mentioned, the output of the REPLACE function is VARCHAR(4000). For the above example the function CHAR has been used to limit the output of REPLACE to 10 bytes.

RIGHT



The schema is SYSFUN.

Returns a string consisting of the rightmost *expression2* bytes in *expression1*. The *expression1* value is effectively padded on the right with the necessary number of blank characters so that the specified substring of *expression1* always exists.

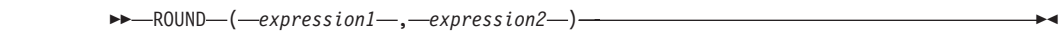
The first argument is a character string or binary string type. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB or a binary string the maximum length is 1 048 576 bytes. The second argument can be INTEGER or SMALLINT.

The result of the function is:

- VARCHAR(4000) if the first argument is VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if the first argument is CLOB or LONG VARCHAR
- BLOB(1M) if the first argument is BLOB.

The result can be null; if any argument is null, the result is the null value.

ROUND



The schema is SYSFUN.

Returns the *expression1* rounded to *expression2* places right of the decimal point. If *expression2* is negative, *expression1* is rounded to the absolute value of *expression2* places to the left of the decimal point.

The first argument can be of any built-in numeric data type. The second argument can be INTEGER or SMALLINT. DECIMAL and REAL are converted to double-precision floating-point number for processing by the function.

The result of the function is:

- INTEGER if the first argument is INTEGER or SMALLINT
- BIGINT if the first argument is BIGINT
- DOUBLE if the first argument is DOUBLE, DECIMAL or REAL.

The result can be null; if any argument is null, the result is the null value.

Example:

- Display the number 873.726 rounded to 2, 1, 0, -1 and -2 decimal places respectively.

```
VALUES (DECIMAL(ROUND(873.726,2),6,3), DECIMAL(ROUND(873.726,1),6,3),
        DECIMAL(ROUND(873.726,0),6,3), DECIMAL(ROUND(873.726,-1),6,3),
        DECIMAL(ROUND(873.726,-2),6,3))
```

  The above example returns:

```
1        2        3        4        5
-------- -------- -------- -------- ---------
 873.730  873.700  874.000  870.000   900.000
```

  As mentioned, the output of the ROUND function is DOUBLE. For the above example the function DECIMAL has been used to limit the output of ROUND.

RTRIM

The schema is SYSIBM. [47]

The RTRIM function removes blanks from the end of *string-expression*.

The argument can be a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC data type.

- If the argument is a graphic string in a DBCS or EUC database, then the trailing double byte blanks are removed.
- If the argument is a graphic string in a Unicode database, then the trailing UCS-2 blanks are removed.
- Otherwise, the trailing single byte blanks are removed.

The result data type of the function is:

- VARCHAR if the data type of *string-expression* is VARCHAR or CHAR
- VARGRAPHIC if the data type of *string-expression* is VARGRAPHIC or GRAPHIC

The length parameter of the returned type is the same as the length parameter of the argument data type.

The actual length of the result for character strings is the length of *string-expression* minus the number of bytes removed for blank characters. The actual length of the result for graphic strings is the length (in number of double byte characters) of *string-expression* minus the number of double byte blank characters removed. If all of the characters are removed, the result is an empty, varying-length string (length is zero).

If the argument can be null, the result can be null; if the argument is null, the result is the null value.

Example: Assume that host variable HELLO is defined as CHAR(9) and has a value of 'Hello'.

```
VALUES RTRIM(:HELLO)
```

The result is 'Hello'.

47. The SYSFUN version of this function continues to be available with support for LONG VARCHAR and CLOB arguments. See "RTRIM (SYSFUN schema)" on page 350 for a description.

RTRIM (SYSFUN schema)

The schema is SYSFUN.

Returns the characters of the argument with trailing blanks removed.

The argument can be of any built-in character string data types. For a VARCHAR the maximum length is 4 000 bytes and for a CLOB the maximum length is 1 048 576 bytes.

The result of the function is:

- VARCHAR(4000) if the argument is VARCHAR (not exceeding 4 000 bytes) or CHAR
- CLOB(1M) if the argument is CLOB or LONG VARCHAR.

The result can be null; if the argument is null, the result is the null value.

SECOND

The schema is SYSIBM.

The SECOND function returns the seconds part of a value.

The argument must be a time, timestamp, time duration, timestamp duration or a valid character string representation of a time or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

- If the argument is a time, timestamp or valid string representation of a time or timestamp:
  - The result is the seconds part of the value, which is an integer between 0 and 59.
- If the argument is a time duration or timestamp duration:
  - The result is the seconds part of the value, which is an integer between –99 and 99. A nonzero result has the same sign as the argument.

Examples:

- Assume that the host variable TIME_DUR (decimal(6,0)) has the value 153045.

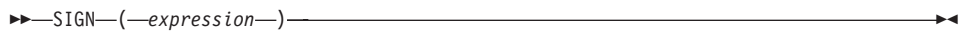
  ```
  SECOND(:TIME_DUR)
  ```

  Returns the value 45.
- Assume that the column RECEIVED (timestamp) has an internal value equivalent to 1988-12-25-17.12.30.000000.

  ```
  SECOND(RECEIVED)
  ```

  Returns the value 30.

SIGN



The schema is SYSFUN.

Returns an indicator of the sign of the argument. If the argument is less than zero, –1 is returned. If argument equals zero, 0 is returned. If argument is greater than zero, 1 is returned.

The argument can be of any built-in numeric data types. DECIMAL and REAL are converted to double-precision floating-point number for processing by the function.

The result of the function is:

- SMALLINT if the argument is SMALLINT
- INTEGER if the argument is INTEGER
- BIGINT if the argument is BIGINT
- DOUBLE otherwise.

The result can be null; if the argument is null, the result is the null value.

SIN

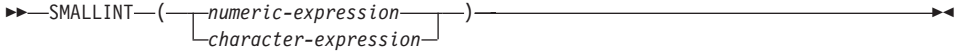
The schema is SYSFUN.

Returns the sine of the argument, where the argument is an angle expressed in radians.

The argument can be of any built-in numeric data types. It is converted to double-precision floating-point number for processing by the function.

The result of the function is a double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

SMALLINT



The schema is SYSIBM.

The SMALLINT function returns a small integer representation of a number or character string in the form of a small integer constant.

numeric-expression
An expression that returns a value of any built-in numeric data type.

If the argument is a *numeric-expression*, the result is the same number that would occur if the argument were assigned to a small integer column or variable. If the whole part of the argument is not within the range of small integers, an error occurs. The decimal part of the argument is truncated if present.

character-expression
An expression that returns a character string value of length not greater than the maximum length of a character constant. Leading and trailing blanks are eliminated and the resulting string must conform to the rules for forming an SQL integer constant (SQLSTATE 22018). However, the value of the constant must be in the range of small integers (SQLSTATE 22003). The character string cannot be a long string.

If the argument is a *character-expression*, the result is the same number that would occur if the corresponding integer constant were assigned to a small integer column or variable.

The result of the function is a small integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

SOUNDEX

The schema is SYSFUN.

Returns a 4 character code representing the sound of the words in the argument. The result can be used to compare with the sound of other strings.

The argument can be a character string that is either a CHAR or VARCHAR not exceeding 4 000 bytes.

The result of the function is CHAR(4). The result can be null; if the argument is null, the result is the null value.

The SOUNDEX function is useful for finding strings for which the sound is known but the precise spelling is not. It makes assumptions about the way that letters and combinations of letters sound that can help to search out words with similar sounds. The comparison can be done directly or by passing the strings as arguments to the DIFFERENCE function (see "DIFFERENCE" on page 285).

Example:

Using the EMPLOYEE table, find the EMPNO and LASTNAME of the employee with a surname that sounds like 'Loucesy'.

```
SELECT EMPNO, LASTNAME FROM EMPLOYEE
  WHERE SOUNDEX(LASTNAME) = SOUNDEX('Loucesy')
```

This example returns the following:

```
EMPNO  LASTNAME
------ ---------------
000110 LUCCHESSI
```

SPACE

The schema is SYSFUN.

Returns a character string consisting of blanks with length specified by the second argument.

The argument can be SMALLINT or INTEGER.

The result of the function is VARCHAR(4000). The result can be null; if the argument is null, the result is the null value.

SQRT

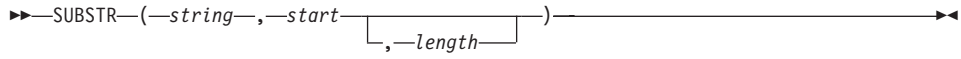
The schema is SYSFUN.

Returns the square root of the argument.

The argument can be any built-in numeric data type. It has to be converted to double-precision floating-point number for processing by the function.

The result of the function is double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

SUBSTR



The schema is SYSIBM.

The SUBSTR function returns a substring of a string.

If *string* is a character string, the result of the function is a character string represented in the code page of its first argument. If it is a binary string, the result of the function is a binary string. If it is a graphic string, the result of the function is a graphic string represented in the code page of its first argument. If any argument of the SUBSTR function can be null, the result can be null; if any argument is null, the result is the null value.

string
An expression that specifies the string from which the result is derived.

If *string* is either a character string or a binary string, a substring of *string* is zero or more contiguous bytes of *string*. If *string* is a graphic string, a substring of *string* is zero or more contiguous double-byte characters of *string*.

start
An expression that specifies the position of the first byte of the result for a character string or a binary string or the position of the first character of the result for a graphic string. *start* must be an integer between 1 and the length or maximum length of *string*, depending on whether *string* is fixed-length or varying-length (SQLSTATE 22011, if out of range). It must be specified as number of bytes in the context of the database code page and not the application code page.

length
An expression that specifies the length of the result. If specified, *length* must be a binary integer in the range 0 to *n*, where *n* equals (the length attribute of *string*) − *start* + 1 (SQLSTATE 22011, if out of range).

If *length* is explicitly specified, *string* is effectively padded on the right with the necessary number of blank characters (single-byte for character strings; double-byte for graphic strings) so that the specified substring of *string* always exists. The default for *length* is the number of bytes from the byte specified by the *start* to the last byte of *string* in the case of character string or binary string or the number of double-byte characters from the character specified by the *start* to the last character of *string* in the case of a graphic string. However, if *string* is a varying-length string with a length less than *start*, the default is zero and the result is the empty string. It

must be specified as number of bytes in the context of the database code page and not the application code page. (For example, the column NAME with a data type of VARCHAR(18) and a value of 'MCKNIGHT' will yield an empty string with `SUBSTR(NAME,10)`).

Table 16 shows that the result type and length of the SUBSTR function depend on the type and attributes of its inputs.

Table 16. Data Type and Length of SUBSTR Result

String Argument Data Type	Length Argument	Result Data Type
CHAR(A)	constant (*l*<255)	CHAR(*l*)
CHAR(A)	not specified but *start* argument is a constant	CHAR(A-*start*+1)
CHAR(A)	not a constant	VARCHAR(A)
VARCHAR(A)	constant (*l*<255)	CHAR(*l*)
VARCHAR(A)	constant (254<*l*<32673)	VARCHAR(*l*)
VARCHAR(A)	not a constant or not specified	VARCHAR(A)
LONG VARCHAR	constant (*l*<255)	CHAR(*l*)
LONG VARCHAR	constant (254<*l*<4001)	VARCHAR(*l*)
LONG VARCHAR	constant (*l*>4000)	LONG VARCHAR
LONG VARCHAR	not a constant or not specified	LONG VARCHAR
CLOB(A)	constant (*l*)	CLOB(*l*)
CLOB(A)	not a constant or not specified	CLOB(A)
GRAPHIC(A)	constant (*l*<128)	GRAPHIC(*l*)
GRAPHIC(A)	not specified but *start* argument is a constant	GRAPHIC(A-*start*+1)
GRAPHIC(A)	not a constant	VARGRAPHIC(A)
VARGRAPHIC(A)	constant (*l*<128)	GRAPHIC(*l*)
VARGRAPHIC(A)	constant (127<*l*<16337)	VARGRAPHIC(*l*)
VARGRAPHIC(A)	not a constant	VARGRAPHIC(A)

Table 16. Data Type and Length of SUBSTR Result (continued)

String Argument Data Type	Length Argument	Result Data Type
LONG VARGRAPHIC	constant (*l*<128)	GRAPHIC(*l*)
LONG VARGRAPHIC	constant (127<*l*<2001)	VARGRAPHIC(*l*)
LONG VARGRAPHIC	constant (*l*>2000)	LONG VARGRAPHIC
LONG VARGRAPHIC	not a constant or not specified	LONG VARGRAPHIC
DBCLOB(A)	constant (*l*)	DBCLOB(*l*)
DBCLOB(A)	not a constant or not specified	DBCLOB(A)
BLOB(A)	constant (*l*)	BLOB(*l*)
BLOB(A)	not a constant or not specified	BLOB(A)

If *string* is a fixed-length string, omission of *length* is an implicit specification of LENGTH(*string*) - *start* + 1. If *string* is a varying-length string, omission of *length* is an implicit specification of zero or LENGTH(*string*) - *start* + 1, whichever is greater.

Examples:

- Assume the host variable NAME (VARCHAR(50)) has a value of 'BLUE JAY' and the host variable SURNAME_POS (int) has a value of 6.

  ```
  SUBSTR(:NAME, :SURNAME_POS):ehp2s
  ```

  Returns the value 'JAY'

  ```
  SUBSTR(:NAME, :SURNAME_POS,1)
  ```

  Returns the value 'J'.

- Select all rows from the PROJECT table for which the project name (PROJNAME) starts with the word 'OPERATION'.

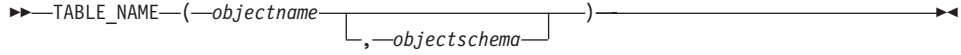
  ```
  SELECT * FROM PROJECT
    WHERE SUBSTR(PROJNAME,1,10) = 'OPERATION '
  ```

  The space at the end of the constant is necessary to preclude initial words such as 'OPERATIONS'.

Notes:

1. In dynamic SQL, *string*, *start*, and *length* may be represented by a parameter marker (?). If a parameter marker is used for *string*, the data type of the operand will be VARCHAR, and the operand will be nullable.
2. Though not explicitly stated in the result definitions above, it follows from these semantics that if *string* is a mixed single- and multi-byte character string, the result may contain fragments of multi-byte characters, depending upon the values of *start* and *length*. That is, the result could possibly begin with the second byte of a double-byte character, and/or end with the first byte of a double-byte character. The SUBSTR function does not detect such fragments, nor provides any special processing should they occur.

TABLE_NAME



The schema is SYSIBM.

The TABLE_NAME function returns an unqualified name of the object found after any alias chains have been resolved. The specified *objectname* (and *objectschema*) are used as the starting point of the resolution. If the starting point does not refer to an alias, the unqualified name of the starting point is returned. The resulting name may be of a table, view, or undefined object.

objectname
A character expression representing the unqualified name (usually of an existing alias) to be resolved. *objectname* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 characters.

objectschema
A character expression representing the schema used to qualify the supplied *objectname* value before resolution. *objectschema* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 characters.

If *objectschema* is not supplied, the default schema is used for the qualifier.

The data type of the result of the function is VARCHAR(128). If *objectname* can be null, the result can be null; if *objectname* is null, the result is the null value. If *objectschema* is the null value, the default schema name is used. The result is the character string representing an unqualified name. The result name could represent one of the following:

table The value for *objectname* was either a table name (the input value is returned) or an alias name that resolved to the table whose name is returned.

view The value for *objectname* was either a view name (the input value is returned) or an alias name that resolved to the view whose name is returned.

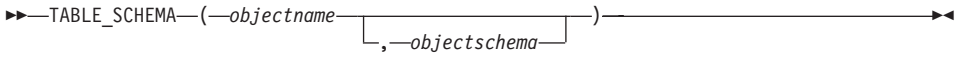
undefined object
The value for *objectname* was either an undefined object (the input value is returned) or an alias name that resolved to the undefined object whose name is returned.

Therefore, if a non-null value is given to this function, a value is always returned, even if no object with the result name exists.

Examples:

See the Examples section in “TABLE_SCHEMA” on page 364.

TABLE_SCHEMA



The schema is SYSIBM.

The TABLE_SCHEMA function returns the schema name of the object found after any alias chains have been resolved. The specified *objectname* (and *objectschema*) are used as the starting point of the resolution. If the starting point does not refer to an alias, the schema name of the starting point is returned. The resulting schema name may be of a table, view, or undefined object.

objectname
A character expression representing the unqualified name (usually of an existing alias) to be resolved. *objectname* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 characters.

objectschema
A character expression representing the schema used to qualify the supplied *objectname* value before resolution. *objectschema* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 characters.

If *objectschema* is not supplied, the default schema is used for the qualifier.

The data type of the result of the function is VARCHAR(128). If *objectname* can be null, the result can be null; if *objectname* is null, the result is the null value. If *objectschema* is the null value, the default schema name is used. The result is the character string representing a schema name. The result schema could represent the schema name for one of the following:

table The value for *objectname* was either a table name (the input or default value of *objectschema* is returned) or an alias name that resolved to a table for which the schema name is returned.

view The value for *objectname* was either a view name (the input or default value of *objectschema* is returned) or an alias name that resolved to a view for which the schema name is returned.

undefined object

The value for *objectname* was either an undefined object (the input or default value of *objectschema* is returned) or an alias name that resolved to an undefined object for which the schema name is returned.

Therefore, if a non-null *objectname* value is given to this function, a value is always returned, even if the object name with the result schema name does not exist. For example, TABLE_SCHEMA('DEPT', 'PEOPLE') returns 'PEOPLE ' if the catalog entry is not found.

Examples:

- PBIRD tries to select the statistics for a given table from SYSCAT.TABLES using an alias PBIRD.A1 defined on the table HEDGES.T1.

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
  WHERE TABNAME = TABLE_NAME ('A1')
  AND TABSCHEMA = TABLE_SCHEMA ('A1')
```

  The requested statistics for HEDGES.T1 are retrieved from the catalog.

- Select the statistics for an object called HEDGES.X1 from SYSCAT.TABLES using HEDGES.X1. Use TABLE_NAME and TABLE_SCHEMA since it is not known whether HEDGES.X1 is an alias or a table.

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
  WHERE TABNAME = TABLE_NAME ('X1','HEDGES')
  AND TABSCHEMA = TABLE_SCHEMA ('X1','HEDGES')
```

  Assuming that HEDGES.X1 is a table, the requested statistics for HEDGES.X1 are retrieved from the catalog.

- Select the statistics for a given table from SYSCAT.TABLES using an alias PBIRD.A2 defined on HEDGES.T2 where HEDGES.T2 does not exist.

```
SELECT NPAGES, CARD FROM SYSCAT.TABLES
  WHERE TABNAME = TABLE_NAME ('A2','PBIRD')
  AND TABSCHEMA = TABLE_SCHEMA ('A2',PBIRD')
```

  The statement returns 0 records as no matching entry is found in SYSCAT.TABLES where TABNAME = 'T2' and TABSCHEMA = 'HEDGES'.

- Select the qualified name of each entry in SYSCAT.TABLES along with the final referenced name for any alias entry.

```
SELECT TABSCHEMA AS SCHEMA,  TABNAME AS NAME,
  TABLE_SCHEMA (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_SCHEMA,
  TABLE_NAME (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_NAME
  FROM SYSCAT.TABLES
```

  The statement returns the qualified name for each object in the catalog and the final referenced name (after alias has been resolved) for any alias entries. For all non-alias entries, BASE_TABNAME and BASE_TABSCHEMA are null so the REAL_SCHEMA and REAL_NAME columns will contain nulls.

TAN

The schema is SYSFUN.

Returns the tangent of the argument, where the argument is an angle expressed in radians.

The argument can be any built-in numeric data type. It has to be converted to double-precision floating-point number for processing by the function.

The result of the function is double-precision floating-point number. The result can be null; if the argument is null, the result is the null value.

TIME

The schema is SYSIBM.

The TIME function returns a time from a value.

The argument must be a time, timestamp, or a valid character string representation of a time or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a time. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument:

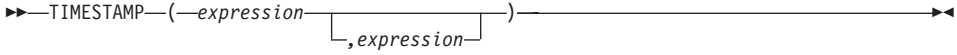
- If the argument is a time:
  - The result is that time.
- If the argument is a timestamp:
  - The result is the time part of the timestamp.
- If the argument is a character string:
  - The result is the time represented by the character string.

Example:

- Select all notes from the IN_TRAY sample table that were received at least one hour later in the day (any day) than the current time.

```
SELECT * FROM IN_TRAY
  WHERE TIME(RECEIVED) >= CURRENT TIME + 1 HOUR
```

TIMESTAMP



The schema is SYSIBM.

The TIMESTAMP function returns a timestamp from a value or a pair of values.

The rules for the arguments depend on whether the second argument is specified.

- If only one argument is specified:
  - It must be a timestamp, a valid character string representation of a timestamp, or a character string of length 14 that is neither a CLOB nor a LONG VARCHAR.

    A character string of length 14 must be a string of digits that represents a valid date and time in the form *yyyyxxddhhmmss*, where *yyyy* is the year, *xx* is the month, *dd* is the day, *hh* is the hour, *mm* is the minute, and *ss* is the seconds.
- If both arguments are specified:
  - The first argument must be a date or a valid character string representation of a date and the second argument must be a time or a valid string representation of a time.

The result of the function is a timestamp. If either argument can be null, the result can be null; if either argument is null, the result is the null value.

The other rules depend on whether the second argument is specified:

- If both arguments are specified:
  - The result is a timestamp with the date specified by the first argument and the time specified by the second argument. The microsecond part of the timestamp is zero.
- If only one argument is specified and it is a timestamp:
  - The result is that timestamp.
- If only one argument is specified and it is a character string:
  - The result is the timestamp represented by that character string. If the argument is a character string of length 14, the timestamp has a microsecond part of zero.

Example:

- Assume the column START_DATE (date) has a value equivalent to 1988-12-25, and the column START_TIME (time) has a value equivalent to 17.12.30.

  ```
  TIMESTAMP(START_DATE, START_TIME)
  ```

  Returns the value '1988-12-25-17.12.30.000000'.

TIMESTAMP_ISO

The schema is SYSFUN.

Returns a timestamp value based on date, time or timestamp argument. If the argument is a date, it inserts zero for all the time elements. If the argument is a time, it inserts the value of CURRENT DATE for the date elements and zero for the fractional time element.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is TIMESTAMP. The result can be null; if the argument is null, the result is the null value.

TIMESTAMPDIFF

The schema is SYSFUN.

Returns an estimated number of intervals of the type defined by the first argument, based on the difference between two timestamps.

The first argument can be either INTEGER or SMALLINT. Valid values of interval (the first argument) are:

1 Fractions of a second

2 Seconds

4 Minutes

8 Hours

16 Days

32 Weeks

64 Months

128 Quarters

256 Years

The second argument is the result of subtracting two timestamps types and converting the result to CHAR(22).

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

The following assumptions may be used in estimating the difference:

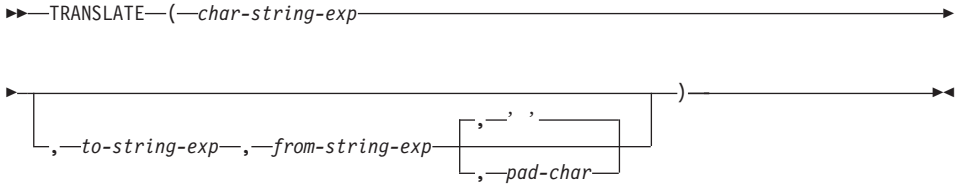
- there are 365 days in a year
- there are 30 days in a month
- there are 24 hours in a day
- there are 60 minutes in an hour
- there are 60 seconds in a minute

These assumptions are used when converting the information in the second argument, which is a timestamp duration, to the interval type specified in the first argument. The returned estimate may vary by a number of days. For example, if the number of days (interval 16) is requested for a difference in timestamps for '1997-03-01-00.00.00' and '1997-02-01-00.00.00', the result is 30.

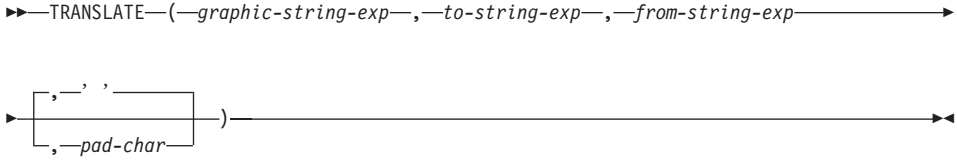
This is because the difference between the timestamps is 1 month so the assumption of 30 days in a month applies.

TRANSLATE

character string expression:



graphic string expression:



The schema is SYSIBM.

The TRANSLATE function returns a value in which one or more characters in a string expression may have been translated into other characters.

The result of the function has the same data type and code page as the first argument. The length attribute of the result is the same as that of the first argument. If any specified expression can be NULL, the result can be NULL. If any specified expression is NULL, the result will be NULL.

char-string-exp or *graphic-string-exp*
A string to be translated.

to-string-exp
Is a string of characters to which certain characters in the *char-string-exp* will be translated.

If the *to-string-exp* is not present and the data type is not graphic, all characters in the *char-string-exp* will be in monocase (that is, the characters a-z will be translated to the characters A-Z, and characters with diacritical marks will be translated to their upper case equivalents if they exist. For example, in code page 850, é maps to É, but ÿ is not mapped since code page 850 does not include Ÿ).

from-string-exp
Is a string of characters which, if found in the *char-string-exp*, will be translated to the corresponding character in the *to-string-exp*. If the

from-string-exp contains duplicate characters, the first one found will be used, and the duplicates will be ignored. If the *to-string-exp* is longer than the *from-string-exp*, the surplus characters will be ignored. If the *to-string-exp* is present, the *from-string-exp* must also be present.

pad-char-exp
Is a single character that will be used to pad the *to-string-exp* if the *to-string-exp* is shorter than the *from-string-exp*. The *pad-char-exp* must have a length attribute of one, or an error is returned. If not present, it will be taken to be a single-byte blank.

The arguments may be either strings of data type CHAR or VARCHAR, or graphic strings of data type GRAPHIC or VARGRAPHIC. They may not have data type LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB.

With *graphic-string-exp*, only the *pad-char-exp* is optional (if not provided, it will be taken to be the double-byte blank), and each argument, including the pad character, must be of graphic data type.

The result is the string that occurs after translating all the characters in the *char-string-exp* or *graphic-string-exp* that occur in the *from-string-exp* to the corresponding character in the *to-string-exp* or, if no corresponding character exists, to the pad character specified by the *pad-char-exp*.

The code page of the result of TRANSLATE is always the same as the code page of the first operand, which is never converted. Each of the other operands is converted to the code page of the first operand unless it or the first operand is defined as FOR BIT DATA (in which case there is no conversion).

If the arguments are of data type CHAR or VARCHAR, the corresponding characters of the *to-string-exp* and the *from-string-exp* must have the same number of bytes. For example, it is not valid to translate a single-byte character to a multi-byte character or vice versa. An error will result if an attempt is made to do this. The *pad-char-exp* must not be the first byte of a valid multi-byte character, or SQLSTATE 42815 is returned. If the *pad-char-exp* is not present, it will be taken to be a single-byte blank.

If only the *char-string-exp* is specified, single-byte characters will be monocased and multi-byte characters will remain unchanged.

Examples:

- Assume the host variable SITE (VARCHAR(30)) has a value of 'Hanauma Bay'.

```
TRANSLATE(:SITE)
```

Returns the value 'HANAUMA BAY'.

```
TRANSLATE(:SITE 'j','B')
```

Returns the value 'Hanauma jay'.

```
TRANSLATE(:SITE,'ei','aa')
```

Returns the value 'Heneume Bey'.

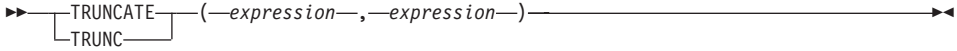
```
TRANSLATE(:SITE,'bA','Bay','%')
```

Returns the value 'HAnAumA bA%'.

```
TRANSLATE(:SITE,'r','Bu')
```

Returns the value 'Hana ma ray'.

TRUNCATE or TRUNC



The schema is SYSFUN.

Returns argument1 truncated to argument2 places right of decimal point. If argument2 is negative, argument1 is truncated to the absolute value of argument2 places to the left of the decimal point.

The first argument can be any built-in numeric data type. The second argument has to be an INTEGER or SMALLINT. DECIMAL and REAL are converted to double-precision floating-point number for processing by the function.

The result of the function is:

- INTEGER if the first argument is INTEGER or SMALLINT
- BIGINT if the first argument is BIGINT
- DOUBLE if the first argument is DOUBLE, DECIMAL or DOUBLE.

The result can be null; if any argument is null, the result is the null value.

TYPE_ID

The schema is SYSIBM.

The TYPE_ID function returns the internal type identifier of the dynamic data type of the *expression*.

The argument must be a user-defined structured type. [48]

The data type of the result of the function is INTEGER. If *expression* can be null, the result can be null; if *expression* is null, the result is the null value.

The value returned by the TYPE_ID function is not portable across databases. The value may be different, even though the type schema and type name of the dynamic data type are the same. When coding for portability, use the TYPE_SCHEMA and TYPE_NAME functions to determine the type schema and type name.

Examples:

- A table hierarchy exists having root table EMPLOYEE of type EMP and subtable MANAGER of type MGR. Another table ACTIVITIES includes a column called WHO_RESPONSIBLE that is defined as REF(EMP) SCOPE EMPLOYEE. For each reference in ACTIVITIES, display the internal type identifier of the row that corresponds to the reference.

  ```
  SELECT TASK, WHO_RESPONSIBLE->NAME,
         TYPE_ID(DEREF(WHO_RESPONSIBLE))
    FROM ACTIVITIES
  ```

  The DEREF function is used to return the object corresponding to the row.

48. This function may not be used as a source function when creating a user-defined function. Since it accepts any structured data type as an argument, it is not necessary to create additional signatures to support different user-defined types.

TYPE_NAME

The schema is SYSIBM.

The TYPE_NAME function returns the unqualified name of the dynamic data type of the *expression*.

The argument must be a user-defined structured type. [49]

The data type of the result of the function is VARCHAR(18). If *expression* can be null, the result can be null; if *expression* is null, the result is the null value. Use the TYPE_SCHEMA function to determine the schema name of the type name returned by TYPE_NAME.

Examples:

- A table hierarchy exists having root table EMPLOYEE of type EMP and subtable MANAGER of type MGR. Another table ACTIVITIES includes a column called WHO_RESPONSIBLE that is defined as REF(EMP) SCOPE EMPLOYEE. For each reference in ACTIVITIES, display the type of the row that corresponds to the reference.

  ```
  SELECT TASK, WHO_RESPONSIBLE->NAME,
         TYPE_NAME(DEREF(WHO_RESPONSIBLE)),
         TYPE_SCHEMA(DEREF(WHO_RESPONSIBLE))
    FROM ACTIVITIES
  ```

  The DEREF function is used to return the object corresponding to the row.

49. This function may not be used as a source function when creating a user-defined function. Since it accepts any structured data type as an argument, it is not necessary to create additional signatures to support different user-defined types.

TYPE_SCHEMA

The schema is SYSIBM.

The TYPE_SCHEMA function returns the schema name of the dynamic data type of the *expression*.

The argument must be a user-defined structured type. [50]

The data type of the result of the function is VARCHAR(128). If *expression* can be null, the result can be null; if *expression* is null, the result is the null value. Use the TYPE_NAME function to determine the type name associated with the schema name returned by TYPE_SCHEMA.

Examples:

See Examples section in "TYPE_NAME" on page 378.

50. This function may not be used as a source function when creating a user-defined function. Since it accepts any structured data type as an argument, it is not necessary to create additional signatures to support different user-defined types.

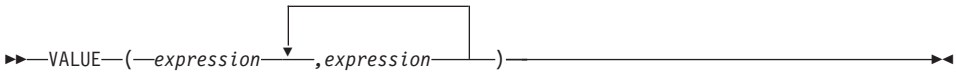
UCASE or UPPER

The schema is SYSIBM.[51]

The UCASE or UPPER function is identical to the TRANSLATE function except that only the first argument (*char-string-exp*) is specified. For more information, see "TRANSLATE" on page 373.

51. The SYSFUN version of this function continues to be available for upward compatibility. See Version 5 documentation for a description.

VALUE



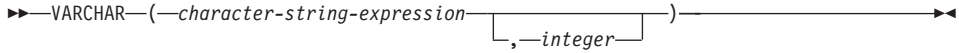
The schema is SYSIBM.

The VALUE function returns the first argument that is not null.

VALUE is a synonym for COALESCE. See "COALESCE" on page 267 for details.

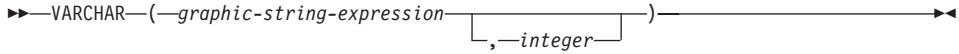
VARCHAR

Character to Varchar:



Datetime to Varchar:

Graphic to Varchar:



The schema is SYSIBM.

The VARCHAR function returns a varying-length character string representation of a character string, datetime value or graphic string (UCS-2 only).

The result of the function is a varying-length string (VARCHAR data type). If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

Graphic to Varchar is valid for a UCS-2 database only. For non-Unicode databases, this is not allowed.

Character to Varchar

character-string-expression
An expression whose value must be of a character-string data type other than LONG VARGRAPHIC and DBCLOB, with a maximum length of 32 672 bytes.

integer
The length attribute for the resulting varying-length character string. The value must be between 0 and 32 672. If this argument is not specified, the length of the result is the same as the length of the argument.

Datetime to Varchar

datetime-expression
An expression whose value must be of a date, time, or timestamp data type.

Graphic to Varchar

graphic-string-expression
An expression whose value must be of a graphic-string data type other than LONG VARGRAPHIC and DBCLOB, with a maximum length of 16 336 bytes.

integer
The length attribute for the resulting varying-length character string. The value must be between 0 and 32 672. If this argument is not specified, the length of the result is the same as the length of the argument.

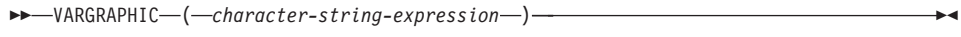
Example:

- Using the EMPLOYEE table, set the host variable JOB_DESC (VARCHAR(8)) to the VARCHAR equivalent of the job description (JOB defined as CHAR(8)) for employee Delores Quintana.

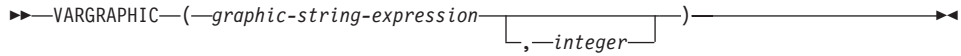
```
SELECT VARCHAR(JOB)
  INTO :JOB_DESC
  FROM EMPLOYEE
  WHERE LASTNAME = 'QUINTANA'
```

VARGRAPHIC

Character to Vargraphic:



Graphic to Vargraphic:



The schema is SYSIBM.

The VARGRAPHIC function returns a graphic string representation of a:

- character string value, converting single byte characters to double byte characters
- graphic string value, if the first argument is any type of graphic string.

The result of the function is a varying length graphic string (VARGRAPHIC data type). If the first argument can be null, the result can be null; if the first argument is null, the result is the null value.

Character to Vargraphic

character-string-expression
An expression whose value must be of a character string data type other than LONG VARCHAR or CLOB, and whose maximum length must not be greater than 16 336 bytes.

The length attribute of the result is equal to the length attribute of the argument.

Let S denote the value of the *character-string-expression*. Each single-byte character in S is converted to its equivalent double-byte representation or to the double-byte substitution character in the result; each double-byte character in S is mapped 'as-is'. If the first byte of a double-byte character appears as the last byte of S, it is converted into the double-byte substitution character. The sequential order of the characters in S is preserved.

The following are additional considerations for the conversion.

- For a Unicode database, this function converts the character-string from the code page of the operand into UCS-2. Every character of the operand,

including DBCS characters, is converted. If the second argument is given, it specifies the desired length (number of UCS-2 characters) of the resulting UCS-2 string.

- The conversion to double-byte code points by the VARGRAPHIC function is based on the code page of the operand.
- Double-byte characters of the operand are not converted (see "Appendix O. Japanese and Traditional-Chinese EUC Considerations" on page 1341 for exception). All other characters are converted to their corresponding double-byte depiction. If there is no corresponding double-byte depiction, the double-byte substitution character for the code page is used.
- No warning or error code is generated if one or more double-byte substitution characters are returned in the result.

Graphic to Vargraphic

graphic-string-expression
An expression that returns a value that is a graphic string.

integer
The length attribute for the resulting varying length graphic string. The value must be between 0 and 16 336. If this argument is not specified, the length of the result is the same as the length of the argument.

If the length of the *graphic-string-expression* is greater than the length attribute of the result, truncation is performed and a warning is returned (SQLSTATE 01004) unless the truncated characters were all blanks and the *graphic-string-expression* was not a long string (LONG VARGRAPHIC or DBCLOB).

WEEK

The schema is SYSFUN.

Returns the week of the year of the argument as an integer value in range 1-54. The week starts with Sunday.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

WEEK_ISO

The schema is SYSFUN.

Returns the week of the year of the argument as an integer value in range 1-53. The week starts with Monday. Week 1 is the first week of the year to contain a Thursday, which is equivalent to the first week containing January 4.

The argument must be a date, timestamp, or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is INTEGER. The result can be null; if the argument is null, the result is the null value.

YEAR

The schema is SYSIBM.

The YEAR function returns the year part of a value.

The argument must be a date, timestamp, date duration, timestamp duration or a valid character string representation of a date or timestamp that is neither a CLOB nor a LONG VARCHAR.

The result of the function is a large integer. If the argument can be null, the result can be null; if the argument is null, the result is the null value.

The other rules depend on the data type of the argument specified:

- If the argument is a date, timestamp, or valid string representation of a date or timestamp:
  - The result is the year part of the value, which is an integer between 1 and 9 999.
- If the argument is a date duration or timestamp duration:
  - The result is the year part of the value, which is an integer between –9 999 and 9 999. A nonzero result has the same sign as the argument.

Examples:

- Select all the projects in the PROJECT table that are scheduled to start (PRSTDATE) and end (PRENDATE) in the same calendar year.

```
SELECT * FROM PROJECT
  WHERE YEAR(PRSTDATE) = YEAR(PRENDATE)
```

- Select all the projects in the PROJECT table that are scheduled to take less than one year to complete.

```
SELECT * FROM PROJECT
  WHERE YEAR(PRENDATE - PRSTDATE) < 1
```

Table Functions

A table function can be used only in the FROM clause of a statement. However, expressions or column functions can not be used within a table function.

Table functions returns columns of a table, resembling a table created by a simple CREATE TABLE statement.

The table functions that follow may be qualified with the schema name.

SQLCACHE_SNAPSHOT

The schema is SYSFUN.

The SQLCACHE_SNAPSHOT returns the results of a snapshot of the DB2 dynamic SQL statement cache.

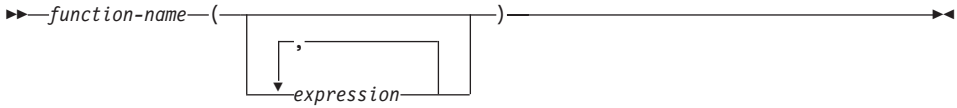
The function does not take any arguments.

It returns a table as listed below. Refer to *System Monitor Guide and Reference* for details on the columns.

Table 17. Column names and data types of the table returned by SQLCACHE_SNAPSHOT table function

Column name	Data type
NUM_EXECUTIONS	INTEGER
NUM_COMPILATIONS	INTEGER
PREP_TIME_WORST	INTEGER
PREP_TIME_BEST	INTEGER
INT_ROWS_DELETED	INTEGER
INT_ROWS_INSERTED	INTEGER
ROWS_READ	INTEGER
INT_ROWS_UPDATED	INTEGER
ROWS_WRITE	INTEGER
STMT_SORTS	INTEGER
TOTAL_EXEC_TIME_S	INTEGER
TOTAL_EXEC_TIME_MS	INTEGER
TOT_U_CPU_TIME_S	INTEGER
TOT_U_CPU_TIME_MS	INTEGER
TOT_S_CPU_TIME_S	INTEGER
TOT_S_CPU_TIME_MS	INTEGER
DB_NAME	VARCHAR(8)
STMT_TEXT	CLOB(64K)

User-Defined Functions



User-defined functions are extensions or additions to the existing built-in functions of the SQL language. A user-defined function can be a scalar function, which returns a single value each time it is called, a column function, which is passed a set of like values and returns a single value for the set, a row function, which returns one row, or a table function, which returns a table. Note that a UDF can be a column function only when it is sourced on an existing column function.

A user-defined scalar or column function registered with the database can be referenced in the same contexts that any built-in function can appear.

A user-defined table function registered with the database can be referenced only in the FROM clause of a SELECT, as described in "from-clause" on page 400.

A user-defined row function can be referenced only implicitly when registered as a transform function for a user-defined type.

A user-defined function is referenced by means of a qualified or unqualified function name, followed by parentheses enclosing the function arguments (if any).

Arguments of the function must correspond in number and position to the parameters specified in the user-defined function as it was registered with the database. In addition, the arguments must be of data types promotable to the data types of the corresponding defined parameters. (see "CREATE FUNCTION" on page 589).

The result of the function is as specified in the RETURNS clause specified when the user-defined function was registered. The RETURNS clause determines if a function is a table function or not.

If the RETURNS NULL ON NULL INPUT clause was specified (or defaulted to) when the function was registered then, if any argument is null, the result is null. For table functions, this is interpreted to mean a return table with no rows (empty table).

There are a collection of user-defined functions provided in the SYSFUN schema (see Table 15 on page 210).

Examples:

- Assume that a scalar function called ADDRESS was written to extract the home address from a script format resume. The ADDRESS function expects a CLOB argument and returns a VARCHAR(4000). The following example illustrates the invocation of the ADDRESS function.

```
SELECT EMPNO, ADDRESS(RESUME) FROM EMP_RESUME
  WHERE RESUME_FORMAT = 'SCRIPT'
```

- Assume a table T2 with a numeric column A and the ADDRESS function described in the previous example. The following example illustrates an attempt to invoke the ADDRESS function with an incorrect argument.

```
SELECT ADDRESS(A) FROM T2
```

  An error (SQLSTATE 42884) is raised since there is no function with a matching name and with a parameter promotable from the argument.

- Assume a table function WHO was written to return information about the sessions on the server machine which were active at the time the statement is executed. The following example illustrates the invocation of WHO in a FROM clause (TABLE keyword with mandatory correlation variable).

```
SELECT ID, START_DATE, ORIG_MACHINE
FROM TABLE( WHO() ) AS QQ
WHERE START_DATE LIKE 'MAY%'
```

  The column names of the WHO() table are defined in the CREATE FUNCTION statement.