9.4. String Functions and Operators

This section describes functions and operators for examining and manipulating string values. Strings in this context include values of the types `character`, `character varying`, and `text`. Unless otherwise noted, all of the functions listed below work on all of these types, but be wary of potential effects of automatic space-padding when using the `character` type. Some functions also exist natively for the bit-string types.

SQL defines some string functions that use key words, rather than commas, to separate arguments. Details are in Table 9-5. PostgreSQL also provides versions of these functions that use the regular function invocation syntax (see Table 9-6).

> **Note:** Before PostgreSQL 8.3, these functions would silently accept values of several non-string data types as well, due to the presence of implicit coercions from those data types to `text`. Those coercions have been removed because they frequently caused surprising

behaviors. However, the string concatenation operator (||) still accepts non-string input, so long as at least one input is of a string type, as shown in Table 9-5. For other cases, insert an explicit coercion to text if you need to duplicate the previous behavior.

Table 9-5. SQL String Functions and Operators

Function	Return Type	Description	Example	Result
string || string	text	String concatenation	'Post' || 'greSQL'	PostgreSQL
string || non-string or non-string || string	text	String concatenation with one non-string input	'Value: ' || 42	Value: 42
bit_length(string)	int	Number of bits in string	bit_length('jose')	32
char_length(string) or character_length(string)	int	Number of characters in string	char_length('jose')	4
lower(string)	text	Convert string to lower case	lower('TOM')	tom
octet_length(string)	int	Number of bytes in string	octet_length('jose')	4
overlay(string placing string from int [for int])	text	Replace substring	overlay('Txxxxas' placing 'hom' from 2 for 4)	Thomas
position(substring in string)	int	Location of specified substring	position('om' in 'Thomas')	3
substring(string [from int] [for int])	text	Extract substring	substring('Thomas' from 2 for 3)	hom
substring(string from *pattern*)	text	Extract substring matching POSIX regular expression. See Section 9.7 for more information on pattern matching.	substring('Thomas' from '...$')	mas

`substring(string from *pattern* for *escape*)`	`text`	Extract substring matching SQL regular expression. See Section 9.7 for more information on pattern matching.	`substring('Thomas' from '%#"o_a#"_' for '#')`	`oma`
`trim([leading | trailing | both] [characters] from string)`	`text`	Remove the longest string containing only the characters (a space by default) from the start/end/both ends of the string	`trim(both 'x' from 'xTomxx')`	`Tom`
`upper(string)`	`text`	Convert string to upper case	`upper('tom')`	`TOM`

Additional string manipulation functions are available and are listed in Table 9-6. Some of them are used internally to implement the SQL-standard string functions listed in Table 9-5.

Table 9-6. Other String Functions

Function	Return Type	Description	Example	Result
`ascii(string)`	`int`	ASCII code of the first character of the argument. For UTF8 returns the Unicode code point of the character. For other multibyte encodings, the argument must be an ASCII character.	`ascii('x')`	`120`
`btrim(string text [, characters text])`	`text`	Remove the longest string consisting only of characters in characters (a space by default) from the start and	`btrim('xyxtrimyyx', 'xy')`	`trim`

		end of string		
chr(int)	text	Character with the given code. For UTF8 the argument is treated as a Unicode code point. For other multibyte encodings the argument must designate an ASCII character. The NULL (0) character is not allowed because text data types cannot store such bytes.	chr(65)	A
concat(str "any" [, str "any" [, ...]])	text	Concatenate all arguments. NULL arguments are ignored.	concat('abcde', 2, NULL, 22)	abcde222
concat_ws(sep text, str "any" [, str "any" [, ...]])	text	Concatenate all but first arguments with separators. The first parameter is used as a separator. NULL arguments are ignored.	concat_ws(',', 'abcde', 2, NULL, 22)	abcde,2,22
		Convert string to dest_encoding. The original encoding is specified by src_encoding. The		

convert(string bytea, src_encoding name, dest_encoding name)	bytea	string must be valid in this encoding. Conversions can be defined by CREATE CONVERSION. Also there are some predefined conversions. See Table 9-7 for available conversions.	convert('text_in_utf8', 'UTF8', 'LATIN1')	text_in_utf8 represented in Latin-1 encoding (ISO 8859-1)
convert_from(string bytea, src_encoding name)	text	Convert string to the database encoding. The original encoding is specified by src_encoding. The string must be valid in this encoding.	convert_from('text_in_utf8', 'UTF8')	text_in_utf8 represented in the current database encoding
convert_to(string text, dest_encoding name)	bytea	Convert string to dest_encoding.	convert_to('some text', 'UTF8')	some text represented in the UTF8 encoding
decode(string text, format text)	bytea	Decode binary data from textual representation in string. Options for format are same as in encode.	decode('MTIzAAE=', 'base64')	\x3132330001
		Encode binary data into a textual		

encode(data bytea, format text)	text	representation. Supported formats are: base64, hex, escape. escape converts zero bytes and high-bit-set bytes to octal sequences (\nnn) and doubles backslashes.	encode(E'123\\000\\001', 'base64')	MTIzAAE=
format(formatstr text [, str "any" [, ...]])	text	Format a string. This function is similar to the C function sprintf; but only the following conversion specifications are recognized: %s interpolates the corresponding argument as a string; %I escapes its argument as an SQL identifier; %L escapes its argument as an SQL literal; %% outputs a literal %. A conversion can reference an explicit parameter position by preceding the	format('Hello %s, %1$s', 'World')	Hello World, World

		conversion specifier with *n*$, where *n* is the argument position. See also Example 39-1.		
`initcap(string)`	`text`	Convert the first letter of each word to upper case and the rest to lower case. Words are sequences of alphanumeric characters separated by non-alphanumeric characters.	`initcap('hi THOMAS')`	`Hi Thomas`
`left(str text, n int)`	`text`	Return first *n* characters in the string. When *n* is negative, return all but last |*n*| characters.	`left('abcde', 2)`	`ab`
`length(string)`	`int`	Number of characters in `string`	`length('jose')`	`4`
`length(string bytea, encoding name )`	`int`	Number of characters in `string` in the given `encoding`. The `string` must be valid in this encoding.	`length('jose', 'UTF8')`	`4`

lpad(string text, length int [, fill text])	text	Fill up the string to length length by prepending the characters fill (a space by default). If the string is already longer than length then it is truncated (on the right).	lpad('hi', 5, 'xy')	xyxhi
ltrim(string text [, characters text])	text	Remove the longest string containing only characters from characters (a space by default) from the start of string	ltrim('zzzytrim', 'xyz')	trim
md5(string)	text	Calculates the MD5 hash of string, returning the result in hexadecimal	md5('abc')	900150983cd24fb0 d6963f7d28e17f72
pg_client_encoding()	name	Current client encoding name	pg_client_encoding()	SQL_ASCII
quote_ident(string text)	text	Return the given string suitably quoted to be used as an identifier in an SQL statement string. Quotes are added only if necessary (i.e., if the string contains non-identifier	quote_ident('Foo bar')	"Foo bar"

		characters or would be case-folded). Embedded quotes are properly doubled. See also Example 39-1.		
`quote_literal(string text)`	`text`	Return the given string suitably quoted to be used as a string literal in an SQL statement string. Embedded single-quotes and backslashes are properly doubled. Note that `quote_literal` returns null on null input; if the argument might be null, `quote_nullable` is often more suitable. See also Example 39-1.	`quote_literal(E'O\'Reilly')`	`'O''Reilly'`
`quote_literal(value anyelement)`	`text`	Coerce the given value to text and then quote it as a literal. Embedded single-quotes and backslashes are properly doubled.	`quote_literal(42.5)`	`'42.5'`
		Return the given string suitably		

quote_nullable(string text)	text	quoted to be used as a string literal in an SQL statement string; or, if the argument is null, return NULL. Embedded single-quotes and backslashes are properly doubled. See also Example 39-1.	quote_nullable(NULL)	NULL
quote_nullable(value anyelement)	text	Coerce the given value to text and then quote it as a literal; or, if the argument is null, return NULL. Embedded single-quotes and backslashes are properly doubled.	quote_nullable(42.5)	'42.5'
regexp_matches(string text, pattern text [, flags text])	setof text[]	Return all captured substrings resulting from matching a POSIX regular expression against the string. See Section 9.7.3 for more information.	regexp_matches('foobarbequebaz', '(bar)(beque)')	{bar,beque}
regexp_replace(string text, pattern text, replacement	text	Replace substring(s) matching a POSIX regular expression	regexp_replace('Thomas',	ThM

pattern text, replacement text [, flags text])	text	regular expression. See Section 9.7.3 for more information.	'.[mN]a.', 'M')	ThM
regexp_split_to_array(string text, pattern text [, flags text])	text[]	Split string using a POSIX regular expression as the delimiter. See Section 9.7.3 for more information.	regexp_split_to_array('hello world', E'\\s+')	{hello,world}
regexp_split_to_table(string text, pattern text [, flags text])	setof text	Split string using a POSIX regular expression as the delimiter. See Section 9.7.3 for more information.	regexp_split_to_table('hello world', E'\\s+')	hello world (2 rows)
repeat(string text, number int)	text	Repeat string the specified number of times	repeat('Pg', 4)	PgPgPgPg
replace(string text, from text, to text)	text	Replace all occurrences in string of substring from with substring to	replace('abcdefabcdef', 'cd', 'XX')	abXXefabXXef
reverse(str)	text	Return reversed string.	reverse('abcde')	edcba
right(str text, n int)	text	Return last *n* characters in the string. When *n* is negative, return all but first |*n*| characters.	right('abcde', 2)	de

rpad(string text, length int [, fill text])	text	Fill up the string to length length by appending the characters fill (a space by default). If the string is already longer than length then it is truncated.	rpad('hi', 5, 'xy')	hixyx
rtrim(string text [, characters text])	text	Remove the longest string containing only characters from characters (a space by default) from the end of string	rtrim('trimxxxx', 'x')	trim
split_part(string text, delimiter text, field int)	text	Split string on delimiter and return the given field (counting from one)	split_part('abc~@~def~@~ghi', '~@~', 2)	def
strpos(string, substring)	int	Location of specified substring (same as position(substring in string), but note the reversed argument order)	strpos('high', 'ig')	2
substr(string, from [, count])	text	Extract substring (same as substring(string from from for count))	substr('alphabet', 3, 2)	ph

to_ascii(string text [, encoding text])	text	Convert string to ASCII from another encoding (only supports conversion from LATIN1, LATIN2, LATIN9, and WIN1250 encodings)	to_ascii('Karel')	Karel
to_hex(number int or bigint)	text	Convert number to its equivalent hexadecimal representation	to_hex(2147483647)	7fffffff
translate(string text, from text, to text)	text	Any character in string that matches a character in the from set is replaced by the corresponding character in the to set. If from is longer than to, occurrences of the extra characters in from are removed.	translate('12345', '143', 'ax')	a2x5

See also the aggregate function string_agg in Section 9.18.

Table 9-7. Built-in Conversions

Conversion Name [a]	Source Encoding	Destination Encoding
ascii_to_mic	SQL_ASCII	MULE_INTERNAL

ascii_to_utf8	SQL_ASCII	UTF8
big5_to_euc_tw	BIG5	EUC_TW
big5_to_mic	BIG5	MULE_INTERNAL
big5_to_utf8	BIG5	UTF8
euc_cn_to_mic	EUC_CN	MULE_INTERNAL
euc_cn_to_utf8	EUC_CN	UTF8
euc_jp_to_mic	EUC_JP	MULE_INTERNAL
euc_jp_to_sjis	EUC_JP	SJIS
euc_jp_to_utf8	EUC_JP	UTF8
euc_kr_to_mic	EUC_KR	MULE_INTERNAL
euc_kr_to_utf8	EUC_KR	UTF8
euc_tw_to_big5	EUC_TW	BIG5
euc_tw_to_mic	EUC_TW	MULE_INTERNAL
euc_tw_to_utf8	EUC_TW	UTF8
gb18030_to_utf8	GB18030	UTF8
gbk_to_utf8	GBK	UTF8
iso_8859_10_to_utf8	LATIN6	UTF8
iso_8859_13_to_utf8	LATIN7	UTF8
iso_8859_14_to_utf8	LATIN8	UTF8
iso_8859_15_to_utf8	LATIN9	UTF8
iso_8859_16_to_utf8	LATIN10	UTF8
iso_8859_1_to_mic	LATIN1	MULE_INTERNAL
iso_8859_1_to_utf8	LATIN1	UTF8
iso_8859_2_to_mic	LATIN2	MULE_INTERNAL

iso_8859_2_to_utf8	LATIN2	UTF8
iso_8859_2_to_windows_1250	LATIN2	WIN1250
iso_8859_3_to_mic	LATIN3	MULE_INTERNAL
iso_8859_3_to_utf8	LATIN3	UTF8
iso_8859_4_to_mic	LATIN4	MULE_INTERNAL
iso_8859_4_to_utf8	LATIN4	UTF8
iso_8859_5_to_koi8_r	ISO_8859_5	KOI8R
iso_8859_5_to_mic	ISO_8859_5	MULE_INTERNAL
iso_8859_5_to_utf8	ISO_8859_5	UTF8
iso_8859_5_to_windows_1251	ISO_8859_5	WIN1251
iso_8859_5_to_windows_866	ISO_8859_5	WIN866
iso_8859_6_to_utf8	ISO_8859_6	UTF8
iso_8859_7_to_utf8	ISO_8859_7	UTF8
iso_8859_8_to_utf8	ISO_8859_8	UTF8
iso_8859_9_to_utf8	LATIN5	UTF8
johab_to_utf8	JOHAB	UTF8
koi8_r_to_iso_8859_5	KOI8R	ISO_8859_5
koi8_r_to_mic	KOI8R	MULE_INTERNAL
koi8_r_to_utf8	KOI8R	UTF8
koi8_r_to_windows_1251	KOI8R	WIN1251
koi8_r_to_windows_866	KOI8R	WIN866
koi8_u_to_utf8	KOI8U	UTF8
mic_to_ascii	MULE_INTERNAL	SQL_ASCII

mic_to_big5	MULE_INTERNAL	BIG5
mic_to_euc_cn	MULE_INTERNAL	EUC_CN
mic_to_euc_jp	MULE_INTERNAL	EUC_JP
mic_to_euc_kr	MULE_INTERNAL	EUC_KR
mic_to_euc_tw	MULE_INTERNAL	EUC_TW
mic_to_iso_8859_1	MULE_INTERNAL	LATIN1
mic_to_iso_8859_2	MULE_INTERNAL	LATIN2
mic_to_iso_8859_3	MULE_INTERNAL	LATIN3
mic_to_iso_8859_4	MULE_INTERNAL	LATIN4
mic_to_iso_8859_5	MULE_INTERNAL	ISO_8859_5
mic_to_koi8_r	MULE_INTERNAL	KOI8R
mic_to_sjis	MULE_INTERNAL	SJIS
mic_to_windows_1250	MULE_INTERNAL	WIN1250
mic_to_windows_1251	MULE_INTERNAL	WIN1251
mic_to_windows_866	MULE_INTERNAL	WIN866
sjis_to_euc_jp	SJIS	EUC_JP
sjis_to_mic	SJIS	MULE_INTERNAL
sjis_to_utf8	SJIS	UTF8
tcvn_to_utf8	WIN1258	UTF8
uhc_to_utf8	UHC	UTF8
utf8_to_ascii	UTF8	SQL_ASCII
utf8_to_big5	UTF8	BIG5
utf8_to_euc_cn	UTF8	EUC_CN
utf8_to_euc_jp	UTF8	EUC_JP

utf8_to_euc_kr	UTF8	EUC_KR
utf8_to_euc_tw	UTF8	EUC_TW
utf8_to_gb18030	UTF8	GB18030
utf8_to_gbk	UTF8	GBK
utf8_to_iso_8859_1	UTF8	LATIN1
utf8_to_iso_8859_10	UTF8	LATIN6
utf8_to_iso_8859_13	UTF8	LATIN7
utf8_to_iso_8859_14	UTF8	LATIN8
utf8_to_iso_8859_15	UTF8	LATIN9
utf8_to_iso_8859_16	UTF8	LATIN10
utf8_to_iso_8859_2	UTF8	LATIN2
utf8_to_iso_8859_3	UTF8	LATIN3
utf8_to_iso_8859_4	UTF8	LATIN4
utf8_to_iso_8859_5	UTF8	ISO_8859_5
utf8_to_iso_8859_6	UTF8	ISO_8859_6
utf8_to_iso_8859_7	UTF8	ISO_8859_7
utf8_to_iso_8859_8	UTF8	ISO_8859_8
utf8_to_iso_8859_9	UTF8	LATIN5
utf8_to_johab	UTF8	JOHAB
utf8_to_koi8_r	UTF8	KOI8R
utf8_to_koi8_u	UTF8	KOI8U
utf8_to_sjis	UTF8	SJIS
utf8_to_tcvn	UTF8	WIN1258

utf8_to_uhc	UTF8	UHC
utf8_to_windows_1250	UTF8	WIN1250
utf8_to_windows_1251	UTF8	WIN1251
utf8_to_windows_1252	UTF8	WIN1252
utf8_to_windows_1253	UTF8	WIN1253
utf8_to_windows_1254	UTF8	WIN1254
utf8_to_windows_1255	UTF8	WIN1255
utf8_to_windows_1256	UTF8	WIN1256
utf8_to_windows_1257	UTF8	WIN1257
utf8_to_windows_866	UTF8	WIN866
utf8_to_windows_874	UTF8	WIN874
windows_1250_to_iso_8859_2	WIN1250	LATIN2
windows_1250_to_mic	WIN1250	MULE_INTERNAL
windows_1250_to_utf8	WIN1250	UTF8
windows_1251_to_iso_8859_5	WIN1251	ISO_8859_5
windows_1251_to_koi8_r	WIN1251	KOI8R
windows_1251_to_mic	WIN1251	MULE_INTERNAL
windows_1251_to_utf8	WIN1251	UTF8
windows_1251_to_windows_866	WIN1251	WIN866
windows_1252_to_utf8	WIN1252	UTF8
windows_1256_to_utf8	WIN1256	UTF8
windows_866_to_iso_8859_5	WIN866	ISO_8859_5
windows_866_to_koi8_r	WIN866	KOI8R
windows_866_to_mic	WIN866	MULE_INTERNAL

windows_866_to_utf8	WIN866	UTF8
windows_866_to_windows_1251	WIN866	WIN
windows_874_to_utf8	WIN874	UTF8
euc_jis_2004_to_utf8	EUC_JIS_2004	UTF8
utf8_to_euc_jis_2004	UTF8	EUC_JIS_2004
shift_jis_2004_to_utf8	SHIFT_JIS_2004	UTF8
utf8_to_shift_jis_2004	UTF8	SHIFT_JIS_2004
euc_jis_2004_to_shift_jis_2004	EUC_JIS_2004	SHIFT_JIS_2004
shift_jis_2004_to_euc_jis_2004	SHIFT_JIS_2004	EUC_JIS_2004

Notes:

a. The conversion names follow a standard naming scheme: The official name of the source encoding with all non-alphanumeric characters replaced by underscores, followed by `_to_`, followed by the similarly processed destination encoding name. Therefore, the names might deviate from the customary encoding names.

Prev | Home | Next
Mathematical Functions and Operators | Up | Binary String Functions and Operators