Package: dd (via r-universe)

October 10, 2026

Title Documentation for Functions Provided by DuckDB

Version 0.0.0.9000

Date 2025-04-10

Description Lists DuckDB functions for integration in R's help system.

License MIT + file LICENSE

URL https://github.com/cynkra/dd, https://cynkra.github.io/dd

BugReports https://github.com/cynkra/dd/issues

Depends R (>= 4.4)

Config/Needs/website cynkra/cynkratemplate

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1.9000

Repository https://cynkra.r-universe.dev

Date/Publication 2026-10-10 01:45:02 UTC

RemoteUrl https://github.com/cynkra/dd

RemoteRef HEAD

RemoteSha c2edbc36f2dc5e66566ae1b842eca3e1d1b0e5d9

Contents

/ 25
& 25
~~* 26
~~~ 26
~~ 27
~ 27
<<= 28
<< 28
>>= 29
>> 29
~~~

abs . . . 30
acos . . . 30
acosh . . . 31
add . . . 31
add_parquet_key . . . 33
age . . . 33
ago . . . 34
alias . . . 34
all_profiling_output . . . 35
any_value . . . 35
approx_count_distinct . . . 36
approx_quantile . . . 36
approx_top_k . . . 37
arg_max . . . 37
arg_max_null . . . 40
arg_max_nulls_last . . . 41
arg_min . . . 44
arg_min_null . . . 47
arg_min_nulls_last . . . 47
array_agg . . . 50
array_append . . . 51
array_cosine_distance . . . 51
array_cosine_similarity . . . 52
array_cross_product . . . 53
array_distance . . . 53
array_extract . . . 54
array_inner_product . . . 55
array_length . . . 56
array_negative_inner_product . . . 57
array_pop_back . . . 57
array_pop_front . . . 58
array_prepend . . . 58
array_push_back . . . 59
array_push_front . . . 59
array_reverse . . . 60
array_to_json . . . 60
array_to_string . . . 61
array_to_string_comma_default . . . 61
array_value . . . 62
arrow->> . . . 62
arrow_scan . . . 63
arrow_scan_dumb . . . 63
ascii . . . 64
asin . . . 64
asinh . . . 65
atan . . . 65
atan2 . . . 66
atanh . . . 66

avg . . . 67
bar . . . 67
bin . . . 68
bit_and . . . 69
bit_count . . . 70
bit_length . . . 70
bit_or . . . 71
bit_position . . . 72
bit_xor . . . 72
bitstring . . . 73
bitstring_agg . . . 73
bool_and . . . 74
bool_or . . . 75
broadcast . . . 75
can_cast_implicitly . . . 76
cardinality . . . 76
cast_to_type . . . 77
cbrt . . . 77
ceil . . . 78
century . . . 78
check_peg_parser . . . 79
checkpoint . . . 79
chr . . . 80
col_description . . . 80
collations . . . 81
combine . . . 81
concat . . . 82
concat_ws . . . 83
constant_or_null . . . 83
contains . . . 84
copy_database . . . 85
copy_dir . . . 86
corr . . . 86
cos . . . 87
cosh . . . 87
cot . . . 88
count . . . 88
count_if . . . 89
count_star . . . 89
covar_pop . . . 90
covar_samp . . . 90
create_fts_index . . . 91
create_sort_key . . . 91
cume_dist . . . 92
current_catalog . . . 92
current_connection_id . . . 93
current_database . . . 93
current_date . . . 94

current_localtime . . . 94
current_localtimestamp . . . 95
current_query . . . 95
current_query_id . . . 96
current_role . . . 96
current_schema . . . 97
current_schemas . . . 97
current_setting . . . 98
current_transaction_id . . . 98
current_user . . . 99
currval . . . 99
damerau_levenshtein . . . 100
database_list . . . 100
database_size . . . 101
date_add . . . 101
date_diff . . . 102
date_part . . . 102
date_sub . . . 103
date_trunc . . . 103
day . . . 104
dayname . . . 104
dayofmonth . . . 105
dayofweek . . . 105
dayofyear . . . 106
days_in_month . . . 106
dbgen . . . 107
dd . . . 107
decade . . . 108
decode . . . 108
degrees . . . 109
delta_domain_metadata . . . 110
delta_filter_pushdown_log . . . 110
delta_filter_pushdown_log_tpcds . . . 111
delta_get_transaction_version . . . 111
delta_list_files . . . 112
delta_scan . . . 112
delta_set_transaction_version . . . 114
dense_rank . . . 114
disable_checkpoint_on_shutdown . . . 115
disable_logging . . . 115
disable_object_cache . . . 115
disable_optimizer . . . 116
disable_peg_parser . . . 116
disable_print_progress_bar . . . 117
disable_profile . . . 117
disable_profiling . . . 117
disable_progress_bar . . . 118
disable_verification . . . 118

disable_verify_external . . . 118
disable_verify_fetch_row . . . 119
disable_verify_parallelism . . . 119
disable_verify_serializer . . . 119
divide . . . 120
drop_fts_index . . . 120
dsdgen . . . 121
duckdb_approx_database_count . . . 121
duckdb_columns . . . 122
duckdb_connection_count . . . 122
duckdb_constraints . . . 122
duckdb_coordinate_systems . . . 123
duckdb_databases . . . 123
duckdb_dependencies . . . 123
duckdb_extensions . . . 124
duckdb_external_file_cache . . . 124
duckdb_functions . . . 124
duckdb_indexes . . . 125
duckdb_keywords . . . 125
duckdb_log_contexts . . . 125
duckdb_logs . . . 126
duckdb_logs_parsed . . . 126
duckdb_memory . . . 127
duckdb_optimizers . . . 127
duckdb_prepared_statements . . . 127
duckdb_profiling_settings . . . 128
DuckDB_PROJ_Compiled_Version . . . 128
DuckDB_Proj_Version . . . 129
duckdb_schemas . . . 129
duckdb_secret_types . . . 130
duckdb_secrets . . . 130
duckdb_sequences . . . 130
duckdb_settings . . . 131
duckdb_table_sample . . . 131
duckdb_tables . . . 131
duckdb_temporary_files . . . 132
duckdb_types . . . 132
duckdb_variables . . . 132
duckdb_views . . . 133
ducklake_add_data_files . . . 133
ducklake_cleanup_old_files . . . 134
ducklake_commit . . . 135
ducklake_current_snapshot . . . 135
ducklake_delete_orphaned_files . . . 136
ducklake_expire_snapshots . . . 137
ducklake_flush_inlined_data . . . 137
ducklake_last_committed_snapshot . . . 138
ducklake_list_files . . . 138

ducklake_merge_adjacent_files . . . 139
ducklake_options . . . 140
ducklake_rewrite_data_files . . . 140
ducklake_scan . . . 141
ducklake_set_commit_message . . . 142
ducklake_set_option . . . 143
ducklake_settings . . . 143
ducklake_snapshots . . . 144
ducklake_table_changes . . . 144
ducklake_table_deletions . . . 145
ducklake_table_info . . . 146
ducklake_table_insertions . . . 146
enable_checkpoint_on_shutdown . . . 147
enable_logging . . . 147
enable_object_cache . . . 148
enable_optimizer . . . 148
enable_peg_parser . . . 149
enable_print_progress_bar . . . 149
enable_profile . . . 150
enable_profiling . . . 150
enable_progress_bar . . . 151
enable_verification . . . 151
encode . . . 151
entropy . . . 152
enum_code . . . 152
enum_first . . . 153
enum_last . . . 153
enum_range . . . 154
enum_range_boundary . . . 154
epoch . . . 155
epoch_ms . . . 155
epoch_ns . . . 156
epoch_us . . . 156
equi_width_bins . . . 157
era . . . 157
error . . . 158
even . . . 158
excel_text . . . 159
exp . . . 159
extension_versions . . . 160
factorial . . . 160
family . . . 161
favg . . . 161
fdiv . . . 162
fill . . . 162
finalize . . . 163
first . . . 163
first_value . . . 164

flatten . . . 164
floor . . . 165
fmod . . . 165
force_checkpoint . . . 166
format . . . 166
format_bytes . . . 167
format_pg_type . . . 168
format_type . . . 168
formatReadableDecimalSize . . . 169
from_base64 . . . 170
from_json . . . 171
from_json_strict . . . 171
functions . . . 172
gamma . . . 172
generate_series . . . 173
generate_subscripts . . . 174
geomean . . . 175
geometric_mean . . . 175
get_bit . . . 176
get_block_size . . . 176
get_current_time . . . 177
get_current_timestamp . . . 177
get_delta_test_expression . . . 178
get_type . . . 178
get_ui_url . . . 179
getvariable . . . 179
glob . . . 180
greatest . . . 180
greatest_common_divisor . . . 181
hamming . . . 182
has_any_column_privilege . . . 182
has_column_privilege . . . 183
has_database_privilege . . . 183
has_foreign_data_wrapper_privilege . . . 184
has_function_privilege . . . 185
has_language_privilege . . . 185
has_schema_privilege . . . 186
has_sequence_privilege . . . 186
has_server_privilege . . . 187
has_table_privilege . . . 187
has_tablespace_privilege . . . 188
hash . . . 189
hex . . . 189
histogram . . . 191
histogram_exact . . . 191
histogram_values . . . 192
hnsw_compact_index . . . 192
hnsw_index_scan . . . 193

host . . . 193
hour . . . 194
html_escape . . . 194
html_unescape . . . 195
iceberg_bucket . . . 195
iceberg_column_stats . . . 196
iceberg_load_table_response . . . 197
iceberg_metadata . . . 197
iceberg_partition_stats . . . 198
iceberg_scan . . . 199
iceberg_schema_properties . . . 201
iceberg_snapshots . . . 201
iceberg_table_properties . . . 202
iceberg_to_ducklake . . . 202
iceberg_truncate . . . 203
iceberg_verify_equality_deletes . . . 203
icu_calendar_names . . . 204
icu_collate_af . . . 204
icu_collate_am . . . 205
icu_collate_ar . . . 205
icu_collate_ar_sa . . . 206
icu_collate_as . . . 206
icu_collate_az . . . 207
icu_collate_be . . . 207
icu_collate_bg . . . 208
icu_collate_bn . . . 208
icu_collate_bo . . . 209
icu_collate_br . . . 209
icu_collate_bs . . . 210
icu_collate_ca . . . 210
icu_collate_ceb . . . 211
icu_collate_chr . . . 211
icu_collate_cs . . . 212
icu_collate_cy . . . 212
icu_collate_da . . . 213
icu_collate_de . . . 213
icu_collate_de_at . . . 214
icu_collate_dsb . . . 214
icu_collate_dz . . . 215
icu_collate_ee . . . 215
icu_collate_el . . . 216
icu_collate_en . . . 216
icu_collate_en_us . . . 217
icu_collate_eo . . . 217
icu_collate_es . . . 218
icu_collate_et . . . 218
icu_collate_fa . . . 219
icu_collate_fa_af . . . 219

icu_collate_ff . . . 220
icu_collate_fi . . . 220
icu_collate_fil . . . 221
icu_collate_fo . . . 221
icu_collate_fr . . . 222
icu_collate_fr_ca . . . 222
icu_collate_fy . . . 223
icu_collate_ga . . . 223
icu_collate_gl . . . 224
icu_collate_gu . . . 224
icu_collate_ha . . . 225
icu_collate_haw . . . 225
icu_collate_he . . . 226
icu_collate_he_il . . . 226
icu_collate_hi . . . 227
icu_collate_hr . . . 227
icu_collate_hsb . . . 228
icu_collate_hu . . . 228
icu_collate_hy . . . 229
icu_collate_id . . . 229
icu_collate_id_id . . . 230
icu_collate_ig . . . 230
icu_collate_is . . . 231
icu_collate_it . . . 231
icu_collate_ja . . . 232
icu_collate_ka . . . 232
icu_collate_kk . . . 233
icu_collate_kl . . . 233
icu_collate_km . . . 234
icu_collate_kn . . . 234
icu_collate_ko . . . 235
icu_collate_kok . . . 235
icu_collate_ku . . . 236
icu_collate_ky . . . 236
icu_collate_lb . . . 237
icu_collate_lij . . . 237
icu_collate_lkt . . . 238
icu_collate_ln . . . 238
icu_collate_lo . . . 239
icu_collate_lt . . . 239
icu_collate_lv . . . 240
icu_collate_mk . . . 240
icu_collate_ml . . . 241
icu_collate_mn . . . 241
icu_collate_mr . . . 242
icu_collate_ms . . . 242
icu_collate_mt . . . 243
icu_collate_my . . . 243

icu_collate_nb . . . 244
icu_collate_nb_no . . . 244
icu_collate_ne . . . 245
icu_collate_nl . . . 245
icu_collate_nn . . . 246
icu_collate_noaccent . . . 246
icu_collate_nso . . . 247
icu_collate_om . . . 247
icu_collate_or . . . 248
icu_collate_pa . . . 248
icu_collate_pa_in . . . 249
icu_collate_pl . . . 249
icu_collate_ps . . . 250
icu_collate_pt . . . 250
icu_collate_ro . . . 251
icu_collate_ru . . . 251
icu_collate_sa . . . 252
icu_collate_se . . . 252
icu_collate_si . . . 253
icu_collate_sk . . . 253
icu_collate_sl . . . 254
icu_collate_smn . . . 254
icu_collate_sq . . . 255
icu_collate_sr . . . 255
icu_collate_sr_ba . . . 256
icu_collate_sr_me . . . 256
icu_collate_sr_rs . . . 257
icu_collate_st . . . 257
icu_collate_sv . . . 258
icu_collate_sw . . . 258
icu_collate_ta . . . 259
icu_collate_te . . . 259
icu_collate_th . . . 260
icu_collate_tk . . . 260
icu_collate_tn . . . 261
icu_collate_to . . . 261
icu_collate_tr . . . 262
icu_collate_ug . . . 262
icu_collate_uk . . . 263
icu_collate_ur . . . 263
icu_collate_uz . . . 264
icu_collate_vi . . . 264
icu_collate_wae . . . 265
icu_collate_wo . . . 265
icu_collate_xh . . . 266
icu_collate_yi . . . 266
icu_collate_yo . . . 267
icu_collate_yue . . . 267

icu_collate_yue_cn . . . 268
icu_collate_zh . . . 268
icu_collate_zh_cn . . . 269
icu_collate_zh_hk . . . 269
icu_collate_zh_mo . . . 270
icu_collate_zh_sg . . . 270
icu_collate_zh_tw . . . 271
icu_collate_zu . . . 271
icu_sort_key . . . 272
ilike_escape . . . 272
import_database . . . 273
in_search_path . . . 273
inet_client_addr . . . 274
inet_client_port . . . 274
inet_server_addr . . . 275
inet_server_port . . . 275
instr . . . 275
is_histogram_other_bin . . . 276
isfinite . . . 277
isinf . . . 277
isnan . . . 278
isodow . . . 278
isoyear . . . 279
jaccard . . . 279
jaro_similarity . . . 280
jaro_winkler_similarity . . . 281
json . . . 282
json_array . . . 282
json_array_length . . . 283
json_contains . . . 283
json_deserialize_sql . . . 284
json_each . . . 284
json_execute_serialized_sql . . . 285
json_exists . . . 286
json_extract . . . 286
json_extract_path . . . 287
json_extract_path_text . . . 287
json_extract_string . . . 288
json_group_array . . . 288
json_group_object . . . 289
json_group_structure . . . 289
json_keys . . . 290
json_merge_patch . . . 290
json_object . . . 291
json_pretty . . . 291
json_quote . . . 292
json_serialize_plan . . . 292
json_serialize_sql . . . 293

json_structure . . . 294
json_transform . . . 294
json_transform_strict . . . 295
json_tree . . . 295
json_type . . . 296
json_valid . . . 297
json_value . . . 297
julian . . . 298
kahan_sum . . . 298
kurtosis . . . 299
kurtosis_pop . . . 299
lag . . . 300
lance_fts . . . 300
lance_hybrid_search . . . 301
lance_vector_search . . . 302
last . . . 303
last_day . . . 303
last_value . . . 304
lead . . . 304
least . . . 305
least_common_multiple . . . 306
left . . . 306
left_grapheme . . . 307
len . . . 308
length_grapheme . . . 309
levenshtein . . . 310
lgamma . . . 310
like_escape . . . 311
list_aggregate . . . 312
list_any_value . . . 312
list_append . . . 313
list_approx_count_distinct . . . 313
list_avg . . . 314
list_bit_and . . . 314
list_bit_or . . . 315
list_bit_xor . . . 315
list_bool_and . . . 316
list_bool_or . . . 316
list_concat . . . 317
list_contains . . . 317
list_cosine_distance . . . 318
list_cosine_similarity . . . 319
list_count . . . 319
list_distance . . . 320
list_distinct . . . 321
list_entropy . . . 321
list_extract . . . 322
list_filter . . . 323

list_first . . . 324
list_grade_up . . . 324
list_has_all . . . 325
list_has_any . . . 326
list_histogram . . . 326
list_inner_product . . . 327
list_intersect . . . 328
list_kurtosis . . . 328
list_kurtosis_pop . . . 329
list_last . . . 329
list_mad . . . 330
list_max . . . 330
list_median . . . 331
list_min . . . 331
list_mode . . . 332
list_negative_inner_product . . . 332
list_position . . . 333
list_prepend . . . 334
list_product . . . 334
list_reduce . . . 335
list_resize . . . 336
list_reverse . . . 337
list_reverse_sort . . . 337
list_select . . . 338
list_sem . . . 339
list_skewness . . . 339
list_slice . . . 340
list_sort . . . 341
list_stddev_pop . . . 342
list_stddev_samp . . . 342
list_string_agg . . . 343
list_sum . . . 343
list_transform . . . 344
list_unique . . . 344
list_value . . . 345
list_var_pop . . . 346
list_var_samp . . . 346
list_where . . . 347
list_zip . . . 347
ln . . . 348
load_aws_credentials . . . 349
log . . . 349
log10 . . . 350
log2 . . . 351
lower . . . 351
lpad . . . 352
ltrim . . . 353
mad . . . 354

make_date . . . 354
make_time . . . 355
make_timestamp . . . 356
make_timestamp_ms . . . 356
make_timestamp_ns . . . 357
make_timestamptz . . . 357
make_type . . . 358
map . . . 359
map_concat . . . 359
map_contains . . . 360
map_contains_entry . . . 360
map_contains_value . . . 361
map_entries . . . 361
map_extract . . . 362
map_extract_value . . . 362
map_from_entries . . . 363
map_keys . . . 363
map_to_pg_oid . . . 364
map_values . . . 364
max . . . 365
md5 . . . 365
md5_number . . . 366
md5_number_lower . . . 367
md5_number_upper . . . 368
median . . . 368
metadata_info . . . 369
microsecond . . . 369
millennium . . . 370
millisecond . . . 370
min . . . 371
minute . . . 371
mod . . . 372
mode . . . 372
month . . . 373
monthname . . . 373
multiply . . . 374
murmur3_32 . . . 374
mysql_clear_cache . . . 375
mysql_debug_execution_plan . . . 375
mysql_execute . . . 376
mysql_explain_federated . . . 376
mysql_query . . . 377
nanosecond . . . 377
netmask . . . 378
network . . . 378
nextafter . . . 379
nextval . . . 379
nfc_normalize . . . 380

normalized_interval . . . 380
not-~~* . . . 381
not-~~ . . . 381
not_ilike_escape . . . 382
not_like_escape . . . 383
nth_value . . . 384
ntile . . . 384
nullif . . . 385
obj_description . . . 385
octet_length . . . 386
or- . . . 386
or– or . . . 387
parquet_bloom_probe . . . 388
parquet_file_metadata . . . 388
parquet_full_metadata . . . 389
parquet_kv_metadata . . . 389
parquet_metadata . . . 390
parquet_scan . . . 390
parquet_schema . . . 391
parse_delta_filter_logline . . . 392
parse_dirname . . . 392
parse_dirpath . . . 393
parse_duckdb_log_message . . . 394
parse_filename . . . 395
parse_formatted_bytes . . . 396
parse_path . . . 397
percent_rank . . . 398
pg_clear_cache . . . 398
pg_collation_is_visible . . . 399
pg_conf_load_time . . . 399
pg_conversion_is_visible . . . 400
pg_function_is_visible . . . 400
pg_get_constraintdef . . . 401
pg_get_expr . . . 401
pg_get_viewdef . . . 402
pg_has_role . . . 402
pg_is_other_temp_schema . . . 403
pg_my_temp_schema . . . 403
pg_opclass_is_visible . . . 404
pg_operator_is_visible . . . 404
pg_opfamily_is_visible . . . 405
pg_postmaster_start_time . . . 405
pg_size_pretty . . . 406
pg_sleep . . . 406
pg_table_is_visible . . . 407
pg_timezone_names . . . 407
pg_ts_config_is_visible . . . 408
pg_ts_dict_is_visible . . . 408

pg_ts_parser_is_visible . . . 409
pg_ts_template_is_visible . . . 409
pg_type_is_visible . . . 410
pg_typeof . . . 410
pi . . . 411
platform . . . 411
postgres_attach . . . 412
postgres_configure_pool . . . 412
postgres_execute . . . 414
postgres_hstore_get . . . 414
postgres_hstore_to_json . . . 415
postgres_query . . . 415
postgres_scan . . . 416
postgres_scan_pushdown . . . 416
pow . . . 417
pragma_collations . . . 417
pragma_database_size . . . 418
pragma_hnsw_index_info . . . 418
pragma_metadata_info . . . 419
pragma_platform . . . 419
pragma_rtree_index_info . . . 420
pragma_show . . . 420
pragma_storage_info . . . 421
pragma_table_info . . . 421
pragma_user_agent . . . 422
pragma_version . . . 422
prefix . . . 422
printf . . . 423
product . . . 424
quack_active_connections . . . 424
quack_check_token . . . 425
quack_clear_cache . . . 425
quack_identify . . . 426
quack_nop_authorization . . . 426
quack_query . . . 427
quack_query_by_name . . . 427
quack_serve . . . 428
quack_server_list . . . 428
quack_stop . . . 429
quack_uri_parser . . . 429
quantile_cont . . . 430
quantile_disc . . . 430
quarter . . . 431
query . . . 432
query_table . . . 432
r_dataframe_scan . . . 433
radians . . . 433
random . . . 434

range . . . 434
rank . . . 436
rank_dense . . . 436
read_avro . . . 436
read_blob . . . 437
read_csv . . . 438
read_csv_auto . . . 440
read_duckdb . . . 443
read_json . . . 444
read_json_auto . . . 446
read_json_objects . . . 447
read_json_objects_auto . . . 449
read_ndjson . . . 450
read_ndjson_auto . . . 451
read_ndjson_objects . . . 453
read_parquet . . . 454
read_postgres_binary . . . 455
read_text . . . 456
read_vortex . . . 456
read_xlsx . . . 457
regexp_escape . . . 458
regexp_extract . . . 458
regexp_extract_all . . . 460
regexp_full_match . . . 461
regexp_matches . . . 461
regexp_replace . . . 462
regexp_split_to_table . . . 463
regr_avgx . . . 463
regr_avgy . . . 464
regr_count . . . 464
regr_intercept . . . 465
regr_r2 . . . 465
regr_slope . . . 466
regr_sxx . . . 466
regr_sxy . . . 467
regr_syy . . . 467
remap_struct . . . 468
remove_iceberg_schema_properties . . . 468
remove_iceberg_table_properties . . . 469
repeat . . . 469
repeat_row . . . 470
replace . . . 471
replace_type . . . 472
reservoir_quantile . . . 472
reverse . . . 474
right . . . 474
right_grapheme . . . 475
round . . . 476

round_even . . . 477
roundbankers . . . 477
row . . . 478
row_number . . . 478
row_to_json . . . 479
rpad . . . 479
rtree_index_dump . . . 480
rtree_index_scan . . . 480
rtrim . . . 481
second . . . 482
sem . . . 482
seq_scan . . . 483
session_user . . . 483
set_bit . . . 483
set_iceberg_schema_properties . . . 484
set_iceberg_table_properties . . . 485
setseed . . . 485
sha1 . . . 486
sha256 . . . 487
shapefile_meta . . . 488
shobj_description . . . 488
show . . . 489
show_databases . . . 489
show_tables . . . 489
show_tables_expanded . . . 490
sign . . . 490
signbit . . . 491
sin . . . 491
sinh . . . 492
skewness . . . 492
sleep_ms . . . 493
sniff_csv . . . 493
split_part . . . 496
sql_auto_complete . . . 496
sqlite_attach . . . 497
sqlite_query . . . 498
sqlite_scan . . . 498
sqrt . . . 499
ST_Affine . . . 499
ST_Area . . . 501
ST_Area_Spheroid . . . 502
ST_AsGeoJSON . . . 502
ST_AsHEXWKB . . . 503
ST_AsMVT . . . 504
ST_AsMVTGeom . . . 506
ST_AsSVG . . . 507
st_astext . . . 507
st_aswkb . . . 509

ST_Azimuth . . . 509
ST_Boundary . . . 510
ST_Buffer . . . 510
ST_BuildArea . . . 511
ST_Centroid . . . 512
ST_ClosestPoint . . . 513
ST_Collect . . . 513
ST_CollectionExtract . . . 514
ST_ConcaveHull . . . 515
ST_Contains . . . 516
ST_ContainsProperly . . . 517
ST_ConvexHull . . . 517
ST_CoverageClean . . . 518
ST_CoverageInvalidEdges . . . 519
ST_CoverageInvalidEdges_Agg . . . 519
ST_CoverageSimplify . . . 520
ST_CoverageSimplify_Agg . . . 521
ST_CoverageUnion . . . 522
ST_CoverageUnion_Agg . . . 522
ST_CoveredBy . . . 523
ST_Covers . . . 523
ST_Crosses . . . 524
st_crs . . . 524
ST_Difference . . . 525
ST_Dimension . . . 525
ST_Disjoint . . . 526
ST_Distance . . . 526
ST_Distance_GEOS . . . 527
ST_Distance_Sphere . . . 528
ST_Distance_Spheroid . . . 528
ST_Drivers . . . 529
ST_Dump . . . 530
ST_DWithin . . . 531
ST_DWithin_GEOS . . . 531
ST_DWithin_Spheroid . . . 532
ST_EndPoint . . . 533
ST_Envelope . . . 533
ST_Envelope_Agg . . . 534
ST_Equals . . . 534
ST_Expand . . . 535
ST_Extent . . . 536
ST_Extent_Agg . . . 536
ST_Extent_Approx . . . 537
ST_ExteriorRing . . . 537
ST_FlipCoordinates . . . 538
ST_Force2D . . . 539
ST_Force3DM . . . 539
ST_Force3DZ . . . 540

ST_Force4D . . . 541
ST_GeneratePoints . . . 542
ST_GeometryType . . . 543
ST_GeomFromGeoJSON . . . 544
ST_GeomFromHEXEWKB . . . 544
ST_GeomFromHEXWKB . . . 545
ST_GeomFromText . . . 545
st_geomfromwkb . . . 546
ST_HasM . . . 547
ST_HasZ . . . 548
ST_Hilbert . . . 549
ST_InteriorRingN . . . 549
ST_InterpolatePoint . . . 550
ST_Intersection . . . 551
ST_Intersection_Agg . . . 551
ST_Intersects . . . 552
ST_IsClosed . . . 552
ST_IsEmpty . . . 553
ST_IsRing . . . 553
ST_IsSimple . . . 554
ST_IsValid . . . 554
ST_Length . . . 555
ST_Length_Spheroid . . . 556
ST_LineInterpolatePoint . . . 556
ST_LineInterpolatePoints . . . 557
ST_LineLocatePoint . . . 558
ST_LineMerge . . . 558
ST_LineString2DFromWKB . . . 559
ST_LineSubstring . . . 559
ST_LocateAlong . . . 560
ST_LocateBetween . . . 561
ST_M . . . 562
ST_MakeBox2D . . . 562
ST_MakeEnvelope . . . 563
ST_MakeLine . . . 563
ST_MakePoint . . . 564
ST_MakePolygon . . . 565
ST_MakeValid . . . 566
ST_MaximumInscribedCircle . . . 566
ST_MemUnion_Agg . . . 567
ST_MinimumRotatedRectangle . . . 568
ST_MMax . . . 568
ST_MMin . . . 569
ST_Multi . . . 569
ST_NGeometries . . . 570
ST_NInteriorRings . . . 571
ST_Node . . . 571
ST_Normalize . . . 572

ST_NPoints . . . 572
ST_NumGeometries . . . 573
ST_NumInteriorRings . . . 574
ST_NumPoints . . . 574
ST_Overlaps . . . 575
ST_Perimeter . . . 576
ST_Perimeter_Spheroid . . . 576
ST_Point . . . 577
ST_Point2D . . . 578
ST_Point2DFromWKB . . . 578
ST_Point3D . . . 579
ST_Point4D . . . 579
ST_PointN . . . 580
ST_PointOnSurface . . . 580
ST_Points . . . 581
ST_Polygon2DFromWKB . . . 581
ST_Polygonize . . . 582
ST_QuadKey . . . 583
ST_Read . . . 584
ST_Read_Meta . . . 586
ST_ReadOSM . . . 586
ST_ReadSHP . . . 588
ST_ReducePrecision . . . 588
ST_RemoveRepeatedPoints . . . 589
ST_Reverse . . . 590
ST_Rotate . . . 590
ST_RotateX . . . 591
ST_RotateY . . . 591
ST_RotateZ . . . 592
ST_Scale . . . 593
st_setcrs . . . 593
ST_ShortestLine . . . 594
ST_Simplify . . . 594
ST_SimplifyPreserveTopology . . . 595
ST_Snap . . . 595
ST_StartPoint . . . 596
ST_SymDifference . . . 596
ST_TileEnvelope . . . 597
ST_Touches . . . 598
ST_Transform . . . 598
ST_Translate . . . 600
ST_TransScale . . . 601
ST_Union . . . 602
ST_Union_Agg . . . 602
ST_VoronoiDiagram . . . 603
ST_Within . . . 603
ST_WithinProperly . . . 604
ST_X . . . 604

ST_XMax . . . 605
ST_XMin . . . 606
ST_Y . . . 607
ST_YMax . . . 607
ST_YMin . . . 608
ST_Z . . . 609
ST_ZMax . . . 609
ST_ZMFlag . . . 610
ST_ZMin . . . 611
start_ui . . . 611
start_ui_server . . . 612
starts_with . . . 612
stats . . . 613
stddev_pop . . . 614
stddev_samp . . . 614
stem . . . 615
stop_ui_server . . . 615
storage_info . . . 616
strftime . . . 616
string_agg . . . 617
string_split . . . 617
string_split_regex . . . 618
strip_accents . . . 619
strlen . . . 620
strptime . . . 620
struct_concat . . . 621
struct_contains . . . 622
struct_extract . . . 622
struct_extract_at . . . 623
struct_insert . . . 623
struct_keys . . . 624
struct_pack . . . 624
struct_position . . . 625
struct_update . . . 625
struct_values . . . 626
substring . . . 626
substring_grapheme . . . 627
subtract . . . 628
suffix . . . 630
sum . . . 630
sum_no_overflow . . . 631
summary . . . 632
switch . . . 632
table_info . . . 633
tan . . . 633
tanh . . . 634
test_all_types . . . 634
test_vector_types . . . 635

text . . . 635
time_bucket . . . 636
timetz_byte_comparable . . . 637
timezone . . . 637
timezone_hour . . . 638
timezone_minute . . . 639
to_base . . . 639
to_base64 . . . 640
to_centuries . . . 641
to_days . . . 642
to_decades . . . 642
to_hours . . . 643
to_json . . . 643
to_microseconds . . . 644
to_millennia . . . 644
to_milliseconds . . . 645
to_minutes . . . 645
to_months . . . 646
to_quarters . . . 646
to_seconds . . . 647
to_timestamp . . . 647
to_weeks . . . 648
to_years . . . 648
today . . . 649
tpcds . . . 649
tpcds_answers . . . 650
tpcds_queries . . . 650
tpch . . . 651
tpch_answers . . . 651
tpch_queries . . . 652
translate . . . 652
trim . . . 653
trunc . . . 654
truncate_duckdb_logs . . . 655
try_strptime . . . 656
txid_current . . . 656
typeof . . . 657
ui_is_started . . . 657
unbin . . . 658
unhex . . . 659
unicode . . . 660
union_extract . . . 660
union_tag . . . 661
union_value . . . 662
unity_catalog_checkpoint_table . . . 662
unity_catalog_force_checkpoint_table . . . 663
unnest . . . 663
unpivot_list . . . 664

upper . . . 664
url_decode . . . 665
url_encode . . . 666
user . . . 667
user_agent . . . 667
uuid . . . 667
uuid_extract_timestamp . . . 668
uuid_extract_version . . . 668
uuidv4 . . . 669
uuidv7 . . . 669
var_pop . . . 670
var_samp . . . 670
variant_bytes_to_variant . . . 671
variant_extract . . . 671
variant_normalize . . . 672
variant_to_parquet_variant . . . 673
variant_typeof . . . 673
vector_type . . . 674
verify_external . . . 674
verify_fetch_row . . . 675
verify_parallelism . . . 675
verify_serializer . . . 675
version . . . 676
vortex_scan . . . 676
vss_join . . . 677
vss_match . . . 677
wavg . . . 678
week . . . 678
weekday . . . 679
weekofyear . . . 679
weighted_avg . . . 680
which_secret . . . 680
whoami . . . 681
write_blob . . . 681
write_log . . . 682
xor . . . 682
year . . . 683
yearweek . . . 683

Index **684**

/ *DuckDB function /*

Description

DuckDB function `/()`.

Usage

```
`/`(col0, col1)
```

Arguments

col0	FLOAT | DOUBLE | INTERVAL
col1	FLOAT | DOUBLE

Value

FLOAT | DOUBLE | INTERVAL

& *DuckDB function &*

Description

Bitwise AND.

Usage

```
`&`(left, right)
```

Arguments

left, right	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT

Value

TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT

SQL examples

```
91 & 15
```

`~~*` *DuckDB function ~~**

Description

DuckDB function `~~*()`.

Usage

```
`~~*`(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`BOOLEAN`

`~~~` *DuckDB function ~~~*

Description

DuckDB function `~~~()`.

Usage

```
`~~~`(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`BOOLEAN`

`~~` *DuckDB function ~~*

Description

DuckDB function `~~()`.

Usage

```
`~~`(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`BOOLEAN`

`~` *DuckDB function ~*

Description

Bitwise NOT.

Usage

```
`~`(input)
```

Arguments

`input`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

SQL examples

```
~15
```

`<<=` *DuckDB function «=*

Description

DuckDB function `<<=()`.

Usage

```
`<<=`(col0, col1)
```

Arguments

`col0`, `col1`	`INET`

Value

`BOOLEAN`

Provided by

The `inet` extension (`LOAD inet;`).

`<<` *DuckDB function «*

Description

Bitwise shift left.

Usage

```
`<<`(input, col1)
```

Arguments

`input`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`
`col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

SQL examples

```
1 << 4
```

`>>=` *DuckDB function »=*

Description

DuckDB function `>>=()`.

Usage

```
`>>=`(col0, col1)
```

Arguments

`col0`, `col1`	`INET`

Value

`BOOLEAN`

Provided by

The `inet` extension (`LOAD inet;`).

`>>` *DuckDB function »*

Description

Bitwise shift right.

Usage

```
`>>`(input, col1)
```

Arguments

`input`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`
`col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

SQL examples

```
8 >> 2
```

`abs` *DuckDB function abs*

Description

Absolute value.

Usage

```
abs(x)
```

Arguments

x	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT

Value

TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT

SQL examples

```
abs(-17.4)
```

`acos` *DuckDB function acos*

Description

Computes the arccosine of x.

Usage

```
acos(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
acos(0.5)
```

`acosh` *DuckDB function acosh*

Description

Computes the inverse hyperbolic cos of x.

Usage

```
acosh(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
acosh(2.3)
```

`add` *DuckDB function add*

Description

DuckDB function `add()`.

Usage

```
add(col0, col1)
```

Arguments

`col0`, `col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTI`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLI`

Overloads

- add(col0 = TINYINT)
- add(col0 = TINYINT, col1 = TINYINT)
- add(col0 = SMALLINT)
- add(col0 = SMALLINT, col1 = SMALLINT)
- add(col0 = INTEGER)
- add(col0 = INTEGER, col1 = INTEGER)
- add(col0 = BIGINT)
- add(col0 = BIGINT, col1 = BIGINT)
- add(col0 = HUGEINT)
- add(col0 = HUGEINT, col1 = HUGEINT)
- add(col0 = FLOAT)
- add(col0 = FLOAT, col1 = FLOAT)
- add(col0 = DOUBLE)
- add(col0 = DOUBLE, col1 = DOUBLE)
- add(col0 = DECIMAL)
- add(col0 = DECIMAL, col1 = DECIMAL)
- add(col0 = UTINYINT)
- add(col0 = UTINYINT, col1 = UTINYINT)
- add(col0 = USMALLINT)
- add(col0 = USMALLINT, col1 = USMALLINT)
- add(col0 = UINTEGER)
- add(col0 = UINTEGER, col1 = UINTEGER)
- add(col0 = UBIGINT)
- add(col0 = UBIGINT, col1 = UBIGINT)
- add(col0 = UHUGEINT)
- add(col0 = UHUGEINT, col1 = UHUGEINT)
- add(col0 = DATE, col1 = INTEGER)
- add(col0 = INTEGER, col1 = DATE)
- add(col0 = INTERVAL, col1 = INTERVAL)
- add(col0 = DATE, col1 = INTERVAL)
- add(col0 = INTERVAL, col1 = DATE)
- add(col0 = TIME, col1 = INTERVAL)
- add(col0 = INTERVAL, col1 = TIME)
- add(col0 = TIMESTAMP, col1 = INTERVAL)
- add(col0 = INTERVAL, col1 = TIMESTAMP)
- add(col0 = `TIME WITH TIME ZONE`, col1 = INTERVAL)

- add(col0 = INTERVAL, col1 = `TIME WITH TIME ZONE`)
- add(col0 = TIME, col1 = DATE)
- add(col0 = DATE, col1 = TIME)
- add(col0 = `TIME WITH TIME ZONE`, col1 = DATE)
- add(col0 = DATE, col1 = `TIME WITH TIME ZONE`)
- add()
- add(col0 = BIGNUM, col1 = BIGNUM)

`add_parquet_key` *DuckDB function add_parquet_key*

Description

DuckDB function `add_parquet_key()`.

Usage

```
add_parquet_key(col0, col1)
```

Arguments

`col0`, `col1` VARCHAR

Value

Unspecified.

`age` *DuckDB function age*

Description

Subtract arguments, resulting in the time difference between the two timestamps.

Usage

```
age(timestamp, timestamp.1)
```

Arguments

`timestamp`, `timestamp.1`
TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

INTERVAL

Overloads

- `age(timestamp = TIMESTAMP)`
- `age(timestamp = TIMESTAMP, timestamp.1 = TIMESTAMP)`
- `age(timestamp = `TIMESTAMP WITH TIME ZONE`)`
- `age(timestamp = `TIMESTAMP WITH TIME ZONE`, timestamp.1 = `TIMESTAMP WITH TIME ZONE`)`

SQL examples

```
age(TIMESTAMP '2001-04-10', TIMESTAMP '1992-09-20')
```

---

ago *DuckDB function ago*

Description

DuckDB macro `ago()`.

Usage

```
ago(i)
```

Arguments

`i`	Unspecified.

Value

Unspecified.

---

alias *DuckDB function alias*

Description

Returns the name of a given expression.

Usage

```
alias(expr)
```

Arguments

`expr`	`ANY`

Value

VARCHAR

SQL examples

```
alias(42 + 1)
```

---

all_profiling_output *DuckDB function all_profiling_output*

Description

DuckDB function all_profiling_output().

Usage

```
all_profiling_output()
```

Value

Unspecified.

---

any_value *DuckDB function any_value*

Description

Returns the first non-NULL value from arg. This function is affected by ordering.

Usage

```
any_value(arg)
```

Arguments

arg	DECIMAL | ANY

Value

DECIMAL | ANY

`approx_count_distinct` *DuckDB function approx_count_distinct*

Description

Computes the approximate count of distinct elements using HyperLogLog.

Usage

```
approx_count_distinct(any)
```

Arguments

any	ANY

Value

```
BIGINT
```

SQL examples

```
approx_count_distinct(A)
```

`approx_quantile` *DuckDB function approx_quantile*

Description

Computes the approximate quantile using T-Digest.

Usage

```
approx_quantile(x, pos)
```

Arguments

x	DECIMAL | SMALLINT | INTEGER | BIGINT | HUGEINT | DOUBLE | DATE | TIME | TIME WI
pos	FLOAT | FLOAT[]

Value

```
DECIMAL | SMALLINT | INTEGER | BIGINT | HUGEINT | DOUBLE | DATE | TIME | TIME WITH TIME ZONE |
```

SQL examples

```
approx_quantile(x, 0.5)
```

`approx_top_k` *DuckDB function approx_top_k*

Description

Finds the k approximately most occurring values in the data set.

Usage

```
approx_top_k(val, k)
```

Arguments

val	ANY
k	BIGINT

Value

```
ANY[]
```

SQL examples

```
approx_top_k(x, 5)
```

`arg_max` *DuckDB function arg_max*

Description

Finds the row with the maximum val. Calculates the non-NULL arg expression at that row.

Usage

```
arg_max(arg, val)
```

Arguments

arg	INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZON
val	INTEGER | BIGINT | HUGEINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WIT

Value

```
INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE | BLOB | DEC
```

Overloads

- arg_max(arg = INTEGER, val = INTEGER)
- arg_max(arg = INTEGER, val = BIGINT)
- arg_max(arg = INTEGER, val = HUGEINT)
- arg_max(arg = INTEGER, val = DOUBLE)
- arg_max(arg = INTEGER, val = VARCHAR)
- arg_max(arg = INTEGER, val = DATE)
- arg_max(arg = INTEGER, val = TIMESTAMP)
- arg_max(arg = INTEGER, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = INTEGER, val = BLOB)
- arg_max(arg = BIGINT, val = INTEGER)
- arg_max(arg = BIGINT, val = BIGINT)
- arg_max(arg = BIGINT, val = HUGEINT)
- arg_max(arg = BIGINT, val = DOUBLE)
- arg_max(arg = BIGINT, val = VARCHAR)
- arg_max(arg = BIGINT, val = DATE)
- arg_max(arg = BIGINT, val = TIMESTAMP)
- arg_max(arg = BIGINT, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = BIGINT, val = BLOB)
- arg_max(arg = DOUBLE, val = INTEGER)
- arg_max(arg = DOUBLE, val = BIGINT)
- arg_max(arg = DOUBLE, val = HUGEINT)
- arg_max(arg = DOUBLE, val = DOUBLE)
- arg_max(arg = DOUBLE, val = VARCHAR)
- arg_max(arg = DOUBLE, val = DATE)
- arg_max(arg = DOUBLE, val = TIMESTAMP)
- arg_max(arg = DOUBLE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = DOUBLE, val = BLOB)
- arg_max(arg = VARCHAR, val = INTEGER)
- arg_max(arg = VARCHAR, val = BIGINT)
- arg_max(arg = VARCHAR, val = HUGEINT)
- arg_max(arg = VARCHAR, val = DOUBLE)
- arg_max(arg = VARCHAR, val = VARCHAR)
- arg_max(arg = VARCHAR, val = DATE)
- arg_max(arg = VARCHAR, val = TIMESTAMP)
- arg_max(arg = VARCHAR, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = VARCHAR, val = BLOB)

- arg_max(arg = DATE, val = INTEGER)
- arg_max(arg = DATE, val = BIGINT)
- arg_max(arg = DATE, val = HUGEINT)
- arg_max(arg = DATE, val = DOUBLE)
- arg_max(arg = DATE, val = VARCHAR)
- arg_max(arg = DATE, val = DATE)
- arg_max(arg = DATE, val = TIMESTAMP)
- arg_max(arg = DATE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = DATE, val = BLOB)
- arg_max(arg = TIMESTAMP, val = INTEGER)
- arg_max(arg = TIMESTAMP, val = BIGINT)
- arg_max(arg = TIMESTAMP, val = HUGEINT)
- arg_max(arg = TIMESTAMP, val = DOUBLE)
- arg_max(arg = TIMESTAMP, val = VARCHAR)
- arg_max(arg = TIMESTAMP, val = DATE)
- arg_max(arg = TIMESTAMP, val = TIMESTAMP)
- arg_max(arg = TIMESTAMP, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = TIMESTAMP, val = BLOB)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = INTEGER)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = BIGINT)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = HUGEINT)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = DOUBLE)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = VARCHAR)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = DATE)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = TIMESTAMP)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = `TIMESTAMP WITH TIME ZONE`, val = BLOB)
- arg_max(arg = BLOB, val = INTEGER)
- arg_max(arg = BLOB, val = BIGINT)
- arg_max(arg = BLOB, val = HUGEINT)
- arg_max(arg = BLOB, val = DOUBLE)
- arg_max(arg = BLOB, val = VARCHAR)
- arg_max(arg = BLOB, val = DATE)
- arg_max(arg = BLOB, val = TIMESTAMP)
- arg_max(arg = BLOB, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = BLOB, val = BLOB)
- arg_max(arg = DECIMAL, val = INTEGER)

- arg_max(arg = DECIMAL, val = BIGINT)
- arg_max(arg = DECIMAL, val = HUGEINT)
- arg_max(arg = DECIMAL, val = DOUBLE)
- arg_max(arg = DECIMAL, val = VARCHAR)
- arg_max(arg = DECIMAL, val = DATE)
- arg_max(arg = DECIMAL, val = TIMESTAMP)
- arg_max(arg = DECIMAL, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = DECIMAL, val = BLOB)
- arg_max(arg = ANY, val = INTEGER)
- arg_max(arg = ANY, val = BIGINT)
- arg_max(arg = ANY, val = HUGEINT)
- arg_max(arg = ANY, val = DOUBLE)
- arg_max(arg = ANY, val = VARCHAR)
- arg_max(arg = ANY, val = DATE)
- arg_max(arg = ANY, val = TIMESTAMP)
- arg_max(arg = ANY, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max(arg = ANY, val = BLOB)
- arg_max(arg = ANY, val = ANY)
- arg_max(arg = ANY, val = ANY, col2 = BIGINT)

SQL examples

```
arg_max(A, B)
```

---

arg_max_null *DuckDB function arg_max_null*

Description

Finds the row with the maximum val. Calculates the arg expression at that row.

Usage

```
arg_max_null(arg, val)
```

Arguments

arg	INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZON
val	INTEGER | BIGINT | HUGEINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WI

Value

INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE | BLOB | DEC

SQL examples

```
arg_max_null(A, B)
```

`arg_max_nulls_last` *DuckDB function arg_max_nulls_last*

Description

Finds the rows with N maximum vals, including nulls. Calculates the arg expression at that row.

Usage

```
arg_max_nulls_last(arg, val)
```

Arguments

`arg`	`INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZON`
`val`	`INTEGER | BIGINT | HUGEINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WI`

Value

`INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE | BLOB | DEC`

Overloads

- `arg_max_nulls_last(arg = INTEGER, val = INTEGER)`
- `arg_max_nulls_last(arg = INTEGER, val = BIGINT)`
- `arg_max_nulls_last(arg = INTEGER, val = HUGEINT)`
- `arg_max_nulls_last(arg = INTEGER, val = DOUBLE)`
- `arg_max_nulls_last(arg = INTEGER, val = VARCHAR)`
- `arg_max_nulls_last(arg = INTEGER, val = DATE)`
- `arg_max_nulls_last(arg = INTEGER, val = TIMESTAMP)`
- `arg_max_nulls_last(arg = INTEGER, val = `TIMESTAMP WITH TIME ZONE`)`
- `arg_max_nulls_last(arg = INTEGER, val = BLOB)`
- `arg_max_nulls_last(arg = BIGINT, val = INTEGER)`
- `arg_max_nulls_last(arg = BIGINT, val = BIGINT)`
- `arg_max_nulls_last(arg = BIGINT, val = HUGEINT)`
- `arg_max_nulls_last(arg = BIGINT, val = DOUBLE)`
- `arg_max_nulls_last(arg = BIGINT, val = VARCHAR)`
- `arg_max_nulls_last(arg = BIGINT, val = DATE)`
- `arg_max_nulls_last(arg = BIGINT, val = TIMESTAMP)`
- `arg_max_nulls_last(arg = BIGINT, val = `TIMESTAMP WITH TIME ZONE`)`

- arg_max_nulls_last(arg = BIGINT, val = BLOB)
- arg_max_nulls_last(arg = DOUBLE, val = INTEGER)
- arg_max_nulls_last(arg = DOUBLE, val = BIGINT)
- arg_max_nulls_last(arg = DOUBLE, val = HUGEINT)
- arg_max_nulls_last(arg = DOUBLE, val = DOUBLE)
- arg_max_nulls_last(arg = DOUBLE, val = VARCHAR)
- arg_max_nulls_last(arg = DOUBLE, val = DATE)
- arg_max_nulls_last(arg = DOUBLE, val = TIMESTAMP)
- arg_max_nulls_last(arg = DOUBLE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = DOUBLE, val = BLOB)
- arg_max_nulls_last(arg = VARCHAR, val = INTEGER)
- arg_max_nulls_last(arg = VARCHAR, val = BIGINT)
- arg_max_nulls_last(arg = VARCHAR, val = HUGEINT)
- arg_max_nulls_last(arg = VARCHAR, val = DOUBLE)
- arg_max_nulls_last(arg = VARCHAR, val = VARCHAR)
- arg_max_nulls_last(arg = VARCHAR, val = DATE)
- arg_max_nulls_last(arg = VARCHAR, val = TIMESTAMP)
- arg_max_nulls_last(arg = VARCHAR, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = VARCHAR, val = BLOB)
- arg_max_nulls_last(arg = DATE, val = INTEGER)
- arg_max_nulls_last(arg = DATE, val = BIGINT)
- arg_max_nulls_last(arg = DATE, val = HUGEINT)
- arg_max_nulls_last(arg = DATE, val = DOUBLE)
- arg_max_nulls_last(arg = DATE, val = VARCHAR)
- arg_max_nulls_last(arg = DATE, val = DATE)
- arg_max_nulls_last(arg = DATE, val = TIMESTAMP)
- arg_max_nulls_last(arg = DATE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = DATE, val = BLOB)
- arg_max_nulls_last(arg = TIMESTAMP, val = INTEGER)
- arg_max_nulls_last(arg = TIMESTAMP, val = BIGINT)
- arg_max_nulls_last(arg = TIMESTAMP, val = HUGEINT)
- arg_max_nulls_last(arg = TIMESTAMP, val = DOUBLE)
- arg_max_nulls_last(arg = TIMESTAMP, val = VARCHAR)
- arg_max_nulls_last(arg = TIMESTAMP, val = DATE)
- arg_max_nulls_last(arg = TIMESTAMP, val = TIMESTAMP)
- arg_max_nulls_last(arg = TIMESTAMP, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = TIMESTAMP, val = BLOB)

- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = INTEGER)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = BIGINT)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = HUGEINT)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = DOUBLE)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = VARCHAR)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = DATE)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = TIMESTAMP)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = BLOB)
- arg_max_nulls_last(arg = BLOB, val = INTEGER)
- arg_max_nulls_last(arg = BLOB, val = BIGINT)
- arg_max_nulls_last(arg = BLOB, val = HUGEINT)
- arg_max_nulls_last(arg = BLOB, val = DOUBLE)
- arg_max_nulls_last(arg = BLOB, val = VARCHAR)
- arg_max_nulls_last(arg = BLOB, val = DATE)
- arg_max_nulls_last(arg = BLOB, val = TIMESTAMP)
- arg_max_nulls_last(arg = BLOB, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = BLOB, val = BLOB)
- arg_max_nulls_last(arg = DECIMAL, val = INTEGER)
- arg_max_nulls_last(arg = DECIMAL, val = BIGINT)
- arg_max_nulls_last(arg = DECIMAL, val = HUGEINT)
- arg_max_nulls_last(arg = DECIMAL, val = DOUBLE)
- arg_max_nulls_last(arg = DECIMAL, val = VARCHAR)
- arg_max_nulls_last(arg = DECIMAL, val = DATE)
- arg_max_nulls_last(arg = DECIMAL, val = TIMESTAMP)
- arg_max_nulls_last(arg = DECIMAL, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = DECIMAL, val = BLOB)
- arg_max_nulls_last(arg = ANY, val = INTEGER)
- arg_max_nulls_last(arg = ANY, val = BIGINT)
- arg_max_nulls_last(arg = ANY, val = HUGEINT)
- arg_max_nulls_last(arg = ANY, val = DOUBLE)
- arg_max_nulls_last(arg = ANY, val = VARCHAR)
- arg_max_nulls_last(arg = ANY, val = DATE)
- arg_max_nulls_last(arg = ANY, val = TIMESTAMP)
- arg_max_nulls_last(arg = ANY, val = `TIMESTAMP WITH TIME ZONE`)
- arg_max_nulls_last(arg = ANY, val = BLOB)
- arg_max_nulls_last(arg = ANY, val = ANY)
- arg_max_nulls_last(arg = ANY, val = ANY, N = BIGINT)

SQL examples

```
arg_min_null_val(A, B, N)
```

`arg_min` *DuckDB function arg_min*

Description

Finds the row with the minimum val. Calculates the non-NULL arg expression at that row.

Usage

```
arg_min(arg, val)
```

Arguments

arg	INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZON
val	INTEGER | BIGINT | HUGEINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WIT

Value

INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE | BLOB | DEC

Overloads

- arg_min(arg = INTEGER, val = INTEGER)
- arg_min(arg = INTEGER, val = BIGINT)
- arg_min(arg = INTEGER, val = HUGEINT)
- arg_min(arg = INTEGER, val = DOUBLE)
- arg_min(arg = INTEGER, val = VARCHAR)
- arg_min(arg = INTEGER, val = DATE)
- arg_min(arg = INTEGER, val = TIMESTAMP)
- arg_min(arg = INTEGER, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = INTEGER, val = BLOB)
- arg_min(arg = BIGINT, val = INTEGER)
- arg_min(arg = BIGINT, val = BIGINT)
- arg_min(arg = BIGINT, val = HUGEINT)
- arg_min(arg = BIGINT, val = DOUBLE)
- arg_min(arg = BIGINT, val = VARCHAR)
- arg_min(arg = BIGINT, val = DATE)
- arg_min(arg = BIGINT, val = TIMESTAMP)
- arg_min(arg = BIGINT, val = `TIMESTAMP WITH TIME ZONE`)

- arg_min(arg = BIGINT, val = BLOB)
- arg_min(arg = DOUBLE, val = INTEGER)
- arg_min(arg = DOUBLE, val = BIGINT)
- arg_min(arg = DOUBLE, val = HUGEINT)
- arg_min(arg = DOUBLE, val = DOUBLE)
- arg_min(arg = DOUBLE, val = VARCHAR)
- arg_min(arg = DOUBLE, val = DATE)
- arg_min(arg = DOUBLE, val = TIMESTAMP)
- arg_min(arg = DOUBLE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = DOUBLE, val = BLOB)
- arg_min(arg = VARCHAR, val = INTEGER)
- arg_min(arg = VARCHAR, val = BIGINT)
- arg_min(arg = VARCHAR, val = HUGEINT)
- arg_min(arg = VARCHAR, val = DOUBLE)
- arg_min(arg = VARCHAR, val = VARCHAR)
- arg_min(arg = VARCHAR, val = DATE)
- arg_min(arg = VARCHAR, val = TIMESTAMP)
- arg_min(arg = VARCHAR, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = VARCHAR, val = BLOB)
- arg_min(arg = DATE, val = INTEGER)
- arg_min(arg = DATE, val = BIGINT)
- arg_min(arg = DATE, val = HUGEINT)
- arg_min(arg = DATE, val = DOUBLE)
- arg_min(arg = DATE, val = VARCHAR)
- arg_min(arg = DATE, val = DATE)
- arg_min(arg = DATE, val = TIMESTAMP)
- arg_min(arg = DATE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = DATE, val = BLOB)
- arg_min(arg = TIMESTAMP, val = INTEGER)
- arg_min(arg = TIMESTAMP, val = BIGINT)
- arg_min(arg = TIMESTAMP, val = HUGEINT)
- arg_min(arg = TIMESTAMP, val = DOUBLE)
- arg_min(arg = TIMESTAMP, val = VARCHAR)
- arg_min(arg = TIMESTAMP, val = DATE)
- arg_min(arg = TIMESTAMP, val = TIMESTAMP)
- arg_min(arg = TIMESTAMP, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = TIMESTAMP, val = BLOB)

- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = INTEGER)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = BIGINT)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = HUGEINT)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = DOUBLE)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = VARCHAR)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = DATE)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = TIMESTAMP)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = `TIMESTAMP WITH TIME ZONE`, val = BLOB)
- arg_min(arg = BLOB, val = INTEGER)
- arg_min(arg = BLOB, val = BIGINT)
- arg_min(arg = BLOB, val = HUGEINT)
- arg_min(arg = BLOB, val = DOUBLE)
- arg_min(arg = BLOB, val = VARCHAR)
- arg_min(arg = BLOB, val = DATE)
- arg_min(arg = BLOB, val = TIMESTAMP)
- arg_min(arg = BLOB, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = BLOB, val = BLOB)
- arg_min(arg = DECIMAL, val = INTEGER)
- arg_min(arg = DECIMAL, val = BIGINT)
- arg_min(arg = DECIMAL, val = HUGEINT)
- arg_min(arg = DECIMAL, val = DOUBLE)
- arg_min(arg = DECIMAL, val = VARCHAR)
- arg_min(arg = DECIMAL, val = DATE)
- arg_min(arg = DECIMAL, val = TIMESTAMP)
- arg_min(arg = DECIMAL, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = DECIMAL, val = BLOB)
- arg_min(arg = ANY, val = INTEGER)
- arg_min(arg = ANY, val = BIGINT)
- arg_min(arg = ANY, val = HUGEINT)
- arg_min(arg = ANY, val = DOUBLE)
- arg_min(arg = ANY, val = VARCHAR)
- arg_min(arg = ANY, val = DATE)
- arg_min(arg = ANY, val = TIMESTAMP)
- arg_min(arg = ANY, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min(arg = ANY, val = BLOB)
- arg_min(arg = ANY, val = ANY)
- arg_min(arg = ANY, val = ANY, col2 = BIGINT)

SQL examples

```
arg_min(A, B)
```

`arg_min_null` *DuckDB function arg_min_null*

Description

Finds the row with the minimum val. Calculates the arg expression at that row.

Usage

```
arg_min_null(arg, val)
```

Arguments

arg	INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZON
val	INTEGER | BIGINT | HUGEINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WIT

Value

INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE | BLOB | DEC

SQL examples

```
arg_min_null(A, B)
```

`arg_min_nulls_last` *DuckDB function arg_min_nulls_last*

Description

Finds the rows with N minimum vals, including nulls. Calculates the arg expression at that row.

Usage

```
arg_min_nulls_last(arg, val)
```

Arguments

arg	INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZON
val	INTEGER | BIGINT | HUGEINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WIT

Value

INTEGER | BIGINT | DOUBLE | VARCHAR | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE | BLOB | DEC

Overloads

- arg_min_nulls_last(arg = INTEGER, val = INTEGER)
- arg_min_nulls_last(arg = INTEGER, val = BIGINT)
- arg_min_nulls_last(arg = INTEGER, val = HUGEINT)
- arg_min_nulls_last(arg = INTEGER, val = DOUBLE)
- arg_min_nulls_last(arg = INTEGER, val = VARCHAR)
- arg_min_nulls_last(arg = INTEGER, val = DATE)
- arg_min_nulls_last(arg = INTEGER, val = TIMESTAMP)
- arg_min_nulls_last(arg = INTEGER, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = INTEGER, val = BLOB)
- arg_min_nulls_last(arg = BIGINT, val = INTEGER)
- arg_min_nulls_last(arg = BIGINT, val = BIGINT)
- arg_min_nulls_last(arg = BIGINT, val = HUGEINT)
- arg_min_nulls_last(arg = BIGINT, val = DOUBLE)
- arg_min_nulls_last(arg = BIGINT, val = VARCHAR)
- arg_min_nulls_last(arg = BIGINT, val = DATE)
- arg_min_nulls_last(arg = BIGINT, val = TIMESTAMP)
- arg_min_nulls_last(arg = BIGINT, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = BIGINT, val = BLOB)
- arg_min_nulls_last(arg = DOUBLE, val = INTEGER)
- arg_min_nulls_last(arg = DOUBLE, val = BIGINT)
- arg_min_nulls_last(arg = DOUBLE, val = HUGEINT)
- arg_min_nulls_last(arg = DOUBLE, val = DOUBLE)
- arg_min_nulls_last(arg = DOUBLE, val = VARCHAR)
- arg_min_nulls_last(arg = DOUBLE, val = DATE)
- arg_min_nulls_last(arg = DOUBLE, val = TIMESTAMP)
- arg_min_nulls_last(arg = DOUBLE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = DOUBLE, val = BLOB)
- arg_min_nulls_last(arg = VARCHAR, val = INTEGER)
- arg_min_nulls_last(arg = VARCHAR, val = BIGINT)
- arg_min_nulls_last(arg = VARCHAR, val = HUGEINT)
- arg_min_nulls_last(arg = VARCHAR, val = DOUBLE)
- arg_min_nulls_last(arg = VARCHAR, val = VARCHAR)
- arg_min_nulls_last(arg = VARCHAR, val = DATE)
- arg_min_nulls_last(arg = VARCHAR, val = TIMESTAMP)
- arg_min_nulls_last(arg = VARCHAR, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = VARCHAR, val = BLOB)

- arg_min_nulls_last(arg = DATE, val = INTEGER)
- arg_min_nulls_last(arg = DATE, val = BIGINT)
- arg_min_nulls_last(arg = DATE, val = HUGEINT)
- arg_min_nulls_last(arg = DATE, val = DOUBLE)
- arg_min_nulls_last(arg = DATE, val = VARCHAR)
- arg_min_nulls_last(arg = DATE, val = DATE)
- arg_min_nulls_last(arg = DATE, val = TIMESTAMP)
- arg_min_nulls_last(arg = DATE, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = DATE, val = BLOB)
- arg_min_nulls_last(arg = TIMESTAMP, val = INTEGER)
- arg_min_nulls_last(arg = TIMESTAMP, val = BIGINT)
- arg_min_nulls_last(arg = TIMESTAMP, val = HUGEINT)
- arg_min_nulls_last(arg = TIMESTAMP, val = DOUBLE)
- arg_min_nulls_last(arg = TIMESTAMP, val = VARCHAR)
- arg_min_nulls_last(arg = TIMESTAMP, val = DATE)
- arg_min_nulls_last(arg = TIMESTAMP, val = TIMESTAMP)
- arg_min_nulls_last(arg = TIMESTAMP, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = TIMESTAMP, val = BLOB)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = INTEGER)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = BIGINT)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = HUGEINT)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = DOUBLE)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = VARCHAR)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = DATE)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = TIMESTAMP)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = `TIMESTAMP WITH TIME ZONE`, val = BLOB)
- arg_min_nulls_last(arg = BLOB, val = INTEGER)
- arg_min_nulls_last(arg = BLOB, val = BIGINT)
- arg_min_nulls_last(arg = BLOB, val = HUGEINT)
- arg_min_nulls_last(arg = BLOB, val = DOUBLE)
- arg_min_nulls_last(arg = BLOB, val = VARCHAR)
- arg_min_nulls_last(arg = BLOB, val = DATE)
- arg_min_nulls_last(arg = BLOB, val = TIMESTAMP)
- arg_min_nulls_last(arg = BLOB, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = BLOB, val = BLOB)

- arg_min_nulls_last(arg = DECIMAL, val = INTEGER)
- arg_min_nulls_last(arg = DECIMAL, val = BIGINT)
- arg_min_nulls_last(arg = DECIMAL, val = HUGEINT)
- arg_min_nulls_last(arg = DECIMAL, val = DOUBLE)
- arg_min_nulls_last(arg = DECIMAL, val = VARCHAR)
- arg_min_nulls_last(arg = DECIMAL, val = DATE)
- arg_min_nulls_last(arg = DECIMAL, val = TIMESTAMP)
- arg_min_nulls_last(arg = DECIMAL, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = DECIMAL, val = BLOB)
- arg_min_nulls_last(arg = ANY, val = INTEGER)
- arg_min_nulls_last(arg = ANY, val = BIGINT)
- arg_min_nulls_last(arg = ANY, val = HUGEINT)
- arg_min_nulls_last(arg = ANY, val = DOUBLE)
- arg_min_nulls_last(arg = ANY, val = VARCHAR)
- arg_min_nulls_last(arg = ANY, val = DATE)
- arg_min_nulls_last(arg = ANY, val = TIMESTAMP)
- arg_min_nulls_last(arg = ANY, val = `TIMESTAMP WITH TIME ZONE`)
- arg_min_nulls_last(arg = ANY, val = BLOB)
- arg_min_nulls_last(arg = ANY, val = ANY)
- arg_min_nulls_last(arg = ANY, val = ANY, N = BIGINT)

SQL examples

```
arg_min_null_val(A, B, N)
```

---

array_agg *DuckDB function array_agg*

Description

Returns a LIST containing all the values of a column.

Usage

```
array_agg(arg)
```

Arguments

arg	T

Value

T[]

SQL examples

```
array_agg(A)
list(A)
```

`array_append` *DuckDB function array_append*

Description

DuckDB macro `array_append()`.

Usage

```
array_append(arr, el)
```

Arguments

`arr`, `el`	Unspecified.

Value

Unspecified.

`array_cosine_distance` *DuckDB function array_cosine_distance*

Description

Computes the cosine distance between two arrays of the same size. The array elements can not be `NULL`. The arrays can have any size as long as the size is the same for both arguments.

Usage

```
array_cosine_distance(array1, array2)
```

Arguments

`array1`, `array2`	`FLOAT[ANY] | DOUBLE[ANY]`

Value

`FLOAT | DOUBLE`

SQL examples

```
array_cosine_distance(array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT), array_value(2.0::FLOAT,
```

See Also

Other array: `array_cosine_similarity()`, `array_cross_product()`, `array_distance()`, `array_inner_product()`, `array_negative_inner_product()`, `array_value()`

---

array_cosine_similarity

DuckDB function array_cosine_similarity

Description

Computes the cosine similarity between two arrays of the same size. The array elements can not be `NULL`. The arrays can have any size as long as the size is the same for both arguments.

Usage

```
array_cosine_similarity(array1, array2)
```

Arguments

`array1`, `array2` `FLOAT[ANY] | DOUBLE[ANY]`

Value

`FLOAT | DOUBLE`

SQL examples

```
array_cosine_similarity(array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT), array_value(2.0::FLOA
```

See Also

Other array: `array_cosine_distance()`, `array_cross_product()`, `array_distance()`, `array_inner_product()`, `array_negative_inner_product()`, `array_value()`

`array_cross_product` *DuckDB function array_cross_product*

Description

Computes the cross product of two arrays of size 3. The array elements can not be `NULL`.

Usage

```
array_cross_product(array, array.1)
```

Arguments

`array`, `array.1` `FLOAT[3] | DOUBLE[3]`

Value

`FLOAT[3] | DOUBLE[3]`

SQL examples

```
array_cross_product(array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT), array_value(2.0::FLOAT, 3
```

See Also

Other array: `array_cosine_distance()`, `array_cosine_similarity()`, `array_distance()`, `array_inner_product()`, `array_negative_inner_product()`, `array_value()`

`array_distance` *DuckDB function array_distance*

Description

Computes the distance between two arrays of the same size. The array elements can not be `NULL`. The arrays can have any size as long as the size is the same for both arguments.

Usage

```
array_distance(array1, array2)
```

Arguments

`array1`, `array2` `FLOAT[ANY] | DOUBLE[ANY]`

Value

`FLOAT | DOUBLE`

SQL examples

```
array_distance(array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT), array_value(2.0::FLOAT, 3.0::FL
```

See Also

Other array: `array_cosine_distance()`, `array_cosine_similarity()`, `array_cross_product()`, `array_inner_product()`, `array_negative_inner_product()`, `array_value()`

`array_extract` *DuckDB function array_extract*

Description

- `array_extract(list = T[], index = BIGINT)`: Extracts the `index`th (1-based) value from the `list`.
- `array_extract(string = VARCHAR, index = BIGINT)`: Extracts a single character from a `string` using a (1-based) `index`.
- `array_extract(struct = STRUCT, entry = VARCHAR)`: Extracts the named `entry` from the `STRUCT`.
- `array_extract(struct = STRUCT, index = BIGINT)`: Extracts the entry from an unnamed `STRUCT` (tuple) using an index (1-based).

Usage

```
array_extract(list, index)
```

Arguments

`list`	`T[]`
`index`	`BIGINT`

Value

`T | VARCHAR | ANY`

Overloads

- `array_extract(list = T[], index = BIGINT)`
- `array_extract(string = VARCHAR, index = BIGINT)`
- `array_extract(struct = STRUCT, entry = VARCHAR)`
- `array_extract(struct = STRUCT, index = BIGINT)`

SQL examples

```
array_extract('DuckDB', 2)
array_extract({'i': 3, 'v2': 3, 'v3': 0}, 'i')
array_extract(row(42, 84), 1)
array_extract([4, 5, 6], 3)
```

See Also

Other list: `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

Other string: `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`array_inner_product` *DuckDB function array_inner_product*

Description

Computes the inner product between two arrays of the same size. The array elements can not be `NULL`. The arrays can have any size as long as the size is the same for both arguments.

Usage

```
array_inner_product(array1, array2)
```

Arguments

`array1`, `array2` `FLOAT[ANY] | DOUBLE[ANY]`

Value

`FLOAT | DOUBLE`

SQL examples

```
array_inner_product(array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT), array_value(2.0::FLOAT, 3
```

See Also

Other array: array_cosine_distance(), array_cosine_similarity(), array_cross_product(), array_distance(), array_negative_inner_product(), array_value()

---

array_length *DuckDB function array_length*

Description

- array_length(list = `ANY[]`): Returns the length of the list.
- array_length(list = `ANY[]`, dimension = BIGINT): array_length for lists with dimensions other than 1 not implemented.

Usage

```
array_length(list, dimension)
```

Arguments

list	ANY[]
dimension	BIGINT

Value

BIGINT

Overloads

- array_length(list = `ANY[]`)
- array_length(list = `ANY[]`, dimension = BIGINT)

SQL examples

```
array_length([1, 2, 3])
```

See Also

Other list: array_extract(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

`array_negative_inner_product`
DuckDB function array_negative_inner_product

Description

Computes the negative inner product between two arrays of the same size. The array elements can not be `NULL`. The arrays can have any size as long as the size is the same for both arguments.

Usage

```
array_negative_inner_product(array1, array2)
```

Arguments

`array1`, `array2` `FLOAT[ANY] | DOUBLE[ANY]`

Value

`FLOAT | DOUBLE`

SQL examples

```
array_negative_inner_product(array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT), array_value(2.0:
```

See Also

Other array: `array_cosine_distance()`, `array_cosine_similarity()`, `array_cross_product()`, `array_distance()`, `array_inner_product()`, `array_value()`

`array_pop_back` *DuckDB function array_pop_back*

Description

DuckDB macro `array_pop_back()`.

Usage

```
array_pop_back(arr)
```

Arguments

`arr`	Unspecified.

Value

Unspecified.

`array_pop_front` *DuckDB function array_pop_front*

Description

DuckDB macro `array_pop_front()`.

Usage

```
array_pop_front(arr)
```

Arguments

`arr`	Unspecified.

Value

Unspecified.

`array_prepend` *DuckDB function array_prepend*

Description

DuckDB macro `array_prepend()`.

Usage

```
array_prepend(el, arr)
```

Arguments

`el`, `arr`	Unspecified.

Value

Unspecified.

`array_push_back` *DuckDB function array_push_back*

Description

DuckDB macro `array_push_back()`.

Usage

```
array_push_back(arr, e)
```

Arguments

`arr`, `e`	Unspecified.

Value

Unspecified.

`array_push_front` *DuckDB function array_push_front*

Description

DuckDB macro `array_push_front()`.

Usage

```
array_push_front(arr, e)
```

Arguments

`arr`, `e`	Unspecified.

Value

Unspecified.

array_reverse *DuckDB function array_reverse*

Description

DuckDB macro `array_reverse()`.

Usage

```
array_reverse(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

array_to_json *DuckDB function array_to_json*

Description

DuckDB function `array_to_json()`.

Usage

```
array_to_json()
```

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

`array_to_string` *DuckDB function array_to_string*

Description

DuckDB macro `array_to_string()`.

Usage

```
array_to_string(arr, sep)
```

Arguments

`arr`, `sep`	Unspecified.

Value

Unspecified.

`array_to_string_comma_default`
DuckDB function array_to_string_comma_default

Description

DuckDB macro `array_to_string_comma_default()`.

Usage

```
array_to_string_comma_default(arr, sep)
```

Arguments

`arr`, `sep`	Unspecified.

Value

Unspecified.

`array_value` *DuckDB function array_value*

Description

Creates an `ARRAY` containing the argument values.

Usage

```
array_value()
```

Value

`ARRAY`

SQL examples

```
array_value(1.0::FLOAT, 2.0::FLOAT, 3.0::FLOAT)
```

See Also

Other array: `array_cosine_distance()`, `array_cosine_similarity()`, `array_cross_product()`, `array_distance()`, `array_inner_product()`, `array_negative_inner_product()`

`arrow->>` *DuckDB function ->>*

Description

DuckDB function `->>()`.

Usage

```
`->>`(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`BIGINT | VARCHAR | VARCHAR[]`

Value

`VARCHAR | VARCHAR[]`

Provided by

The `json` extension (`LOAD json;`).

arrow_scan *DuckDB function arrow_scan*

Description

DuckDB function `arrow_scan()`.

Usage

```
arrow_scan(col0, col1, col2)
```

Arguments

`col0`, `col1`, `col2`
: `POINTER`

Value

Unspecified.

arrow_scan_dumb *DuckDB function arrow_scan_dumb*

Description

DuckDB function `arrow_scan_dumb()`.

Usage

```
arrow_scan_dumb(col0, col1, col2)
```

Arguments

`col0`, `col1`, `col2`
: `POINTER`

Value

Unspecified.

`ascii` *DuckDB function ascii*

Description

Returns an integer that represents the Unicode code point of the first character of the `string`.

Usage

```
ascii(string)
```

Arguments

`string`	`VARCHAR`

Value

`INTEGER`

SQL examples

```
ascii('Ω')
```

See Also

Other string: `array_extract()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`asin` *DuckDB function asin*

Description

Computes the arcsine of x.

Usage

```
asin(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
asin(0.5)
```

---

asinh *DuckDB function asinh*

Description

Computes the inverse hyperbolic sin of x.

Usage

```
asinh(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
asinh(0.5)
```

---

atan *DuckDB function atan*

Description

Computes the arctangent of x.

Usage

```
atan(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
atan(0.5)
```

---

atan2 — *DuckDB function atan2*

Description

Computes the arctangent (y, x).

Usage

```
atan2(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
atan2(1.0, 0.0)
```

---

atanh — *DuckDB function atanh*

Description

Computes the inverse hyperbolic tan of x.

Usage

```
atanh(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
atanh(0.5)
```

---

avg *DuckDB function avg*

Description

Calculates the average value for all tuples in x.

Usage

```
avg(x)
```

Arguments

x	DECIMAL | SMALLINT | INTEGER | BIGINT | HUGEINT | INTERVAL | DOUBLE | TIMESTAMP

Value

DECIMAL | DOUBLE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE | TIME | TIME WITH TIME ZONE

SQL examples

```
SUM(x) / COUNT(*)
```

---

bar *DuckDB function bar*

Description

Draws a band whose width is proportional to (`x - min`) and equal to `width` characters when `x = max`. `width` defaults to 80.

Usage

```
bar(x, min, max, width)
```

Arguments

x, min, max, width
: DOUBLE

Value

VARCHAR

Overloads

- bar(x = DOUBLE, min = DOUBLE, max = DOUBLE, width = DOUBLE)
- bar(x = DOUBLE, min = DOUBLE, max = DOUBLE)

SQL examples

```
bar(5, 0, 20, 10)
```

See Also

Other string: array_extract(), ascii(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

bin *DuckDB function bin*

Description

- bin(string = VARCHAR): Converts the string to binary representation.
- bin(value = BIGNUM), bin(value = UBIGINT), bin(value = BIGINT), bin(value = HUGEINT), bin(value = UHUGEINT): Converts the value to binary representation.

Usage

```
bin(value)
```

Arguments

value
: BIGNUM | UBIGINT | BIGINT | HUGEINT | UHUGEINT

Value

VARCHAR

Overloads

- bin(string = VARCHAR)
- bin(value = BIGNUM)
- bin(value = UBIGINT)
- bin(value = BIGINT)
- bin(value = HUGEINT)
- bin(value = UHUGEINT)

SQL examples

```
bin('Aa')
bin(42)
```

See Also

Other numeric: bit_length(), formatReadableDecimalSize(), format_bytes(), greatest(), hex(), least(), len(), parse_formatted_bytes(), to_base()

Other string: array_extract(), ascii(), bar(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

bit_and *DuckDB function bit_and*

Description

Returns the bitwise AND of all bits in a given expression.

Usage

```
bit_and(arg)
```

Arguments

arg	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT

Value

TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT

SQL examples

```
bit_and(A)
```

---

bit_count *DuckDB function bit_count*

Description

Returns the number of bits that are set.

Usage

```
bit_count(x)
```

Arguments

x	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | BIT

Value

TINYINT | BIGINT

SQL examples

```
bit_count(31)
```

---

bit_length *DuckDB function bit_length*

Description

- `bit_length(string = VARCHAR)`: Number of bits in a `string`.
- `bit_length(bit = BIT)`: Returns the bit-length of the `bit` argument.

Usage

```
bit_length(string)
```

Arguments

string	VARCHAR

Value

BIGINT

Overloads

- bit_length(string = VARCHAR)
- bit_length(bit = BIT)

SQL examples

```
bit_length('abc')
bit_length(42::TINYINT::BIT)
```

See Also

Other numeric: bin(), formatReadableDecimalSize(), format_bytes(), greatest(), hex(), least(), len(), parse_formatted_bytes(), to_base()

Other string: array_extract(), ascii(), bar(), bin(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

bit_or *DuckDB function bit_or*

Description

Returns the bitwise OR of all bits in a given expression.

Usage

```
bit_or(arg)
```

Arguments

arg	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT

Value

TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT

SQL examples

```
bit_or(A)
```

`bit_position` *DuckDB function bit_position*

Description

Returns first starting index of the specified substring within bits, or zero if it is not present. The first (leftmost) bit is indexed 1.

Usage

```
bit_position(substring, bitstring)
```

Arguments

`substring`, `bitstring`	`BIT`

Value

```
INTEGER
```

SQL examples

```
bit_position('010'::BIT, '1110101'::BIT)
```

`bit_xor` *DuckDB function bit_xor*

Description

Returns the bitwise XOR of all bits in a given expression.

Usage

```
bit_xor(arg)
```

Arguments

`arg`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

Value

```
TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER
| UBIGINT | UHUGEINT | BIT
```

SQL examples

```
bit_xor(A)
```

`bitstring` *DuckDB function bitstring*

Description

Pads the bitstring until the specified length.

Usage

```
bitstring(bitstring, length)
```

Arguments

`bitstring`	`VARCHAR | BIT`
`length`	`INTEGER`

Value

`BIT`

SQL examples

```
bitstring('1010'::BIT, 7)
```

`bitstring_agg` *DuckDB function bitstring_agg*

Description

Returns a bitstring with bits set for each distinct value.

Usage

```
bitstring_agg(arg, col1, col2)
```

Arguments

`arg`, `col1`, `col2`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

Value

`BIT`

Overloads

- bitstring_agg(arg = TINYINT)
- bitstring_agg(arg = TINYINT, col1 = TINYINT, col2 = TINYINT)
- bitstring_agg(arg = SMALLINT)
- bitstring_agg(arg = SMALLINT, col1 = SMALLINT, col2 = SMALLINT)
- bitstring_agg(arg = INTEGER)
- bitstring_agg(arg = INTEGER, col1 = INTEGER, col2 = INTEGER)
- bitstring_agg(arg = BIGINT)
- bitstring_agg(arg = BIGINT, col1 = BIGINT, col2 = BIGINT)
- bitstring_agg(arg = HUGEINT)
- bitstring_agg(arg = HUGEINT, col1 = HUGEINT, col2 = HUGEINT)
- bitstring_agg(arg = UTINYINT)
- bitstring_agg(arg = UTINYINT, col1 = UTINYINT, col2 = UTINYINT)
- bitstring_agg(arg = USMALLINT)
- bitstring_agg(arg = USMALLINT, col1 = USMALLINT, col2 = USMALLINT)
- bitstring_agg(arg = UINTEGER)
- bitstring_agg(arg = UINTEGER, col1 = UINTEGER, col2 = UINTEGER)
- bitstring_agg(arg = UBIGINT)
- bitstring_agg(arg = UBIGINT, col1 = UBIGINT, col2 = UBIGINT)
- bitstring_agg(arg = UHUGEINT)
- bitstring_agg(arg = UHUGEINT, col1 = UHUGEINT, col2 = UHUGEINT)

SQL examples

```
bitstring_agg(A)
```

---

bool_and *DuckDB function bool_and*

Description

Returns TRUE if every input value is TRUE, otherwise FALSE.

Usage

```
bool_and(arg)
```

Arguments

arg	BOOLEAN

Value

BOOLEAN

SQL examples

```
bool_and(A)
```

`bool_or` *DuckDB function bool_or*

Description

Returns TRUE if any input value is TRUE, otherwise FALSE.

Usage

```
bool_or(arg)
```

Arguments

arg	BOOLEAN

Value

BOOLEAN

SQL examples

```
bool_or(A)
```

`broadcast` *DuckDB function broadcast*

Description

DuckDB function `broadcast()`.

Usage

```
broadcast(col0)
```

Arguments

col0	INET

Value

INET

Provided by

The inet extension (LOAD inet;).

---

can_cast_implicitly *DuckDB function can_cast_implicitly*

Description

Whether or not we can implicitly cast from the source type to the other type.

Usage

```
can_cast_implicitly(source_type, target_type)
```

Arguments

source_type, target_type	ANY

Value

BOOLEAN

SQL examples

```
can_cast_implicitly(NULL::INTEGER, NULL::BIGINT)
```

---

cardinality *DuckDB function cardinality*

Description

Returns the size of the map (or the number of entries in the map).

Usage

```
cardinality(map)
```

Arguments

map	ANY

Value

UBIGINT

SQL examples

```
cardinality( map([4, 2], ['a', 'b']) );
```

`cast_to_type` *DuckDB function cast_to_type*

Description

Casts the first argument to the type of the second argument.

Usage

```
cast_to_type(param, type)
```

Arguments

param, type	ANY

Value

ANY

SQL examples

```
cast_to_type('42', NULL::INTEGER)
```

`cbrt` *DuckDB function cbrt*

Description

Returns the cube root of x.

Usage

```
cbrt(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
cbrt(8)
```

`ceil` *DuckDB function ceil*

Description

Rounds the number up.

Usage

```
ceil(x)
```

Arguments

x	FLOAT | DOUBLE | DECIMAL

Value

FLOAT | DOUBLE | DECIMAL

SQL examples

```
ceil(17.4)
```

`century` *DuckDB function century*

Description

Extract the century component from a date or timestamp.

Usage

```
century(ts)
```

Arguments

ts	DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

BIGINT

SQL examples

```
century(timestamp '2021-08-03 11:59:44.123456')
```

`check_peg_parser` *DuckDB function check_peg_parser*

Description

DuckDB function `check_peg_parser()`.

Usage

```
check_peg_parser(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `autocomplete` extension (`LOAD autocomplete;`).

`checkpoint` *DuckDB function checkpoint*

Description

DuckDB function `checkpoint()`.

Usage

```
checkpoint(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Overloads

- `checkpoint()`
- `checkpoint(col0 = VARCHAR)`

chr *DuckDB function chr*

Description

Returns a character which is corresponding the ASCII code value or Unicode code point.

Usage

```
chr(code_point)
```

Arguments

code_point	INTEGER

Value

VARCHAR

SQL examples

```
chr(65)
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

col_description *DuckDB function col_description*

Description

DuckDB macro col_description().

Usage

```
col_description(table_oid, column_number)
```

Arguments

`table_oid`, `column_number`
: Unspecified.

Value

Unspecified.

`collations` *DuckDB function collations*

Description

DuckDB function `collations()`.

Usage

```
collations()
```

Value

Unspecified.

`combine` *DuckDB function combine*

Description

DuckDB function `combine()`.

Usage

```
combine(col0, col1)
```

Arguments

`col0`	`AGGREGATE_STATE<?>`
`col1`	`ANY`

Value

`AGGREGATE_STATE<?>`

concat *DuckDB function concat*

Description

Concatenates multiple strings or lists. NULL inputs are skipped. See also operator ||.

Usage

```
concat(value)
```

Arguments

value	ANY

Value

ANY

SQL examples

```
concat('Hello', ' ', 'World')
concat([1, 2, 3], NULL, [4, 5, 6])
```

See Also

Other list: array_extract(), array_length(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

concat_ws *DuckDB function concat_ws*

Description

Concatenates many strings, separated by `separator`. `NULL` inputs are skipped.

Usage

```
concat_ws(separator, string)
```

Arguments

separator	VARCHAR
string	ANY

Value

VARCHAR

SQL examples

```
concat_ws(', ', 'Banana', 'Apple', 'Melon')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

constant_or_null *DuckDB function constant_or_null*

Description

If arg2 is NULL, return NULL. Otherwise, return arg1.

Usage

```
constant_or_null(arg1, arg2)
```

Arguments

arg1, arg2 ANY

Value

ANY

SQL examples

```
constant_or_null(42, NULL)
```

---

contains *DuckDB function contains*

Description

- contains(string = VARCHAR, search_string = VARCHAR): Returns true if search_string is found within string.
- contains(list = `T[]`, element = T): Returns true if the list contains the element.
- contains(map = `MAP(K, V)`, key = K): Checks if a map contains a given key.
- contains(col0 = STRUCT, col1 = ANY): (Description missing.)

Usage

```
contains(string, search_string)
```

Arguments

string, search_string
VARCHAR

Value

BOOLEAN

Overloads

- contains(string = VARCHAR, search_string = VARCHAR)
- contains(list = `T[]`, element = T)
- contains(map = `MAP(K, V)`, key = K)
- contains(col0 = STRUCT, col1 = ANY)

SQL examples

```
contains('abc', 'a')
contains([1, 2, NULL], 1)
contains(MAP {'key1': 10, 'key2': 20, 'key3': 30}, 'key2')
```

See Also

Other list: array_extract(), array_length(), concat(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

copy_database *DuckDB function copy_database*

Description

DuckDB function copy_database().

Usage

```
copy_database(col0, col1)
```

Arguments

col0, col1	VARCHAR

Value

Unspecified.

`copy_dir` *DuckDB function copy_dir*

Description

DuckDB macro `copy_dir()`.

Usage

```
copy_dir(src_dir, dst_dir)
```

Arguments

`src_dir`, `dst_dir`
: Unspecified.

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`corr` *DuckDB function corr*

Description

Returns the correlation coefficient for non-NULL pairs in a group.

Usage

```
corr(y, x)
```

Arguments

`y`, `x` `DOUBLE`

Value

`DOUBLE`

SQL examples

```
COVAR_POP(y, x) / (STDDEV_POP(x) * STDDEV_POP(y))
```

`cos` *DuckDB function cos*

Description

Computes the cos of x.

Usage

```
cos(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
cos(90)
```

`cosh` *DuckDB function cosh*

Description

Computes the hyperbolic cos of x.

Usage

```
cosh(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
cosh(1)
```

`cot` *DuckDB function cot*

Description

Computes the cotangent of x.

Usage

```
cot(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
cot(0.5)
```

`count` *DuckDB function count*

Description

Returns the number of non-NULL values in arg.

Usage

```
count(arg)
```

Arguments

arg	ANY

Value

BIGINT

Overloads

- `count(arg = ANY)`
- `count()`

SQL examples

```
count(A)
```

`count_if` *DuckDB function count_if*

Description

Counts the total number of TRUE values for a boolean column.

Usage

```
count_if(arg)
```

Arguments

`arg`	`BOOLEAN`

Value

`HUGEINT`

SQL examples

```
count_if(A)
```

`count_star` *DuckDB function count_star*

Description

DuckDB function `count_star()`.

Usage

```
count_star()
```

Value

`BIGINT`

`covar_pop` *DuckDB function covar_pop*

Description

Returns the population covariance of input values.

Usage

```
covar_pop(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
(SUM(x*y) - SUM(x) * SUM(y) / COUNT(*)) / COUNT(*)
```

`covar_samp` *DuckDB function covar_samp*

Description

Returns the sample covariance for non-NULL pairs in a group.

Usage

```
covar_samp(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
(SUM(x*y) - SUM(x) * SUM(y) / COUNT(*)) / (COUNT(*) - 1)
```

`create_fts_index` *DuckDB function create_fts_index*

Description

DuckDB function `create_fts_index()`.

Usage

```
create_fts_index(col0, col1, overwrite, lower, ignore, stopwords, strip_accents, stemmer)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`overwrite`	`BOOLEAN`
`lower`	`BOOLEAN`
`ignore`	`VARCHAR`
`stopwords`	`VARCHAR`
`strip_accents`	`BOOLEAN`
`stemmer`	`VARCHAR`

Value

Unspecified.

Provided by

The `fts` extension (`LOAD fts;`).

`create_sort_key` *DuckDB function create_sort_key*

Description

Constructs a binary-comparable sort key based on a set of input parameters and sort qualifiers.

Usage

```
create_sort_key(parameters...)
```

Arguments

`parameters...`	`ANY`

Value

BLOB

SQL examples

```
create_sort_key('A', 'DESC')
```

---

cume_dist *DuckDB function cume_dist*

Description

DuckDB function `cume_dist()`.

Usage

```
cume_dist()
```

Value

DOUBLE

---

current_catalog *DuckDB function current_catalog*

Description

DuckDB macro `current_catalog()`.

Usage

```
current_catalog()
```

Value

Unspecified.

`current_connection_id`
DuckDB function current_connection_id

Description

Get the current connection_id.

Usage

```
current_connection_id()
```

Value

```
UBIGINT
```

SQL examples

```
current_connection_id()
```

`current_database` *DuckDB function current_database*

Description

Returns the name of the currently active database.

Usage

```
current_database()
```

Value

```
VARCHAR
```

SQL examples

```
current_database()
```

current_date *DuckDB function current_date*

Description

DuckDB function `current_date()`.

Usage

```
current_date()
```

Value

`DATE`

Provided by

The `icu` extension (`LOAD icu;`).

current_localtime *DuckDB function current_localtime*

Description

DuckDB function `current_localtime()`.

Usage

```
current_localtime()
```

Value

`TIME`

Provided by

The `icu` extension (`LOAD icu;`).

`current_localtimestamp`
DuckDB function current_localtimestamp

Description

DuckDB function `current_localtimestamp()`.

Usage

```
current_localtimestamp()
```

Value

`TIMESTAMP`

Provided by

The `icu` extension (`LOAD icu;`).

`current_query` *DuckDB function current_query*

Description

Returns the current query as a string.

Usage

```
current_query()
```

Value

`VARCHAR`

SQL examples

```
current_query()
```

current_query_id *DuckDB function current_query_id*

Description

Get the current query_id.

Usage

```
current_query_id()
```

Value

```
UBIGINT
```

SQL examples

```
current_query_id()
```

current_role *DuckDB function current_role*

Description

DuckDB macro `current_role()`.

Usage

```
current_role()
```

Value

Unspecified.

`current_schema` *DuckDB function current_schema*

Description

Returns the name of the currently active schema. Default is main.

Usage

```
current_schema()
```

Value

```
VARCHAR
```

SQL examples

```
current_schema()
```

`current_schemas` *DuckDB function current_schemas*

Description

Returns list of schemas. Pass a parameter of True to include implicit schemas.

Usage

```
current_schemas(include_implicit)
```

Arguments

```
include_implicit
               BOOLEAN
```

Value

```
VARCHAR[]
```

SQL examples

```
current_schemas(true)
```

`current_setting` *DuckDB function current_setting*

Description

Returns the current value of the configuration setting.

Usage

```
current_setting(setting_name)
```

Arguments

```
setting_name   VARCHAR
```

Value

```
ANY
```

SQL examples

```
current_setting('access_mode')
```

`current_transaction_id` *DuckDB function current_transaction_id*

Description

Get the current global transaction_id.

Usage

```
current_transaction_id()
```

Value

```
UBIGINT
```

SQL examples

```
current_transaction_id()
```

`current_user` *DuckDB function current_user*

Description

DuckDB macro `current_user()`.

Usage

```
current_user()
```

Value

Unspecified.

`currval` *DuckDB function currval*

Description

Return the current value of the sequence. Note that nextval must be called at least once prior to calling currval.

Usage

```
currval(sequence_name)
```

Arguments

```
sequence_name  VARCHAR
```

Value

```
BIGINT
```

SQL examples

```
currval('my_sequence_name')
```

damerau_levenshtein *DuckDB function damerau_levenshtein*

Description

Extension of Levenshtein distance to also include transposition of adjacent characters as an allowed edit operation. In other words, the minimum number of edit operations (insertions, deletions, substitutions or transpositions) required to change one string to another. Characters of different cases (e.g., `a` and `A`) are considered different.

Usage

```
damerau_levenshtein(s1, s2)
```

Arguments

`s1`, `s2` VARCHAR

Value

`BIGINT`

SQL examples

```
damerau_levenshtein('duckdb', 'udckbd')
```

See Also

Other text_similarity: `hamming()`, `jaccard()`, `jaro_similarity()`, `jaro_winkler_similarity()`, `levenshtein()`

database_list *DuckDB function database_list*

Description

DuckDB function `database_list()`.

Usage

```
database_list()
```

Value

Unspecified.

`database_size` *DuckDB function database_size*

Description

DuckDB function `database_size()`.

Usage

```
database_size()
```

Value

Unspecified.

`date_add` *DuckDB function date_add*

Description

DuckDB macro `date_add()`.

Usage

```
date_add(date, interval)
```

Arguments

`date`, `interval` Unspecified.

Value

Unspecified.

`date_diff` *DuckDB function date_diff*

Description

The number of partition boundaries between the timestamps.

Usage

```
date_diff(part, startdate, enddate)
```

Arguments

part	VARCHAR
startdate	DATE | TIME | TIMESTAMP | TIMESTAMP WITH TIME ZONE
enddate	DATE | TIME | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

```
BIGINT
```

SQL examples

```
date_diff('hour', TIMESTAMPTZ '1992-09-30 23:59:59', TIMESTAMPTZ '1992-10-01 01:58:00')
```

`date_part` *DuckDB function date_part*

Description

Get subfield (equivalent to extract).

Usage

```
date_part(ts, col1)
```

Arguments

ts	VARCHAR[] | VARCHAR
col1	DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W

Value

```
STRUCT() | BIGINT
```

SQL examples

```
date_part('minute', TIMESTAMP '1992-09-20 20:38:40')
```

`date_sub` *DuckDB function date_sub*

Description

The number of complete partitions between the timestamps.

Usage

```
date_sub(part, startdate, enddate)
```

Arguments

part	VARCHAR
startdate	DATE | TIME | TIMESTAMP | TIMESTAMP WITH TIME ZONE
enddate	DATE | TIME | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

BIGINT

SQL examples

```
date_sub('hour', TIMESTAMPTZ '1992-09-30 23:59:59', TIMESTAMPTZ '1992-10-01 01:58:00')
```

`date_trunc` *DuckDB function date_trunc*

Description

Truncate to specified precision.

Usage

```
date_trunc(part, timestamp)
```

Arguments

part	VARCHAR
timestamp	DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

TIMESTAMP | INTERVAL | TIMESTAMP WITH TIME ZONE

SQL examples

```
date_trunc('hour', TIMESTAMPTZ '1992-09-20 20:38:40')
```

`day` *DuckDB function day*

Description

Extract the day component from a date or timestamp.

Usage

```
day(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
day(timestamp '2021-08-03 11:59:44.123456')
```

`dayname` *DuckDB function dayname*

Description

The (English) name of the weekday.

Usage

```
dayname(ts)
```

Arguments

`ts`	`DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`VARCHAR`

SQL examples

```
dayname(TIMESTAMP '1992-03-22')
```

`dayofmonth` *DuckDB function dayofmonth*

Description

Extract the dayofmonth component from a date or timestamp.

Usage

```
dayofmonth(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
dayofmonth(timestamp '2021-08-03 11:59:44.123456')
```

`dayofweek` *DuckDB function dayofweek*

Description

Extract the dayofweek component from a date or timestamp.

Usage

```
dayofweek(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
dayofweek(timestamp '2021-08-03 11:59:44.123456')
```

`dayofyear` *DuckDB function dayofyear*

Description

Extract the dayofyear component from a date or timestamp.

Usage

```
dayofyear(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
dayofyear(timestamp '2021-08-03 11:59:44.123456')
```

`days_in_month` *DuckDB function days_in_month*

Description

DuckDB macro `days_in_month()`.

Usage

```
days_in_month(date)
```

Arguments

`date`	Unspecified.

Value

Unspecified.

`dbgen` *DuckDB function dbgen*

Description

DuckDB function `dbgen()`.

Usage

```
dbgen(step, suffix, schema, catalog, overwrite, children, sf)
```

Arguments

`step`	`UINTEGER`
`suffix`	`VARCHAR`
`schema`	`VARCHAR`
`catalog`	`VARCHAR`
`overwrite`	`BOOLEAN`
`children`	`UINTEGER`
`sf`	`DOUBLE`

Value

Unspecified.

Provided by

The `tpch` extension (`LOAD tpch;`).

`dd` *DuckDB functions*

Description

A list of known DuckDB functions, as provided by DuckDB 1.5.5.

Usage

```
dd
```

Examples

```
dd[1:3]
```

`decade` *DuckDB function decade*

Description

Extract the decade component from a date or timestamp.

Usage

```
decade(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
decade(timestamp '2021-08-03 11:59:44.123456')
```

`decode` *DuckDB function decode*

Description

Converts `blob` to `VARCHAR`. Invalid UTF-8 is handled based on the error behavior argument. Can be 'strict' (default, fail), 'replace' to replace invalid characters with '?', or 'ignore' to skip invalid characters.

Usage

```
decode(blob, varchar)
```

Arguments

`blob`	`BLOB`
`varchar`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `decode(blob = BLOB)`
- `decode(blob = BLOB, varchar = VARCHAR)`

SQL examples

```
decode('\xC3\xBC'::BLOB)
decode('\xA0'::BLOB, 'replace')
decode('\xA0'::BLOB, 'ignore')
```

See Also

Other blob: `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unbin()`, `unhex()`

`degrees` *DuckDB function degrees*

Description

Converts radians to degrees.

Usage

```
degrees(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
degrees(pi())
```

`delta_domain_metadata`

DuckDB function delta_domain_metadata

Description

DuckDB function `delta_domain_metadata()`.

Usage

```
delta_domain_metadata(col0, version)
```

Arguments

`col0`	`VARCHAR`
`version`	`UBIGINT`

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`delta_filter_pushdown_log`

DuckDB function delta_filter_pushdown_log

Description

DuckDB macro `delta_filter_pushdown_log()`.

Usage

```
delta_filter_pushdown_log()
```

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`delta_filter_pushdown_log_tpcds`
DuckDB function delta_filter_pushdown_log_tpcds

Description

DuckDB macro `delta_filter_pushdown_log_tpcds()`.

Usage

```
delta_filter_pushdown_log_tpcds()
```

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`delta_get_transaction_version`
DuckDB function delta_get_transaction_version

Description

DuckDB function `delta_get_transaction_version()`.

Usage

```
delta_get_transaction_version(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

delta_list_files *DuckDB function delta_list_files*

Description

DuckDB function `delta_list_files()`.

Usage

```
delta_list_files(col0, version, delete_count, transform_expression)
```

Arguments

col0	VARCHAR
version	UBIGINT
delete_count	BOOLEAN
transform_expression	BOOLEAN

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

delta_scan *DuckDB function delta_scan*

Description

DuckDB function `delta_scan()`.

Usage

```
delta_scan(
  col0,
  version,
  log_tail,
  pushdown_partition_info,
  can_have_nan,
  filename,
  pushdown_filters,
  union_by_name,
```

```
  debug_use_openssl,
  hive_partitioning,
  parquet_version,
  encryption_config,
  hive_types_autocast,
  binary_as_string,
  explicit_cardinality,
  compression,
  file_row_number,
  hive_types
)
```

Arguments

col0	VARCHAR | VARCHAR[]
version	UBIGINT
log_tail	STRUCT(file_name VARCHAR, "timestamp" BIGINT, file_size UBIGINT)[]
pushdown_partition_info	BOOLEAN
can_have_nan	BOOLEAN
filename	ANY
pushdown_filters	VARCHAR
union_by_name	BOOLEAN
debug_use_openssl	BOOLEAN
hive_partitioning	BOOLEAN
parquet_version	VARCHAR
encryption_config	ANY
hive_types_autocast	BOOLEAN
binary_as_string	BOOLEAN
explicit_cardinality	UBIGINT
compression	VARCHAR
file_row_number	BOOLEAN
hive_types	ANY

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`delta_set_transaction_version` *DuckDB function delta_set_transaction_version*

Description

DuckDB function `delta_set_transaction_version()`.

Usage

```
delta_set_transaction_version(col0, col1, col2, col3)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`col2`	`UBIGINT`
`col3`	`UBIGINT`

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`dense_rank` *DuckDB function dense_rank*

Description

DuckDB function `dense_rank()`.

Usage

```
dense_rank()
```

Value

`BIGINT`

`disable_checkpoint_on_shutdown` *DuckDB function disable_checkpoint_on_shutdown*

Description

DuckDB function `disable_checkpoint_on_shutdown()`.

Usage

```
disable_checkpoint_on_shutdown()
```

Value

Unspecified.

`disable_logging` *DuckDB function disable_logging*

Description

DuckDB function `disable_logging()`.

Usage

```
disable_logging()
```

Value

Unspecified.

`disable_object_cache` *DuckDB function disable_object_cache*

Description

DuckDB function `disable_object_cache()`.

Usage

```
disable_object_cache()
```

Value

Unspecified.

disable_optimizer *DuckDB function disable_optimizer*

Description

DuckDB function `disable_optimizer()`.

Usage

```
disable_optimizer()
```

Value

Unspecified.

disable_peg_parser *DuckDB function disable_peg_parser*

Description

DuckDB function `disable_peg_parser()`.

Usage

```
disable_peg_parser()
```

Value

Unspecified.

Provided by

The `autocomplete` extension (`LOAD autocomplete;`).

`disable_print_progress_bar` *DuckDB function disable_print_progress_bar*

Description

DuckDB function `disable_print_progress_bar()`.

Usage

```
disable_print_progress_bar()
```

Value

Unspecified.

`disable_profile` *DuckDB function disable_profile*

Description

DuckDB function `disable_profile()`.

Usage

```
disable_profile()
```

Value

Unspecified.

`disable_profiling` *DuckDB function disable_profiling*

Description

DuckDB function `disable_profiling()`.

Usage

```
disable_profiling()
```

Value

Unspecified.

`disable_progress_bar` *DuckDB function disable_progress_bar*

Description

DuckDB function `disable_progress_bar()`.

Usage

```
disable_progress_bar()
```

Value

Unspecified.

`disable_verification` *DuckDB function disable_verification*

Description

DuckDB function `disable_verification()`.

Usage

```
disable_verification()
```

Value

Unspecified.

`disable_verify_external`
DuckDB function disable_verify_external

Description

DuckDB function `disable_verify_external()`.

Usage

```
disable_verify_external()
```

Value

Unspecified.

`disable_verify_fetch_row`
DuckDB function disable_verify_fetch_row

Description

DuckDB function `disable_verify_fetch_row()`.

Usage

```
disable_verify_fetch_row()
```

Value

Unspecified.

`disable_verify_parallelism`
DuckDB function disable_verify_parallelism

Description

DuckDB function `disable_verify_parallelism()`.

Usage

```
disable_verify_parallelism()
```

Value

Unspecified.

`disable_verify_serializer`
DuckDB function disable_verify_serializer

Description

DuckDB function `disable_verify_serializer()`.

Usage

```
disable_verify_serializer()
```

Value

Unspecified.

`divide` *DuckDB function divide*

Description

DuckDB function `divide()`.

Usage

```
divide(col0, col1)
```

Arguments

`col0`, `col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

`drop_fts_index` *DuckDB function drop_fts_index*

Description

DuckDB function `drop_fts_index()`.

Usage

```
drop_fts_index(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `fts` extension (`LOAD fts;`).

`dsdgen` *DuckDB function dsdgen*

Description

DuckDB function `dsdgen()`.

Usage

```
dsdgen(suffix, schema, catalog, keys, overwrite, sf)
```

Arguments

`suffix`	`VARCHAR`
`schema`	`VARCHAR`
`catalog`	`VARCHAR`
`keys`	`BOOLEAN`
`overwrite`	`BOOLEAN`
`sf`	`DOUBLE`

Value

Unspecified.

Provided by

The `tpcds` extension (`LOAD tpcds;`).

`duckdb_approx_database_count`
DuckDB function duckdb_approx_database_count

Description

DuckDB function `duckdb_approx_database_count()`.

Usage

```
duckdb_approx_database_count()
```

Value

Unspecified.

`duckdb_columns` *DuckDB function duckdb_columns*

Description

DuckDB function `duckdb_columns()`.

Usage

```
duckdb_columns()
```

Value

Unspecified.

`duckdb_connection_count`
DuckDB function duckdb_connection_count

Description

DuckDB function `duckdb_connection_count()`.

Usage

```
duckdb_connection_count()
```

Value

Unspecified.

`duckdb_constraints` *DuckDB function duckdb_constraints*

Description

DuckDB function `duckdb_constraints()`.

Usage

```
duckdb_constraints()
```

Value

Unspecified.

duckdb_coordinate_systems

DuckDB function duckdb_coordinate_systems

Description

DuckDB function `duckdb_coordinate_systems()`.

Usage

```
duckdb_coordinate_systems()
```

Value

Unspecified.

duckdb_databases

DuckDB function duckdb_databases

Description

DuckDB function `duckdb_databases()`.

Usage

```
duckdb_databases()
```

Value

Unspecified.

duckdb_dependencies

DuckDB function duckdb_dependencies

Description

DuckDB function `duckdb_dependencies()`.

Usage

```
duckdb_dependencies()
```

Value

Unspecified.

duckdb_extensions *DuckDB function duckdb_extensions*

Description

DuckDB function `duckdb_extensions()`.

Usage

```
duckdb_extensions()
```

Value

Unspecified.

duckdb_external_file_cache *DuckDB function duckdb_external_file_cache*

Description

DuckDB function `duckdb_external_file_cache()`.

Usage

```
duckdb_external_file_cache()
```

Value

Unspecified.

duckdb_functions *DuckDB function duckdb_functions*

Description

DuckDB function `duckdb_functions()`.

Usage

```
duckdb_functions()
```

Value

Unspecified.

duckdb_indexes *DuckDB function duckdb_indexes*

Description

DuckDB function `duckdb_indexes()`.

Usage

```
duckdb_indexes()
```

Value

Unspecified.

duckdb_keywords *DuckDB function duckdb_keywords*

Description

DuckDB function `duckdb_keywords()`.

Usage

```
duckdb_keywords()
```

Value

Unspecified.

duckdb_log_contexts *DuckDB function duckdb_log_contexts*

Description

DuckDB function `duckdb_log_contexts()`.

Usage

```
duckdb_log_contexts()
```

Value

Unspecified.

duckdb_logs *DuckDB function duckdb_logs*

Description

DuckDB function `duckdb_logs()`.

Usage

```
duckdb_logs(denormalized_table)
```

Arguments

```
denormalized_table
               BOOLEAN
```

Value

Unspecified.

duckdb_logs_parsed *DuckDB function duckdb_logs_parsed*

Description

DuckDB macro `duckdb_logs_parsed()`.

Usage

```
duckdb_logs_parsed(log_type)
```

Arguments

`log_type`	Unspecified.

Value

Unspecified.

duckdb_memory *DuckDB function duckdb_memory*

Description

DuckDB function `duckdb_memory()`.

Usage

```
duckdb_memory()
```

Value

Unspecified.

duckdb_optimizers *DuckDB function duckdb_optimizers*

Description

DuckDB function `duckdb_optimizers()`.

Usage

```
duckdb_optimizers()
```

Value

Unspecified.

duckdb_prepared_statements *DuckDB function duckdb_prepared_statements*

Description

DuckDB function `duckdb_prepared_statements()`.

Usage

```
duckdb_prepared_statements()
```

Value

Unspecified.

`duckdb_profiling_settings`
DuckDB function duckdb_profiling_settings

Description

DuckDB macro `duckdb_profiling_settings()`.

Usage

```
duckdb_profiling_settings()
```

Value

Unspecified.

`DuckDB_PROJ_Compiled_Version`
DuckDB function DuckDB_PROJ_Compiled_Version

Description

Returns a text description of the PROJ library version that this instance of DuckDB was compiled against.

Usage

```
DuckDB_PROJ_Compiled_Version()
```

Value

`VARCHAR`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT duckdb_proj_compiled_version();
-- +--------------------------------+
-- | duckdb_proj_compiled_version() |
-- |            varchar             |
-- +--------------------------------+
-- | Rel. 9.1.1, December 1st, 2022 |
-- +--------------------------------+
```

`DuckDB_Proj_Version` *DuckDB function DuckDB_Proj_Version*

Description

Returns a text description of the PROJ library version that is being used by this instance of DuckDB.

Usage

```
DuckDB_Proj_Version()
```

Value

```
VARCHAR
```

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT duckdb_proj_version();
-- +-----------------------+
-- | duckdb_proj_version() |
-- |        varchar        |
-- +-----------------------+
-- | 9.1.1                 |geometry_always_xy
-- +-----------------------+
```

`duckdb_schemas` *DuckDB function duckdb_schemas*

Description

DuckDB function `duckdb_schemas()`.

Usage

```
duckdb_schemas()
```

Value

Unspecified.

`duckdb_secret_types` *DuckDB function duckdb_secret_types*

Description

DuckDB function `duckdb_secret_types()`.

Usage

```
duckdb_secret_types()
```

Value

Unspecified.

`duckdb_secrets` *DuckDB function duckdb_secrets*

Description

DuckDB function `duckdb_secrets()`.

Usage

```
duckdb_secrets(redact)
```

Arguments

`redact`	`BOOLEAN`

Value

Unspecified.

`duckdb_sequences` *DuckDB function duckdb_sequences*

Description

DuckDB function `duckdb_sequences()`.

Usage

```
duckdb_sequences()
```

Value

Unspecified.

`duckdb_settings` *DuckDB function duckdb_settings*

Description

DuckDB function `duckdb_settings()`.

Usage

```
duckdb_settings()
```

Value

Unspecified.

`duckdb_table_sample` *DuckDB function duckdb_table_sample*

Description

DuckDB function `duckdb_table_sample()`.

Usage

```
duckdb_table_sample(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`duckdb_tables` *DuckDB function duckdb_tables*

Description

DuckDB function `duckdb_tables()`.

Usage

```
duckdb_tables()
```

Value

Unspecified.

duckdb_temporary_files

DuckDB function duckdb_temporary_files

Description

DuckDB function `duckdb_temporary_files()`.

Usage

```
duckdb_temporary_files()
```

Value

Unspecified.

duckdb_types

DuckDB function duckdb_types

Description

DuckDB function `duckdb_types()`.

Usage

```
duckdb_types()
```

Value

Unspecified.

duckdb_variables

DuckDB function duckdb_variables

Description

DuckDB function `duckdb_variables()`.

Usage

```
duckdb_variables()
```

Value

Unspecified.

`duckdb_views` *DuckDB function duckdb_views*

Description

DuckDB function `duckdb_views()`.

Usage

```
duckdb_views()
```

Value

Unspecified.

`ducklake_add_data_files`
DuckDB function ducklake_add_data_files

Description

DuckDB function `ducklake_add_data_files()`.

Usage

```
ducklake_add_data_files(
  col0,
  col1,
  col2,
  schema,
  hive_partitioning,
  ignore_extra_columns,
  allow_missing
)
```

Arguments

col0	VARCHAR
col1	VARCHAR
col2	VARCHAR | VARCHAR[]
schema	VARCHAR
hive_partitioning	BOOLEAN
ignore_extra_columns	BOOLEAN
allow_missing	BOOLEAN

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_cleanup_old_files`
DuckDB function ducklake_cleanup_old_files

Description

DuckDB function `ducklake_cleanup_old_files()`.

Usage

```
ducklake_cleanup_old_files(col0, dry_run, cleanup_all, older_than)
```

Arguments

`col0`	`VARCHAR`
`dry_run`	`BOOLEAN`
`cleanup_all`	`BOOLEAN`
`older_than`	`TIMESTAMP WITH TIME ZONE`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_commit` *DuckDB function ducklake_commit*

Description

DuckDB function `ducklake_commit()`.

Usage

```
ducklake_commit(col0, col1, retry_backoff, retry_wait_ms, max_retry_count)
```

Arguments

`col0`	`VARCHAR`
`col1`	`BIGINT`
`retry_backoff`	`DOUBLE`
`retry_wait_ms`	`BIGINT`
`max_retry_count`	`BIGINT`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_current_snapshot`
DuckDB function ducklake_current_snapshot

Description

DuckDB function `ducklake_current_snapshot()`.

Usage

```
ducklake_current_snapshot(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_delete_orphaned_files`
DuckDB function ducklake_delete_orphaned_files

Description

DuckDB function `ducklake_delete_orphaned_files()`.

Usage

```
ducklake_delete_orphaned_files(col0, dry_run, cleanup_all, older_than)
```

Arguments

`col0`	`VARCHAR`
`dry_run`	`BOOLEAN`
`cleanup_all`	`BOOLEAN`
`older_than`	`TIMESTAMP WITH TIME ZONE`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_expire_snapshots`
DuckDB function ducklake_expire_snapshots

Description

DuckDB function `ducklake_expire_snapshots()`.

Usage

```
ducklake_expire_snapshots(col0, dry_run, versions, older_than)
```

Arguments

`col0`	`VARCHAR`
`dry_run`	`BOOLEAN`
`versions`	`UBIGINT[]`
`older_than`	`TIMESTAMP WITH TIME ZONE`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_flush_inlined_data`
DuckDB function ducklake_flush_inlined_data

Description

DuckDB function `ducklake_flush_inlined_data()`.

Usage

```
ducklake_flush_inlined_data(col0, table_name, schema_name)
```

Arguments

`col0`, `table_name`, `schema_name`
`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_last_committed_snapshot`
DuckDB function ducklake_last_committed_snapshot

Description

DuckDB function `ducklake_last_committed_snapshot()`.

Usage

```
ducklake_last_committed_snapshot(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_list_files` *DuckDB function ducklake_list_files*

Description

DuckDB function `ducklake_list_files()`.

Usage

```
ducklake_list_files(col0, col1, snapshot_time, snapshot_version, schema)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`snapshot_time`	`TIMESTAMP WITH TIME ZONE`
`snapshot_version`	`BIGINT`
`schema`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_merge_adjacent_files`
DuckDB function ducklake_merge_adjacent_files

Description

DuckDB function `ducklake_merge_adjacent_files()`.

Usage

```
ducklake_merge_adjacent_files(
  col0,
  col1,
  schema,
  max_compacted_files,
  max_file_size,
  min_file_size
)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`schema`	`VARCHAR`
`max_compacted_files`	`UBIGINT`
`max_file_size`	`UBIGINT`
`min_file_size`	`UBIGINT`

Value

Unspecified.

Overloads

- `ducklake_merge_adjacent_files(col0 = VARCHAR, col1 = VARCHAR, schema = VARCHAR, max_compacted_files = UBIGINT, max_file_size = UBIGINT, min_file_size = UBIGINT)`
- `ducklake_merge_adjacent_files(col0 = VARCHAR, max_compacted_files = UBIGINT, max_file_size = UBIGINT, min_file_size = UBIGINT)`

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_options` *DuckDB function ducklake_options*

Description

DuckDB function `ducklake_options()`.

Usage

```
ducklake_options(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_rewrite_data_files`
DuckDB function ducklake_rewrite_data_files

Description

DuckDB function `ducklake_rewrite_data_files()`.

Usage

```
ducklake_rewrite_data_files(col0, col1, schema, delete_threshold)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`schema`	`VARCHAR`
`delete_threshold`	`DOUBLE`

Value

Unspecified.

Overloads

- `ducklake_rewrite_data_files(col0 = VARCHAR, col1 = VARCHAR, schema = VARCHAR, delete_threshold = DOUBLE)`
- `ducklake_rewrite_data_files(col0 = VARCHAR, delete_threshold = DOUBLE)`

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_scan` *DuckDB function ducklake_scan*

Description

DuckDB function `ducklake_scan()`.

Usage

```
ducklake_scan(
  col0,
  can_have_nan,
  filename,
  union_by_name,
  debug_use_openssl,
  hive_partitioning,
  parquet_version,
  encryption_config,
  hive_types_autocast,
  binary_as_string,
  explicit_cardinality,
  compression,
  file_row_number,
  hive_types,
  schema
)
```

Arguments

`col0`	`VARCHAR`
`can_have_nan`	`BOOLEAN`
`filename`	`ANY`
`union_by_name`	`BOOLEAN`

```
debug_use_openssl
               BOOLEAN
hive_partitioning
               BOOLEAN
parquet_version
               VARCHAR
encryption_config
               ANY
hive_types_autocast
               BOOLEAN
binary_as_string
               BOOLEAN
explicit_cardinality
               UBIGINT
compression    VARCHAR
file_row_number
               BOOLEAN
hive_types     ANY
schema         ANY
```

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_set_commit_message`
DuckDB function ducklake_set_commit_message

Description

DuckDB function `ducklake_set_commit_message()`.

Usage

```
ducklake_set_commit_message(col0, col1, col2, extra_info)
```

Arguments

`col0`, `col1`, `col2`, `extra_info`
`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_set_option` *DuckDB function ducklake_set_option*

Description

DuckDB function `ducklake_set_option()`.

Usage

```
ducklake_set_option(col0, col1, col2, schema, table_name)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`col2`	`ANY`
`schema`	`VARCHAR`
`table_name`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_settings` *DuckDB function ducklake_settings*

Description

DuckDB function `ducklake_settings()`.

Usage

```
ducklake_settings(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_snapshots` *DuckDB function ducklake_snapshots*

Description

DuckDB function `ducklake_snapshots()`.

Usage

```
ducklake_snapshots(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_table_changes`
DuckDB function ducklake_table_changes

Description

DuckDB macro `ducklake_table_changes()`.

Usage

```
ducklake_table_changes(catalog, schema_name, table_name, start_snapshot, end_snapshot)
```

Arguments

`catalog`, `schema_name`, `table_name`, `start_snapshot`, `end_snapshot`	Unspecified.

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_table_deletions`
DuckDB function ducklake_table_deletions

Description

DuckDB function `ducklake_table_deletions()`.

Usage

```
ducklake_table_deletions(col0, col1, col2, col3, col4)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`col2`	`VARCHAR`
`col3`	`BIGINT | TIMESTAMP WITH TIME ZONE`
`col4`	`BIGINT | TIMESTAMP WITH TIME ZONE`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_table_info` *DuckDB function ducklake_table_info*

Description

DuckDB function `ducklake_table_info()`.

Usage

```
ducklake_table_info(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`ducklake_table_insertions`
DuckDB function ducklake_table_insertions

Description

DuckDB function `ducklake_table_insertions()`.

Usage

```
ducklake_table_insertions(col0, col1, col2, col3, col4)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`col2`	`VARCHAR`
`col3`	`BIGINT | TIMESTAMP WITH TIME ZONE`
`col4`	`BIGINT | TIMESTAMP WITH TIME ZONE`

Value

Unspecified.

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`enable_checkpoint_on_shutdown` *DuckDB function enable_checkpoint_on_shutdown*

Description

DuckDB function `enable_checkpoint_on_shutdown()`.

Usage

```
enable_checkpoint_on_shutdown()
```

Value

Unspecified.

`enable_logging` *DuckDB function enable_logging*

Description

DuckDB function `enable_logging()`.

Usage

```
enable_logging(
  storage_buffer_size,
  storage_normalize,
  storage_path,
  storage_config,
  storage,
  level
)
```

Arguments

storage_buffer_size	UBIGINT
storage_normalize	BOOLEAN
storage_path	VARCHAR
storage_config	ANY
storage	VARCHAR
level	VARCHAR

Value

Unspecified.

`enable_object_cache` *DuckDB function enable_object_cache*

Description

DuckDB function `enable_object_cache()`.

Usage

```
enable_object_cache()
```

Value

Unspecified.

`enable_optimizer` *DuckDB function enable_optimizer*

Description

DuckDB function `enable_optimizer()`.

Usage

```
enable_optimizer()
```

Value

Unspecified.

`enable_peg_parser` *DuckDB function enable_peg_parser*

Description

DuckDB function `enable_peg_parser()`.

Usage

```
enable_peg_parser()
```

Value

Unspecified.

Provided by

The `autocomplete` extension (`LOAD autocomplete;`).

`enable_print_progress_bar` *DuckDB function enable_print_progress_bar*

Description

DuckDB function `enable_print_progress_bar()`.

Usage

```
enable_print_progress_bar()
```

Value

Unspecified.

enable_profile *DuckDB function enable_profile*

Description

DuckDB function `enable_profile()`.

Usage

```
enable_profile()
```

Value

Unspecified.

enable_profiling *DuckDB function enable_profiling*

Description

DuckDB function `enable_profiling()`.

Usage

```
enable_profiling(mode, metrics, save_location, coverage, format)
```

Arguments

`mode`	`VARCHAR`
`metrics`	`ANY`
`save_location`	`VARCHAR`
`coverage`	`VARCHAR`
`format`	`VARCHAR`

Value

Unspecified.

Overloads

- `enable_profiling(mode = VARCHAR, metrics = ANY, save_location = VARCHAR, coverage = VARCHAR, format = VARCHAR)`
- `enable_profiling()`

enable_progress_bar *DuckDB function enable_progress_bar*

Description

DuckDB function `enable_progress_bar()`.

Usage

```
enable_progress_bar()
```

Value

Unspecified.

enable_verification *DuckDB function enable_verification*

Description

DuckDB function `enable_verification()`.

Usage

```
enable_verification()
```

Value

Unspecified.

encode *DuckDB function encode*

Description

Converts the `string` to `BLOB`. Converts UTF-8 characters into literal encoding.

Usage

```
encode(string)
```

Arguments

`string`	`VARCHAR`

Value

BLOB

SQL examples

```
encode('my_string_with_ü')
```

See Also

Other blob: decode(), from_base64(), hex(), md5(), md5_number(), octet_length(), or--or, repeat(), sha1(), sha256(), to_base64(), unbin(), unhex()

---

entropy *DuckDB function entropy*

Description

Returns the log-2 entropy of count input-values.

Usage

```
entropy(x)
```

Arguments

x	ANY

Value

DOUBLE

---

enum_code *DuckDB function enum_code*

Description

Returns the numeric value backing the given enum value.

Usage

```
enum_code(enum)
```

Arguments

enum	ANY

Value

ANY

SQL examples

```
enum_code('happy'::mood)
```

`enum_first` *DuckDB function enum_first*

Description

Returns the first value of the input enum type.

Usage

```
enum_first(enum)
```

Arguments

enum	ANY

Value

VARCHAR

SQL examples

```
enum_first(NULL::mood)
```

`enum_last` *DuckDB function enum_last*

Description

Returns the last value of the input enum type.

Usage

```
enum_last(enum)
```

Arguments

enum	ANY

Value

VARCHAR

SQL examples

```
enum_last(NULL::mood)
```

---

enum_range *DuckDB function enum_range*

Description

Returns all values of the input enum type as an array.

Usage

```
enum_range(enum)
```

Arguments

enum	ANY

Value

VARCHAR[]

SQL examples

```
enum_range(NULL::mood)
```

---

enum_range_boundary *DuckDB function enum_range_boundary*

Description

Returns the range between the two given enum values as an array. The values must be of the same enum type. When the first parameter is NULL, the result starts with the first value of the enum type. When the second parameter is NULL, the result ends with the last value of the enum type.

Usage

```
enum_range_boundary(start, end)
```

Arguments

start, end	ANY

Value

VARCHAR[]

SQL examples

```
enum_range_boundary(NULL, 'happy'::mood)
```

`epoch` *DuckDB function epoch*

Description

Extract the epoch component from a temporal type.

Usage

```
epoch(temporal)
```

Arguments

temporal	DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W

Value

DOUBLE

SQL examples

```
epoch(timestamp '2021-08-03 11:59:44.123456')
```

`epoch_ms` *DuckDB function epoch_ms*

Description

Extract the epoch component in milliseconds from a temporal type.

Usage

```
epoch_ms(temporal)
```

Arguments

temporal	DATE | TIMESTAMP | INTERVAL | TIME | TIME_NS | TIME WITH TIME ZONE | TIMESTAMP W

Value

BIGINT | TIMESTAMP

SQL examples

```
epoch_ms(timestamp '2021-08-03 11:59:44.123456')
```

`epoch_ns` *DuckDB function epoch_ns*

Description

Extract the epoch component in nanoseconds from a temporal type.

Usage

```
epoch_ns(temporal)
```

Arguments

`temporal`	`DATE | TIMESTAMP | INTERVAL | TIME | TIME_NS | TIME WITH TIME ZONE | TIMESTAMP W`

Value

`BIGINT`

SQL examples

```
epoch_ns(timestamp '2021-08-03 11:59:44.123456')
```

`epoch_us` *DuckDB function epoch_us*

Description

Extract the epoch component in microseconds from a temporal type.

Usage

```
epoch_us(temporal)
```

Arguments

`temporal`	`DATE | TIMESTAMP | INTERVAL | TIME | TIME_NS | TIME WITH TIME ZONE | TIMESTAMP W`

Value

`BIGINT`

SQL examples

```
epoch_us(timestamp '2021-08-03 11:59:44.123456')
```

`equi_width_bins` *DuckDB function equi_width_bins*

Description

Generates bin_count equi-width bins between the min and max. If enabled nice_rounding makes the numbers more readable/less jagged.

Usage

```
equi_width_bins(min, max, bin_count, nice_rounding)
```

Arguments

`min`	`BIGINT | DOUBLE | TIMESTAMP | ANY`
`max`	`BIGINT | DOUBLE | TIMESTAMP | ANY`
`bin_count`	`BIGINT`
`nice_rounding`	`BOOLEAN`

Value

`ANY[]`

SQL examples

```
equi_width_bins(0, 10, 2, true)
```

`era` *DuckDB function era*

Description

Extract the era component from a date or timestamp.

Usage

```
era(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
era(timestamp '2021-08-03 11:59:44.123456')
```

`error` *DuckDB function error*

Description

Throws the given error message.

Usage

```
error(message)
```

Arguments

message	VARCHAR

Value

```
"NULL"
```

SQL examples

```
error('access_mode')
```

`even` *DuckDB function even*

Description

Rounds x to next even number by rounding away from zero.

Usage

```
even(x)
```

Arguments

x	DOUBLE

Value

```
DOUBLE
```

SQL examples

```
even(2.9)
```

`excel_text` *DuckDB function excel_text*

Description

DuckDB function `excel_text()`.

Usage

```
excel_text(col0, col1)
```

Arguments

`col0`	`DOUBLE`
`col1`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `excel` extension (`LOAD excel;`).

`exp` *DuckDB function exp*

Description

Computes e to the power of x.

Usage

```
exp(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
exp(1)
```

extension_versions *DuckDB function extension_versions*

Description

DuckDB function `extension_versions()`.

Usage

```
extension_versions()
```

Value

Unspecified.

factorial *DuckDB function factorial*

Description

Factorial of x. Computes the product of the current integer and all integers below it.

Usage

```
factorial(x)
```

Arguments

x	INTEGER

Value

`HUGEINT`

SQL examples

```
4!
```

`family` *DuckDB function family*

Description

DuckDB function `family()`.

Usage

```
family(col0)
```

Arguments

`col0`	`INET`

Value

`UTINYINT`

Provided by

The `inet` extension (`LOAD inet;`).

`favg` *DuckDB function favg*

Description

Calculates the average using a more accurate floating point summation (Kahan Sum).

Usage

```
favg(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
favg(A)
```

fdiv *DuckDB function fdiv*

Description

DuckDB macro `fdiv()`.

Usage

```
fdiv(x, y)
```

Arguments

`x`, `y`	Unspecified.

Value

Unspecified.

fill *DuckDB function fill*

Description

DuckDB function `fill()`.

Usage

```
fill(col0)
```

Arguments

`col0`	`T`

Value

`T`

`finalize` *DuckDB function finalize*

Description

DuckDB function `finalize()`.

Usage

```
finalize(col0)
```

Arguments

`col0`	`AGGREGATE_STATE<?>`

Value

`INVALID`

`first` *DuckDB function first*

Description

Returns the first value (NULL or non-NULL) from arg. This function is affected by ordering.

Usage

```
first(arg)
```

Arguments

`arg`	`DECIMAL | ANY`

Value

`DECIMAL | ANY | T`

Overloads

- `first(arg = DECIMAL)`
- `first(arg = ANY)`
- `first(col0 = T)`

SQL examples

```
first(A)
```

`first_value` *DuckDB function first_value*

Description

DuckDB function `first_value()`.

Usage

```
first_value(col0)
```

Arguments

`col0`	`T`

Value

`T`

`flatten` *DuckDB function flatten*

Description

Flattens a nested list by one level.

Usage

```
flatten(nested_list)
```

Arguments

`nested_list`	`T[][]`

Value

`T[]`

SQL examples

```
flatten([[1, 2, 3], [4, 5]])
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

---

floor *DuckDB function floor*

Description

Rounds the number down.

Usage

```
floor(x)
```

Arguments

x	FLOAT | DOUBLE | DECIMAL

Value

FLOAT | DOUBLE | DECIMAL

SQL examples

```
floor(17.4)
```

---

fmod *DuckDB function fmod*

Description

DuckDB macro fmod().

Usage

```
fmod(x, y)
```

Arguments

x, y	Unspecified.

Value

Unspecified.

`force_checkpoint` *DuckDB function force_checkpoint*

Description

DuckDB function `force_checkpoint()`.

Usage

```
force_checkpoint()
```

Value

Unspecified.

Overloads

- `force_checkpoint()`
- `force_checkpoint(col0 = VARCHAR)`

`format` *DuckDB function format*

Description

Formats a string using the fmt syntax.

Usage

```
format(format)
```

Arguments

`format`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
format('Benchmark "{}" took {} seconds', 'CSV', 42)
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`format_bytes` *DuckDB function format_bytes*

Description

Converts `integer` to a human-readable representation using units based on powers of 2 (KiB, MiB, GiB, etc.).

Usage

```
format_bytes(integer)
```

Arguments

`integer`	`BIGINT`

Value

`VARCHAR`

SQL examples

```
format_bytes(16_000)
```

See Also

Other numeric: `bin()`, `bit_length()`, `formatReadableDecimalSize()`, `greatest()`, `hex()`, `least()`, `len()`, `parse_formatted_bytes()`, `to_base()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`,

rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

format_pg_type *DuckDB function format_pg_type*

Description

DuckDB macro format_pg_type().

Usage

```
format_pg_type(logical_type, type_name)
```

Arguments

logical_type, type_name
: Unspecified.

Value

Unspecified.

---

format_type *DuckDB function format_type*

Description

DuckDB macro format_type().

Usage

```
format_type(type_oid, typemod)
```

Arguments

type_oid, typemod
: Unspecified.

Value

Unspecified.

`formatReadableDecimalSize`

DuckDB function formatReadableDecimalSize

Description

Converts `integer` to a human-readable representation using units based on powers of 10 (KB, MB, GB, etc.).

Usage

```
formatReadableDecimalSize(integer)
```

Arguments

`integer`	`BIGINT`

Value

`VARCHAR`

SQL examples

```
formatReadableDecimalSize(16_000)
```

See Also

Other numeric: `bin()`, `bit_length()`, `format_bytes()`, `greatest()`, `hex()`, `least()`, `len()`, `parse_formatted_bytes()`, `to_base()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`from_base64` *DuckDB function from_base64*

Description

Converts a base64 encoded `string` to a character string (`BLOB`).

Usage

```
from_base64(string)
```

Arguments

`string`	`VARCHAR`

Value

`BLOB`

SQL examples

```
from_base64('QQ==')
```

See Also

Other blob: `decode()`, `encode()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unbin()`, `unhex()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`from_json` *DuckDB function from_json*

Description

DuckDB function `from_json()`.

Usage

```
from_json(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR`

Value

`ANY`

Provided by

The `json` extension (`LOAD json;`).

`from_json_strict` *DuckDB function from_json_strict*

Description

DuckDB function `from_json_strict()`.

Usage

```
from_json_strict(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR`

Value

`ANY`

Provided by

The `json` extension (`LOAD json;`).

functions *DuckDB function functions*

Description

DuckDB function `functions()`.

Usage

```
functions()
```

Value

Unspecified.

gamma *DuckDB function gamma*

Description

Interpolation of (x-1) factorial (so decimal inputs are allowed).

Usage

```
gamma(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
gamma(5.5)
```

`generate_series` *DuckDB function generate_series*

Description

Creates a list of values between `start` and `stop` - the stop parameter is inclusive.

Usage

```
generate_series(start, stop, step)
```

Arguments

`start`	`BIGINT | TIMESTAMP | TIMESTAMP WITH TIME ZONE`
`stop`	`BIGINT | TIMESTAMP | TIMESTAMP WITH TIME ZONE`
`step`	`BIGINT | INTERVAL`

Value

`BIGINT[] | TIMESTAMP[] | TIMESTAMP WITH TIME ZONE[]`

Overloads

- `generate_series(col0 = BIGINT)`
- `generate_series(col0 = BIGINT, col1 = BIGINT)`
- `generate_series(col0 = BIGINT, col1 = BIGINT, col2 = BIGINT)`
- `generate_series(col0 = TIMESTAMP, col1 = TIMESTAMP, col2 = INTERVAL)`
- `generate_series(col0 = `TIMESTAMP WITH TIME ZONE`, col1 = `TIMESTAMP WITH TIME ZONE`, col2 = INTERVAL)`
- `generate_series(start = BIGINT)`
- `generate_series(start = BIGINT, stop = BIGINT)`
- `generate_series(start = BIGINT, stop = BIGINT, step = BIGINT)`
- `generate_series(start = TIMESTAMP, stop = TIMESTAMP, step = INTERVAL)`
- `generate_series(start = `TIMESTAMP WITH TIME ZONE`, stop = `TIMESTAMP WITH TIME ZONE`, step = INTERVAL)`

Provided by

- `generate_series(col0 = BIGINT)`: built in
- `generate_series(col0 = BIGINT, col1 = BIGINT)`: built in
- `generate_series(col0 = BIGINT, col1 = BIGINT, col2 = BIGINT)`: built in
- `generate_series(col0 = TIMESTAMP, col1 = TIMESTAMP, col2 = INTERVAL)`: built in
- `generate_series(col0 = `TIMESTAMP WITH TIME ZONE`, col1 = `TIMESTAMP WITH TIME ZONE`, col2 = INTERVAL)`: `icu`

- `generate_series(start = BIGINT)`: built in
- `generate_series(start = BIGINT, stop = BIGINT)`: built in
- `generate_series(start = BIGINT, stop = BIGINT, step = BIGINT)`: built in
- `generate_series(start = TIMESTAMP, stop = TIMESTAMP, step = INTERVAL)`: built in
- `generate_series(start = \`TIMESTAMP WITH TIME ZONE\`, stop = \`TIMESTAMP WITH TIME ZONE\`, step = INTERVAL)`: built in

SQL examples

```
generate_series(2, 5, 3)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`generate_subscripts` *DuckDB function generate_subscripts*

Description

DuckDB macro `generate_subscripts()`.

Usage

```
generate_subscripts(arr, dim)
```

Arguments

`arr`, `dim`	Unspecified.

Value

Unspecified.

`geomean` *DuckDB function geomean*

Description

DuckDB macro `geomean()`.

Usage

```
geomean(x)
```

Arguments

`x`	Unspecified.

Value

Unspecified.

`geometric_mean` *DuckDB function geometric_mean*

Description

DuckDB macro `geometric_mean()`.

Usage

```
geometric_mean(x)
```

Arguments

`x`	Unspecified.

Value

Unspecified.

`get_bit` *DuckDB function get_bit*

Description

Extracts the nth bit from bitstring; the first (leftmost) bit is indexed 0.

Usage

```
get_bit(bitstring, index)
```

Arguments

`bitstring`	`BIT`
`index`	`INTEGER`

Value

`INTEGER`

SQL examples

```
get_bit('0110010'::BIT, 2)
```

`get_block_size` *DuckDB function get_block_size*

Description

DuckDB macro `get_block_size()`.

Usage

```
get_block_size(db_name)
```

Arguments

`db_name`	Unspecified.

Value

Unspecified.

get_current_time *DuckDB function get_current_time*

Description

DuckDB function `get_current_time()`.

Usage

```
get_current_time()
```

Value

`TIME WITH TIME ZONE`

Provided by

The `icu` extension (`LOAD icu;`).

get_current_timestamp *DuckDB function get_current_timestamp*

Description

Returns the current timestamp.

Usage

```
get_current_timestamp()
```

Value

`TIMESTAMP WITH TIME ZONE`

SQL examples

```
get_current_timestamp()
```

`get_delta_test_expression` *DuckDB function get_delta_test_expression*

Description

DuckDB function `get_delta_test_expression()`.

Usage

```
get_delta_test_expression()
```

Value

`VARCHAR[]`

Provided by

The `delta` extension (`LOAD delta;`).

`get_type` *DuckDB function get_type*

Description

Returns the type of the result of the expression.

Usage

```
get_type(expression)
```

Arguments

`expression`	`ANY`

Value

`TYPE`

SQL examples

```
get_type('abc')
```

`get_ui_url` *DuckDB function get_ui_url*

Description

DuckDB function `get_ui_url()`.

Usage

```
get_ui_url()
```

Value

Unspecified.

Provided by

The `ui` extension (`LOAD ui;`).

`getvariable` *DuckDB function getvariable*

Description

DuckDB function `getvariable()`.

Usage

```
getvariable(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`ANY`

`glob` *DuckDB function glob*

Description

DuckDB function `glob()`.

Usage

```
glob(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

`greatest` *DuckDB function greatest*

Description

Returns the largest value. For strings lexicographical ordering is used. Note that lowercase characters are considered "larger" than uppercase characters and collations are not supported.

Usage

```
greatest(arg1)
```

Arguments

`arg1`	`ANY`

Value

`ANY`

SQL examples

```
greatest(42, 84)
greatest('abc', 'bcd', 'cde', 'EFG')
```

See Also

Other aggregate: `least()`

Other date: `least()`

Other numeric: `bin()`, `bit_length()`, `formatReadableDecimalSize()`, `format_bytes()`, `hex()`, `least()`, `len()`, `parse_formatted_bytes()`, `to_base()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

Other timestamp: `least()`

---

greatest_common_divisor

DuckDB function greatest_common_divisor

Description

Computes the greatest common divisor of x and y.

Usage

```
greatest_common_divisor(x, y)
```

Arguments

x, y	BIGINT | HUGEINT

Value

```
BIGINT | HUGEINT
```

SQL examples

```
greatest_common_divisor(42, 57)
```

hamming *DuckDB function hamming*

Description

The Hamming distance between to strings, i.e., the number of positions with different characters for two strings of equal length. Strings must be of equal length. Characters of different cases (e.g., `a` and `A`) are considered different.

Usage

```
hamming(s1, s2)
```

Arguments

`s1`, `s2` VARCHAR

Value

`BIGINT`

SQL examples

```
hamming('duck', 'luck')
```

See Also

Other text_similarity: `damerau_levenshtein()`, `jaccard()`, `jaro_similarity()`, `jaro_winkler_similarity()`, `levenshtein()`

has_any_column_privilege *DuckDB function has_any_column_privilege*

Description

DuckDB function `has_any_column_privilege()`.

Usage

```
has_any_column_privilege(user, table, privilege)
```

Arguments

`user`, `table`, `privilege`
Unspecified.

Value

Unspecified.

Overloads

- `has_any_column_privilege(table, privilege)`
- `has_any_column_privilege(user, table, privilege)`

`has_column_privilege` *DuckDB function has_column_privilege*

Description

DuckDB function `has_column_privilege()`.

Usage

```
has_column_privilege(user, table, column, privilege)
```

Arguments

`user`, `table`, `column`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_column_privilege(table, column, privilege)`
- `has_column_privilege(user, table, column, privilege)`

`has_database_privilege` *DuckDB function has_database_privilege*

Description

DuckDB function `has_database_privilege()`.

Usage

```
has_database_privilege(user, database, privilege)
```

Arguments

`user`, `database`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_database_privilege(database, privilege)`
- `has_database_privilege(user, database, privilege)`

`has_foreign_data_wrapper_privilege`

DuckDB function has_foreign_data_wrapper_privilege

Description

DuckDB function `has_foreign_data_wrapper_privilege()`.

Usage

```
has_foreign_data_wrapper_privilege(user, fdw, privilege)
```

Arguments

`user`, `fdw`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_foreign_data_wrapper_privilege(fdw, privilege)`
- `has_foreign_data_wrapper_privilege(user, fdw, privilege)`

`has_function_privilege`
DuckDB function has_function_privilege

Description

DuckDB function `has_function_privilege()`.

Usage

```
has_function_privilege(user, function., privilege)
```

Arguments

`user`, `function.`, `privilege`
Unspecified.

Value

Unspecified.

Overloads

- `has_function_privilege(function., privilege)`
- `has_function_privilege(user, function., privilege)`

`has_language_privilege`
DuckDB function has_language_privilege

Description

DuckDB function `has_language_privilege()`.

Usage

```
has_language_privilege(user, language, privilege)
```

Arguments

`user`, `language`, `privilege`
Unspecified.

Value

Unspecified.

Overloads

- `has_language_privilege(language, privilege)`
- `has_language_privilege(user, language, privilege)`

`has_schema_privilege` *DuckDB function has_schema_privilege*

Description

DuckDB function `has_schema_privilege()`.

Usage

```
has_schema_privilege(user, schema, privilege)
```

Arguments

`user`, `schema`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_schema_privilege(schema, privilege)`
- `has_schema_privilege(user, schema, privilege)`

`has_sequence_privilege`
DuckDB function has_sequence_privilege

Description

DuckDB function `has_sequence_privilege()`.

Usage

```
has_sequence_privilege(user, sequence, privilege)
```

Arguments

`user`, `sequence`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_sequence_privilege(sequence, privilege)`
- `has_sequence_privilege(user, sequence, privilege)`

`has_server_privilege` *DuckDB function has_server_privilege*

Description

DuckDB function `has_server_privilege()`.

Usage

```
has_server_privilege(user, server, privilege)
```

Arguments

`user`, `server`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_server_privilege(server, privilege)`
- `has_server_privilege(user, server, privilege)`

`has_table_privilege` *DuckDB function has_table_privilege*

Description

DuckDB function `has_table_privilege()`.

Usage

```
has_table_privilege(user, table, privilege)
```

Arguments

`user`, `table`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_table_privilege(table, privilege)`
- `has_table_privilege(user, table, privilege)`

`has_tablespace_privilege`

DuckDB function has_tablespace_privilege

Description

DuckDB function `has_tablespace_privilege()`.

Usage

```
has_tablespace_privilege(user, tablespace, privilege)
```

Arguments

`user`, `tablespace`, `privilege`
: Unspecified.

Value

Unspecified.

Overloads

- `has_tablespace_privilege(tablespace, privilege)`
- `has_tablespace_privilege(user, tablespace, privilege)`

`hash` *DuckDB function hash*

Description

Returns a `UBIGINT` with the hash of the `value`. Note that this is not a cryptographic hash.

Usage

```
hash(value)
```

Arguments

`value`	`ANY`

Value

`UBIGINT`

SQL examples

```
hash(' ')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`hex` *DuckDB function hex*

Description

- `hex(string = VARCHAR)`: Converts the `string` to hexadecimal representation.
- `hex(value = BIGNUM)`, `hex(value = BIGINT)`, `hex(value = UBIGINT)`, `hex(value = HUGEINT)`, `hex(value = UHUGEINT)`: Converts the `value` to `VARCHAR` using hexadecimal representation.
- `hex(blob = BLOB)`: Converts `blob` to `VARCHAR` using hexadecimal encoding.

Usage

```
hex(value)
```

Arguments

value	BIGNUM | BIGINT | UBIGINT | HUGEINT | UHUGEINT

Value

VARCHAR

Overloads

- hex(string = VARCHAR)
- hex(value = BIGNUM)
- hex(blob = BLOB)
- hex(value = BIGINT)
- hex(value = UBIGINT)
- hex(value = HUGEINT)
- hex(value = UHUGEINT)

SQL examples

```
hex('Hello')
hex(42)
hex('\xAA\xBB'::BLOB)
```

See Also

Other blob: decode(), encode(), from_base64(), md5(), md5_number(), octet_length(), or--or, repeat(), sha1(), sha256(), to_base64(), unbin(), unhex()

Other numeric: bin(), bit_length(), formatReadableDecimalSize(), format_bytes(), greatest(), least(), len(), parse_formatted_bytes(), to_base()

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`histogram` *DuckDB function histogram*

Description

Returns a LIST of STRUCTs with the fields bucket and count.

Usage

```
histogram(source, col_name, bin_count, technique)
```

Arguments

`source`, `col_name`, `bin_count`, `technique`
: Unspecified.

Value

`MAP`

Overloads

- `histogram(arg = ANY, col1 = `ANY[]`)`
- `histogram(arg = ANY)`
- `histogram(source, col_name, bin_count, technique)`

SQL examples

```
histogram(A)
```

`histogram_exact` *DuckDB function histogram_exact*

Description

Returns a LIST of STRUCTs with the fields bucket and count matching the buckets exactly.

Usage

```
histogram_exact(arg, bins)
```

Arguments

`arg`	`ANY`
`bins`	`ANY[]`

Value

MAP

SQL examples

```
histogram_exact(A, [0, 1, 2])
```

---

histogram_values *DuckDB function histogram_values*

Description

DuckDB macro `histogram_values()`.

Usage

```
histogram_values(source, col_name, bin_count, technique)
```

Arguments

`source`, `col_name`, `bin_count`, `technique`
: Unspecified.

Value

Unspecified.

---

hnsw_compact_index *DuckDB function hnsw_compact_index*

Description

DuckDB function `hnsw_compact_index()`.

Usage

```
hnsw_compact_index(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `vss` extension (`LOAD vss;`).

`hnsw_index_scan` *DuckDB function hnsw_index_scan*

Description

DuckDB function `hnsw_index_scan()`.

Usage

```
hnsw_index_scan()
```

Value

Unspecified.

Provided by

The `vss` extension (`LOAD vss;`).

`host` *DuckDB function host*

Description

DuckDB function `host()`.

Usage

```
host(col0)
```

Arguments

`col0`	`INET`

Value

`VARCHAR`

Provided by

The `inet` extension (`LOAD inet;`).

`hour` *DuckDB function hour*

Description

Extract the hour component from a date or timestamp.

Usage

```
hour(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W`

Value

`BIGINT`

SQL examples

```
hour(timestamp '2021-08-03 11:59:44.123456')
```

`html_escape` *DuckDB function html_escape*

Description

DuckDB function `html_escape()`.

Usage

```
html_escape(col0, col1)
```

Arguments

`col0`	`VARCHAR`
`col1`	`BOOLEAN`

Value

`VARCHAR`

Overloads

- `html_escape(col0 = VARCHAR)`
- `html_escape(col0 = VARCHAR, col1 = BOOLEAN)`

Provided by

The `inet` extension (`LOAD inet;`).

`html_unescape` *DuckDB function html_unescape*

Description

DuckDB function `html_unescape()`.

Usage

```
html_unescape(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `inet` extension (`LOAD inet;`).

`iceberg_bucket` *DuckDB function iceberg_bucket*

Description

DuckDB function `iceberg_bucket()`.

Usage

```
iceberg_bucket(col0, col1)
```

Arguments

`col0`	`INTEGER`
`col1`	`INTEGER | BIGINT | VARCHAR | BLOB | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`INTEGER`

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_column_stats` *DuckDB function iceberg_column_stats*

Description

DuckDB function `iceberg_column_stats()`.

Usage

```
iceberg_column_stats(
  col0,
  snapshot_from_timestamp,
  snapshot_from_id,
  version_name_format,
  version,
  metadata_compression_codec,
  allow_moved_paths
)
```

Arguments

```
col0            VARCHAR
snapshot_from_timestamp
                TIMESTAMP
snapshot_from_id
                UBIGINT
version_name_format
                VARCHAR
version         VARCHAR
metadata_compression_codec
                VARCHAR
allow_moved_paths
                BOOLEAN
```

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

iceberg_load_table_response

DuckDB function iceberg_load_table_response

Description

DuckDB function `iceberg_load_table_response()`.

Usage

```
iceberg_load_table_response(col0)
```

Arguments

col0	VARCHAR

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

iceberg_metadata

DuckDB function iceberg_metadata

Description

DuckDB function `iceberg_metadata()`.

Usage

```
iceberg_metadata(
  col0,
  snapshot_from_timestamp,
  snapshot_from_id,
  version_name_format,
  version,
  metadata_compression_codec,
  allow_moved_paths
)
```

Arguments

```
col0            VARCHAR
snapshot_from_timestamp
                TIMESTAMP
snapshot_from_id
                UBIGINT
version_name_format
                VARCHAR
version         VARCHAR
metadata_compression_codec
                VARCHAR
allow_moved_paths
                BOOLEAN
```

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_partition_stats`

DuckDB function iceberg__partition__stats

Description

DuckDB function `iceberg_partition_stats()`.

Usage

```
iceberg_partition_stats(
  col0,
  snapshot_from_timestamp,
  snapshot_from_id,
  version_name_format,
  version,
  metadata_compression_codec,
  allow_moved_paths
)
```

Arguments

```
col0            VARCHAR
snapshot_from_timestamp
                TIMESTAMP
snapshot_from_id
                UBIGINT
version_name_format
                VARCHAR
version         VARCHAR
metadata_compression_codec
                VARCHAR
allow_moved_paths
                BOOLEAN
```

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_scan` *DuckDB function iceberg_scan*

Description

DuckDB function `iceberg_scan()`.

Usage

```
iceberg_scan(
  col0,
  snapshot_from_timestamp,
  version,
  metadata_compression_codec,
  mode,
  can_have_nan,
  allow_moved_paths,
  filename,
  union_by_name,
  snapshot_from_id,
  debug_use_openssl,
  hive_partitioning,
  parquet_version,
  encryption_config,
  version_name_format,
```

```
  hive_types_autocast,
  binary_as_string,
  explicit_cardinality,
  compression,
  file_row_number,
  hive_types
)
```

Arguments

```
col0            VARCHAR | VARCHAR[]
snapshot_from_timestamp
                TIMESTAMP
version         VARCHAR
metadata_compression_codec
                VARCHAR
mode            VARCHAR
can_have_nan    BOOLEAN
allow_moved_paths
                BOOLEAN
filename        ANY
union_by_name   BOOLEAN
snapshot_from_id
                UBIGINT
debug_use_openssl
                BOOLEAN
hive_partitioning
                BOOLEAN
parquet_version
                VARCHAR
encryption_config
                ANY
version_name_format
                VARCHAR
hive_types_autocast
                BOOLEAN
binary_as_string
                BOOLEAN
explicit_cardinality
                UBIGINT
compression     VARCHAR
file_row_number
                BOOLEAN
hive_types      ANY
```

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_schema_properties`
DuckDB function iceberg_schema_properties

Description

DuckDB function `iceberg_schema_properties()`.

Usage

```
iceberg_schema_properties(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_snapshots` *DuckDB function iceberg_snapshots*

Description

DuckDB function `iceberg_snapshots()`.

Usage

```
iceberg_snapshots(col0, version_name_format, version, metadata_compression_codec)
```

Arguments

`col0`, `version_name_format`, `version`, `metadata_compression_codec`	`VARCHAR`

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_table_properties` *DuckDB function iceberg_table_properties*

Description

DuckDB function `iceberg_table_properties()`.

Usage

```
iceberg_table_properties(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_to_ducklake` *DuckDB function iceberg_to_ducklake*

Description

DuckDB function `iceberg_to_ducklake()`.

Usage

```
iceberg_to_ducklake(col0, col1, skip_tables)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`skip_tables`	`VARCHAR[]`

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_truncate` *DuckDB function iceberg_truncate*

Description

DuckDB function `iceberg_truncate()`.

Usage

```
iceberg_truncate(col0, col1)
```

Arguments

`col0`	`INTEGER`
`col1`	`INTEGER | BIGINT | VARCHAR | BLOB | DECIMAL`

Value

`INTEGER | BIGINT | VARCHAR | BLOB | DECIMAL`

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`iceberg_verify_equality_deletes`
DuckDB function iceberg_verify_equality_deletes

Description

DuckDB function `iceberg_verify_equality_deletes()`.

Usage

```
iceberg_verify_equality_deletes()
```

Value

`BOOLEAN`

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`icu_calendar_names` *DuckDB function icu_calendar_names*

Description

DuckDB function `icu_calendar_names()`.

Usage

```
icu_calendar_names()
```

Value

Unspecified.

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_af` *DuckDB function icu_collate_af*

Description

DuckDB function `icu_collate_af()`.

Usage

```
icu_collate_af(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_am` *DuckDB function icu_collate_am*

Description

DuckDB function `icu_collate_am()`.

Usage

```
icu_collate_am(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ar` *DuckDB function icu_collate_ar*

Description

DuckDB function `icu_collate_ar()`.

Usage

```
icu_collate_ar(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ar_sa *DuckDB function icu_collate_ar_sa*

Description

DuckDB function `icu_collate_ar_sa()`.

Usage

```
icu_collate_ar_sa(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_as *DuckDB function icu_collate_as*

Description

DuckDB function `icu_collate_as()`.

Usage

```
icu_collate_as(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_az` *DuckDB function icu_collate_az*

Description

DuckDB function `icu_collate_az()`.

Usage

```
icu_collate_az(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_be` *DuckDB function icu_collate_be*

Description

DuckDB function `icu_collate_be()`.

Usage

```
icu_collate_be(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_bg` *DuckDB function icu_collate_bg*

Description

DuckDB function `icu_collate_bg()`.

Usage

```
icu_collate_bg(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_bn` *DuckDB function icu_collate_bn*

Description

DuckDB function `icu_collate_bn()`.

Usage

```
icu_collate_bn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_bo` *DuckDB function icu_collate_bo*

Description

DuckDB function `icu_collate_bo()`.

Usage

```
icu_collate_bo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_br` *DuckDB function icu_collate_br*

Description

DuckDB function `icu_collate_br()`.

Usage

```
icu_collate_br(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_bs *DuckDB function icu_collate_bs*

Description

DuckDB function `icu_collate_bs()`.

Usage

```
icu_collate_bs(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ca *DuckDB function icu_collate_ca*

Description

DuckDB function `icu_collate_ca()`.

Usage

```
icu_collate_ca(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ceb` *DuckDB function icu_collate_ceb*

Description

DuckDB function `icu_collate_ceb()`.

Usage

```
icu_collate_ceb(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_chr` *DuckDB function icu_collate_chr*

Description

DuckDB function `icu_collate_chr()`.

Usage

```
icu_collate_chr(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_cs *DuckDB function icu_collate_cs*

Description

DuckDB function `icu_collate_cs()`.

Usage

```
icu_collate_cs(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_cy *DuckDB function icu_collate_cy*

Description

DuckDB function `icu_collate_cy()`.

Usage

```
icu_collate_cy(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_da` *DuckDB function icu_collate_da*

Description

DuckDB function `icu_collate_da()`.

Usage

```
icu_collate_da(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_de` *DuckDB function icu_collate_de*

Description

DuckDB function `icu_collate_de()`.

Usage

```
icu_collate_de(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_de_at *DuckDB function icu_collate_de_at*

Description

DuckDB function `icu_collate_de_at()`.

Usage

```
icu_collate_de_at(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_dsb *DuckDB function icu_collate_dsb*

Description

DuckDB function `icu_collate_dsb()`.

Usage

```
icu_collate_dsb(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_dz` *DuckDB function icu_collate_dz*

Description

DuckDB function `icu_collate_dz()`.

Usage

```
icu_collate_dz(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ee` *DuckDB function icu_collate_ee*

Description

DuckDB function `icu_collate_ee()`.

Usage

```
icu_collate_ee(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_el *DuckDB function icu_collate_el*

Description

DuckDB function `icu_collate_el()`.

Usage

```
icu_collate_el(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_en *DuckDB function icu_collate_en*

Description

DuckDB function `icu_collate_en()`.

Usage

```
icu_collate_en(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_en_us` *DuckDB function icu_collate_en_us*

Description

DuckDB function `icu_collate_en_us()`.

Usage

```
icu_collate_en_us(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_eo` *DuckDB function icu_collate_eo*

Description

DuckDB function `icu_collate_eo()`.

Usage

```
icu_collate_eo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_es *DuckDB function icu_collate_es*

Description

DuckDB function `icu_collate_es()`.

Usage

```
icu_collate_es(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_et *DuckDB function icu_collate_et*

Description

DuckDB function `icu_collate_et()`.

Usage

```
icu_collate_et(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_fa` *DuckDB function icu_collate_fa*

Description

DuckDB function `icu_collate_fa()`.

Usage

```
icu_collate_fa(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_fa_af` *DuckDB function icu_collate_fa_af*

Description

DuckDB function `icu_collate_fa_af()`.

Usage

```
icu_collate_fa_af(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ff *DuckDB function icu_collate_ff*

Description

DuckDB function `icu_collate_ff()`.

Usage

```
icu_collate_ff(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_fi *DuckDB function icu_collate_fi*

Description

DuckDB function `icu_collate_fi()`.

Usage

```
icu_collate_fi(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_fil` *DuckDB function icu_collate_fil*

Description

DuckDB function `icu_collate_fil()`.

Usage

```
icu_collate_fil(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_fo` *DuckDB function icu_collate_fo*

Description

DuckDB function `icu_collate_fo()`.

Usage

```
icu_collate_fo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_fr *DuckDB function icu_collate_fr*

Description

DuckDB function `icu_collate_fr()`.

Usage

```
icu_collate_fr(col0)
```

Arguments

col0	VARCHAR

Value

VARCHAR

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_fr_ca *DuckDB function icu_collate_fr_ca*

Description

DuckDB function `icu_collate_fr_ca()`.

Usage

```
icu_collate_fr_ca(col0)
```

Arguments

col0	VARCHAR

Value

VARCHAR

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_fy` *DuckDB function icu_collate_fy*

Description

DuckDB function `icu_collate_fy()`.

Usage

```
icu_collate_fy(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ga` *DuckDB function icu_collate_ga*

Description

DuckDB function `icu_collate_ga()`.

Usage

```
icu_collate_ga(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_gl` *DuckDB function icu_collate_gl*

Description

DuckDB function `icu_collate_gl()`.

Usage

```
icu_collate_gl(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_gu` *DuckDB function icu_collate_gu*

Description

DuckDB function `icu_collate_gu()`.

Usage

```
icu_collate_gu(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ha` *DuckDB function icu_collate_ha*

Description

DuckDB function `icu_collate_ha()`.

Usage

```
icu_collate_ha(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_haw` *DuckDB function icu_collate_haw*

Description

DuckDB function `icu_collate_haw()`.

Usage

```
icu_collate_haw(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_he *DuckDB function icu_collate_he*

Description

DuckDB function `icu_collate_he()`.

Usage

```
icu_collate_he(col0)
```

Arguments

col0	VARCHAR

Value

VARCHAR

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_he_il *DuckDB function icu_collate_he_il*

Description

DuckDB function `icu_collate_he_il()`.

Usage

```
icu_collate_he_il(col0)
```

Arguments

col0	VARCHAR

Value

VARCHAR

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_hi` *DuckDB function icu_collate_hi*

Description

DuckDB function `icu_collate_hi()`.

Usage

```
icu_collate_hi(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_hr` *DuckDB function icu_collate_hr*

Description

DuckDB function `icu_collate_hr()`.

Usage

```
icu_collate_hr(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_hsb` *DuckDB function icu_collate_hsb*

Description

DuckDB function `icu_collate_hsb()`.

Usage

```
icu_collate_hsb(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_hu` *DuckDB function icu_collate_hu*

Description

DuckDB function `icu_collate_hu()`.

Usage

```
icu_collate_hu(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_hy` *DuckDB function icu_collate_hy*

Description

DuckDB function `icu_collate_hy()`.

Usage

```
icu_collate_hy(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_id` *DuckDB function icu_collate_id*

Description

DuckDB function `icu_collate_id()`.

Usage

```
icu_collate_id(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_id_id *DuckDB function icu_collate_id_id*

Description

DuckDB function `icu_collate_id_id()`.

Usage

```
icu_collate_id_id(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ig *DuckDB function icu_collate_ig*

Description

DuckDB function `icu_collate_ig()`.

Usage

```
icu_collate_ig(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_is` *DuckDB function icu_collate_is*

Description

DuckDB function `icu_collate_is()`.

Usage

```
icu_collate_is(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_it` *DuckDB function icu_collate_it*

Description

DuckDB function `icu_collate_it()`.

Usage

```
icu_collate_it(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ja *DuckDB function icu_collate_ja*

Description

DuckDB function `icu_collate_ja()`.

Usage

```
icu_collate_ja(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ka *DuckDB function icu_collate_ka*

Description

DuckDB function `icu_collate_ka()`.

Usage

```
icu_collate_ka(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_kk` *DuckDB function icu_collate_kk*

Description

DuckDB function `icu_collate_kk()`.

Usage

```
icu_collate_kk(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_kl` *DuckDB function icu_collate_kl*

Description

DuckDB function `icu_collate_kl()`.

Usage

```
icu_collate_kl(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_km` *DuckDB function icu_collate_km*

Description

DuckDB function `icu_collate_km()`.

Usage

```
icu_collate_km(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_kn` *DuckDB function icu_collate_kn*

Description

DuckDB function `icu_collate_kn()`.

Usage

```
icu_collate_kn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ko` *DuckDB function icu_collate_ko*

Description

DuckDB function `icu_collate_ko()`.

Usage

```
icu_collate_ko(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_kok` *DuckDB function icu_collate_kok*

Description

DuckDB function `icu_collate_kok()`.

Usage

```
icu_collate_kok(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ku` *DuckDB function icu_collate_ku*

Description

DuckDB function `icu_collate_ku()`.

Usage

```
icu_collate_ku(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ky` *DuckDB function icu_collate_ky*

Description

DuckDB function `icu_collate_ky()`.

Usage

```
icu_collate_ky(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_lb` *DuckDB function icu_collate_lb*

Description

DuckDB function `icu_collate_lb()`.

Usage

```
icu_collate_lb(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_lij` *DuckDB function icu_collate_lij*

Description

DuckDB function `icu_collate_lij()`.

Usage

```
icu_collate_lij(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_lkt *DuckDB function icu_collate_lkt*

Description

DuckDB function `icu_collate_lkt()`.

Usage

```
icu_collate_lkt(col0)
```

Arguments

col0	VARCHAR

Value

VARCHAR

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_ln *DuckDB function icu_collate_ln*

Description

DuckDB function `icu_collate_ln()`.

Usage

```
icu_collate_ln(col0)
```

Arguments

col0	VARCHAR

Value

VARCHAR

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_lo` *DuckDB function icu_collate_lo*

Description

DuckDB function `icu_collate_lo()`.

Usage

```
icu_collate_lo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_lt` *DuckDB function icu_collate_lt*

Description

DuckDB function `icu_collate_lt()`.

Usage

```
icu_collate_lt(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_lv *DuckDB function icu_collate_lv*

Description

DuckDB function `icu_collate_lv()`.

Usage

```
icu_collate_lv(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_mk *DuckDB function icu_collate_mk*

Description

DuckDB function `icu_collate_mk()`.

Usage

```
icu_collate_mk(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ml` *DuckDB function icu_collate_ml*

Description

DuckDB function `icu_collate_ml()`.

Usage

```
icu_collate_ml(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_mn` *DuckDB function icu_collate_mn*

Description

DuckDB function `icu_collate_mn()`.

Usage

```
icu_collate_mn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_mr` *DuckDB function icu_collate_mr*

Description

DuckDB function `icu_collate_mr()`.

Usage

```
icu_collate_mr(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ms` *DuckDB function icu_collate_ms*

Description

DuckDB function `icu_collate_ms()`.

Usage

```
icu_collate_ms(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_mt` *DuckDB function icu_collate_mt*

Description

DuckDB function `icu_collate_mt()`.

Usage

```
icu_collate_mt(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_my` *DuckDB function icu_collate_my*

Description

DuckDB function `icu_collate_my()`.

Usage

```
icu_collate_my(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_nb *DuckDB function icu_collate_nb*

Description

DuckDB function `icu_collate_nb()`.

Usage

```
icu_collate_nb(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_nb_no *DuckDB function icu_collate_nb_no*

Description

DuckDB function `icu_collate_nb_no()`.

Usage

```
icu_collate_nb_no(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ne` *DuckDB function icu_collate_ne*

Description

DuckDB function `icu_collate_ne()`.

Usage

```
icu_collate_ne(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_nl` *DuckDB function icu_collate_nl*

Description

DuckDB function `icu_collate_nl()`.

Usage

```
icu_collate_nl(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_nn *DuckDB function icu_collate_nn*

Description

DuckDB function `icu_collate_nn()`.

Usage

```
icu_collate_nn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_noaccent *DuckDB function icu_collate_noaccent*

Description

DuckDB function `icu_collate_noaccent()`.

Usage

```
icu_collate_noaccent(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_nso` *DuckDB function icu_collate_nso*

Description

DuckDB function `icu_collate_nso()`.

Usage

```
icu_collate_nso(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_om` *DuckDB function icu_collate_om*

Description

DuckDB function `icu_collate_om()`.

Usage

```
icu_collate_om(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_or *DuckDB function icu_collate_or*

Description

DuckDB function `icu_collate_or()`.

Usage

```
icu_collate_or(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_pa *DuckDB function icu_collate_pa*

Description

DuckDB function `icu_collate_pa()`.

Usage

```
icu_collate_pa(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_pa_in` *DuckDB function icu_collate_pa_in*

Description

DuckDB function `icu_collate_pa_in()`.

Usage

```
icu_collate_pa_in(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_pl` *DuckDB function icu_collate_pl*

Description

DuckDB function `icu_collate_pl()`.

Usage

```
icu_collate_pl(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ps` *DuckDB function icu_collate_ps*

Description

DuckDB function `icu_collate_ps()`.

Usage

```
icu_collate_ps(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_pt` *DuckDB function icu_collate_pt*

Description

DuckDB function `icu_collate_pt()`.

Usage

```
icu_collate_pt(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ro` *DuckDB function icu_collate_ro*

Description

DuckDB function `icu_collate_ro()`.

Usage

```
icu_collate_ro(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ru` *DuckDB function icu_collate_ru*

Description

DuckDB function `icu_collate_ru()`.

Usage

```
icu_collate_ru(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_sa *DuckDB function icu_collate_sa*

Description

DuckDB function `icu_collate_sa()`.

Usage

```
icu_collate_sa(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_se *DuckDB function icu_collate_se*

Description

DuckDB function `icu_collate_se()`.

Usage

```
icu_collate_se(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_si` *DuckDB function icu_collate_si*

Description

DuckDB function `icu_collate_si()`.

Usage

```
icu_collate_si(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sk` *DuckDB function icu_collate_sk*

Description

DuckDB function `icu_collate_sk()`.

Usage

```
icu_collate_sk(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sl` *DuckDB function icu_collate_sl*

Description

DuckDB function `icu_collate_sl()`.

Usage

```
icu_collate_sl(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_smn` *DuckDB function icu_collate_smn*

Description

DuckDB function `icu_collate_smn()`.

Usage

```
icu_collate_smn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sq` *DuckDB function icu_collate_sq*

Description

DuckDB function `icu_collate_sq()`.

Usage

```
icu_collate_sq(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sr` *DuckDB function icu_collate_sr*

Description

DuckDB function `icu_collate_sr()`.

Usage

```
icu_collate_sr(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sr_ba` *DuckDB function icu_collate_sr_ba*

Description

DuckDB function `icu_collate_sr_ba()`.

Usage

```
icu_collate_sr_ba(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sr_me` *DuckDB function icu_collate_sr_me*

Description

DuckDB function `icu_collate_sr_me()`.

Usage

```
icu_collate_sr_me(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_sr_rs` *DuckDB function icu_collate_sr_rs*

Description

DuckDB function `icu_collate_sr_rs()`.

Usage

```
icu_collate_sr_rs(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_st` *DuckDB function icu_collate_st*

Description

DuckDB function `icu_collate_st()`.

Usage

```
icu_collate_st(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_sv *DuckDB function icu_collate_sv*

Description

DuckDB function `icu_collate_sv()`.

Usage

```
icu_collate_sv(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

icu_collate_sw *DuckDB function icu_collate_sw*

Description

DuckDB function `icu_collate_sw()`.

Usage

```
icu_collate_sw(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ta` *DuckDB function icu_collate_ta*

Description

DuckDB function `icu_collate_ta()`.

Usage

```
icu_collate_ta(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_te` *DuckDB function icu_collate_te*

Description

DuckDB function `icu_collate_te()`.

Usage

```
icu_collate_te(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_th` *DuckDB function icu_collate_th*

Description

DuckDB function `icu_collate_th()`.

Usage

```
icu_collate_th(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_tk` *DuckDB function icu_collate_tk*

Description

DuckDB function `icu_collate_tk()`.

Usage

```
icu_collate_tk(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_tn` *DuckDB function icu_collate_tn*

Description

DuckDB function `icu_collate_tn()`.

Usage

```
icu_collate_tn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_to` *DuckDB function icu_collate_to*

Description

DuckDB function `icu_collate_to()`.

Usage

```
icu_collate_to(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_tr` *DuckDB function icu_collate_tr*

Description

DuckDB function `icu_collate_tr()`.

Usage

```
icu_collate_tr(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ug` *DuckDB function icu_collate_ug*

Description

DuckDB function `icu_collate_ug()`.

Usage

```
icu_collate_ug(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_uk` *DuckDB function icu_collate_uk*

Description

DuckDB function `icu_collate_uk()`.

Usage

```
icu_collate_uk(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_ur` *DuckDB function icu_collate_ur*

Description

DuckDB function `icu_collate_ur()`.

Usage

```
icu_collate_ur(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_uz` *DuckDB function icu_collate_uz*

Description

DuckDB function `icu_collate_uz()`.

Usage

```
icu_collate_uz(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_vi` *DuckDB function icu_collate_vi*

Description

DuckDB function `icu_collate_vi()`.

Usage

```
icu_collate_vi(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_wae` *DuckDB function icu_collate_wae*

Description

DuckDB function `icu_collate_wae()`.

Usage

```
icu_collate_wae(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_wo` *DuckDB function icu_collate_wo*

Description

DuckDB function `icu_collate_wo()`.

Usage

```
icu_collate_wo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_xh` *DuckDB function icu_collate_xh*

Description

DuckDB function `icu_collate_xh()`.

Usage

```
icu_collate_xh(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_yi` *DuckDB function icu_collate_yi*

Description

DuckDB function `icu_collate_yi()`.

Usage

```
icu_collate_yi(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_yo` *DuckDB function icu_collate_yo*

Description

DuckDB function `icu_collate_yo()`.

Usage

```
icu_collate_yo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_yue` *DuckDB function icu_collate_yue*

Description

DuckDB function `icu_collate_yue()`.

Usage

```
icu_collate_yue(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_yue_cn` *DuckDB function icu_collate_yue_cn*

Description

DuckDB function `icu_collate_yue_cn()`.

Usage

```
icu_collate_yue_cn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zh` *DuckDB function icu_collate_zh*

Description

DuckDB function `icu_collate_zh()`.

Usage

```
icu_collate_zh(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zh_cn` *DuckDB function icu_collate_zh_cn*

Description

DuckDB function `icu_collate_zh_cn()`.

Usage

```
icu_collate_zh_cn(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zh_hk` *DuckDB function icu_collate_zh_hk*

Description

DuckDB function `icu_collate_zh_hk()`.

Usage

```
icu_collate_zh_hk(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zh_mo` *DuckDB function icu_collate_zh_mo*

Description

DuckDB function `icu_collate_zh_mo()`.

Usage

```
icu_collate_zh_mo(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zh_sg` *DuckDB function icu_collate_zh_sg*

Description

DuckDB function `icu_collate_zh_sg()`.

Usage

```
icu_collate_zh_sg(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zh_tw` *DuckDB function icu_collate_zh_tw*

Description

DuckDB function `icu_collate_zh_tw()`.

Usage

```
icu_collate_zh_tw(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_collate_zu` *DuckDB function icu_collate_zu*

Description

DuckDB function `icu_collate_zu()`.

Usage

```
icu_collate_zu(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`icu_sort_key` *DuckDB function icu_sort_key*

Description

DuckDB function `icu_sort_key()`.

Usage

```
icu_sort_key(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `icu` extension (`LOAD icu;`).

`ilike_escape` *DuckDB function ilike_escape*

Description

Returns `true` if the `string` matches the `like_specifier` (see Pattern Matching) using case-insensitive matching. `escape_character` is used to search for wildcard characters in the `string`.

Usage

```
ilike_escape(string, like_specifier, escape_character)
```

Arguments

`string`, `like_specifier`, `escape_character`	`VARCHAR`

Value

`BOOLEAN`

SQL examples

```
ilike_escape('A%c', 'a$%C', '$')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

import_database *DuckDB function import_database*

Description

DuckDB function `import_database()`.

Usage

```
import_database(col0)
```

Arguments

col0	VARCHAR

Value

Unspecified.

---

in_search_path *DuckDB function in_search_path*

Description

Returns whether or not the database/schema are in the search path.

Usage

```
in_search_path(database_name, schema_name)
```

Arguments

database_name, schema_name	VARCHAR

Value

BOOLEAN

SQL examples

```
in_search_path('memory', 'main')
```

---

inet_client_addr *DuckDB function inet_client_addr*

Description

DuckDB macro `inet_client_addr()`.

Usage

```
inet_client_addr()
```

Value

Unspecified.

---

inet_client_port *DuckDB function inet_client_port*

Description

DuckDB macro `inet_client_port()`.

Usage

```
inet_client_port()
```

Value

Unspecified.

`inet_server_addr` *DuckDB function inet_server_addr*

Description

DuckDB macro `inet_server_addr()`.

Usage

```
inet_server_addr()
```

Value

Unspecified.

`inet_server_port` *DuckDB function inet_server_port*

Description

DuckDB macro `inet_server_port()`.

Usage

```
inet_server_port()
```

Value

Unspecified.

`instr` *DuckDB function instr*

Description

Returns location of first occurrence of `search_string` in `string`, counting from 1. Returns 0 if no match found.

Usage

```
instr(string, search_string)
```

Arguments

`string`, `search_string`
VARCHAR

Value

BIGINT

SQL examples

```
instr('test test', 'es')
position('b' IN 'abc')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

is_histogram_other_bin

DuckDB function is_histogram_other_bin

Description

Whether or not the provided value is the histogram "other" bin (used for values not belonging to any provided bin).

Usage

```
is_histogram_other_bin(val)
```

Arguments

val	ANY

Value

BOOLEAN

SQL examples

```
is_histogram_other_bin(v)
```

`isfinite` *DuckDB function isfinite*

Description

Returns true if the floating point value is finite, false otherwise.

Usage

```
isfinite(x)
```

Arguments

x	FLOAT | DOUBLE | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

BOOLEAN

SQL examples

```
isfinite(5.5)
```

`isinf` *DuckDB function isinf*

Description

Returns true if the floating point value is infinite, false otherwise.

Usage

```
isinf(x)
```

Arguments

x	FLOAT | DOUBLE | DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

BOOLEAN

SQL examples

```
isinf('Infinity'::float)
```

`isnan` *DuckDB function isnan*

Description

Returns true if the floating point value is not a number, false otherwise.

Usage

```
isnan(x)
```

Arguments

x	FLOAT | DOUBLE

Value

```
BOOLEAN
```

SQL examples

```
isnan('NaN'::FLOAT)
```

`isodow` *DuckDB function isodow*

Description

Extract the isodow component from a date or timestamp.

Usage

```
isodow(ts)
```

Arguments

ts	DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

```
BIGINT
```

SQL examples

```
isodow(timestamp '2021-08-03 11:59:44.123456')
```

`isoyear` *DuckDB function isoyear*

Description

Extract the isoyear component from a date or timestamp.

Usage

```
isoyear(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
isoyear(timestamp '2021-08-03 11:59:44.123456')
```

`jaccard` *DuckDB function jaccard*

Description

The Jaccard similarity between two strings. Characters of different cases (e.g., `a` and `A`) are considered different. Returns a number between 0 and 1.

Usage

```
jaccard(s1, s2)
```

Arguments

`s1`, `s2`	`VARCHAR`

Value

`DOUBLE`

SQL examples

```
jaccard('duck', 'luck')
```

See Also

Other text_similarity: damerau_levenshtein(), hamming(), jaro_similarity(), jaro_winkler_similarity(), levenshtein()

---

jaro_similarity *DuckDB function jaro_similarity*

Description

The Jaro similarity between two strings. Characters of different cases (e.g., a and A) are considered different. Returns a number between 0 and 1. For similarity < score_cutoff, 0 is returned instead. score_cutoff defaults to 0.

Usage

```
jaro_similarity(s1, s2, score_cutoff)
```

Arguments

s1	VARCHAR
s2	VARCHAR
score_cutoff	DOUBLE

Value

DOUBLE

Overloads

- jaro_similarity(s1 = VARCHAR, s2 = VARCHAR)
- jaro_similarity(s1 = VARCHAR, s2 = VARCHAR, score_cutoff = DOUBLE)

SQL examples

```
jaro_similarity('duck', 'duckdb')
```

See Also

Other text_similarity: damerau_levenshtein(), hamming(), jaccard(), jaro_winkler_similarity(), levenshtein()

`jaro_winkler_similarity`

DuckDB function jaro_winkler_similarity

Description

The Jaro-Winkler similarity between two strings. Characters of different cases (e.g., `a` and `A`) are considered different. Returns a number between 0 and 1. For similarity < `score_cutoff`, 0 is returned instead. `score_cutoff` defaults to 0.

Usage

```
jaro_winkler_similarity(s1, s2, score_cutoff)
```

Arguments

`s1`	`VARCHAR`
`s2`	`VARCHAR`
`score_cutoff`	`DOUBLE`

Value

`DOUBLE`

Overloads

- `jaro_winkler_similarity(s1 = VARCHAR, s2 = VARCHAR)`
- `jaro_winkler_similarity(s1 = VARCHAR, s2 = VARCHAR, score_cutoff = DOUBLE)`

SQL examples

```
jaro_winkler_similarity('duck', 'duckdb')
```

See Also

Other text_similarity: `damerau_levenshtein()`, `hamming()`, `jaccard()`, `jaro_similarity()`, `levenshtein()`

json *DuckDB function json*

Description

DuckDB macro `json()`.

Usage

```
json(x)
```

Arguments

`x`	Unspecified.

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

json_array *DuckDB function json_array*

Description

DuckDB function `json_array()`.

Usage

```
json_array()
```

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

`json_array_length` *DuckDB function json_array_length*

Description

DuckDB function `json_array_length()`.

Usage

```
json_array_length(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR | VARCHAR[]`

Value

`UBIGINT | UBIGINT[]`

Overloads

- `json_array_length(col0 = VARCHAR)`
- `json_array_length(col0 = VARCHAR, col1 = VARCHAR)`
- `json_array_length(col0 = VARCHAR, col1 = `VARCHAR[]`)`
- `json_array_length(col0 = JSON)`
- `json_array_length(col0 = JSON, col1 = VARCHAR)`
- `json_array_length(col0 = JSON, col1 = `VARCHAR[]`)`

Provided by

The `json` extension (`LOAD json;`).

`json_contains` *DuckDB function json_contains*

Description

DuckDB function `json_contains()`.

Usage

```
json_contains(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR | JSON`

Value

`BOOLEAN`

Provided by

The `json` extension (`LOAD json;`).

`json_deserialize_sql` *DuckDB function json_deserialize_sql*

Description

DuckDB function `json_deserialize_sql()`.

Usage

```
json_deserialize_sql(col0)
```

Arguments

`col0`	`JSON`

Value

`VARCHAR`

Provided by

The `json` extension (`LOAD json;`).

`json_each` *DuckDB function json_each*

Description

DuckDB function `json_each()`.

Usage

```
json_each(col0, col1)
```

Arguments

col0	VARCHAR | JSON
col1	VARCHAR

Value

Unspecified.

Overloads

- `json_each(col0 = VARCHAR)`
- `json_each(col0 = VARCHAR, col1 = VARCHAR)`
- `json_each(col0 = JSON)`
- `json_each(col0 = JSON, col1 = VARCHAR)`

Provided by

The `json` extension (`LOAD json;`).

`json_execute_serialized_sql`

DuckDB function json_execute_serialized_sql

Description

DuckDB function `json_execute_serialized_sql()`.

Usage

```
json_execute_serialized_sql(col0)
```

Arguments

col0	VARCHAR

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`json_exists` *DuckDB function json_exists*

Description

DuckDB function `json_exists()`.

Usage

```
json_exists(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR | VARCHAR[]`

Value

`BOOLEAN | BOOLEAN[]`

Provided by

The `json` extension (`LOAD json;`).

`json_extract` *DuckDB function json_extract*

Description

DuckDB function `json_extract()`.

Usage

```
json_extract(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`BIGINT | VARCHAR | VARCHAR[]`

Value

`JSON | JSON[]`

Provided by

The `json` extension (`LOAD json;`).

`json_extract_path` *DuckDB function json_extract_path*

Description

DuckDB function `json_extract_path()`.

Usage

```
json_extract_path(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`BIGINT | VARCHAR | VARCHAR[]`

Value

`JSON | JSON[]`

Provided by

The `json` extension (`LOAD json;`).

`json_extract_path_text` *DuckDB function json_extract_path_text*

Description

DuckDB function `json_extract_path_text()`.

Usage

```
json_extract_path_text(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`BIGINT | VARCHAR | VARCHAR[]`

Value

`VARCHAR | VARCHAR[]`

Provided by

The `json` extension (`LOAD json;`).

json_extract_string *DuckDB function json_extract_string*

Description

DuckDB function `json_extract_string()`.

Usage

```
json_extract_string(col0, col1)
```

Arguments

col0	VARCHAR | JSON
col1	BIGINT | VARCHAR | VARCHAR[]

Value

`VARCHAR | VARCHAR[]`

Provided by

The `json` extension (`LOAD json;`).

json_group_array *DuckDB function json_group_array*

Description

DuckDB macro `json_group_array()`.

Usage

```
json_group_array(x)
```

Arguments

x	Unspecified.

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`json_group_object` *DuckDB function json_group_object*

Description

DuckDB macro `json_group_object()`.

Usage

```
json_group_object(n, v)
```

Arguments

`n`, `v`	Unspecified.

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`json_group_structure` *DuckDB function json_group_structure*

Description

DuckDB macro `json_group_structure()`.

Usage

```
json_group_structure(x)
```

Arguments

`x`	Unspecified.

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`json_keys` *DuckDB function json_keys*

Description

DuckDB function `json_keys()`.

Usage

```
json_keys(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR | VARCHAR[]`

Value

`VARCHAR[] | VARCHAR[][]`

Overloads

- `json_keys(col0 = VARCHAR)`
- `json_keys(col0 = VARCHAR, col1 = VARCHAR)`
- `json_keys(col0 = VARCHAR, col1 = \`VARCHAR[]\`)`
- `json_keys(col0 = JSON)`
- `json_keys(col0 = JSON, col1 = VARCHAR)`
- `json_keys(col0 = JSON, col1 = \`VARCHAR[]\`)`

Provided by

The `json` extension (`LOAD json;`).

`json_merge_patch` *DuckDB function json_merge_patch*

Description

DuckDB function `json_merge_patch()`.

Usage

```
json_merge_patch(col0, col1)
```

Arguments

`col0`, `col1` `JSON`

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

`json_object` *DuckDB function json_object*

Description

DuckDB function `json_object()`.

Usage

```
json_object()
```

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

`json_pretty` *DuckDB function json_pretty*

Description

DuckDB function `json_pretty()`.

Usage

```
json_pretty(col0)
```

Arguments

`col0` `JSON`

Value

VARCHAR

Provided by

The json extension (LOAD json;).

---

json_quote *DuckDB function json_quote*

Description

DuckDB function json_quote().

Usage

```
json_quote()
```

Value

JSON

Provided by

The json extension (LOAD json;).

---

json_serialize_plan *DuckDB function json_serialize_plan*

Description

DuckDB function json_serialize_plan().

Usage

```
json_serialize_plan(col0, col1, col2, col3, col4)
```

Arguments

col0	VARCHAR
col1	BOOLEAN
col2	BOOLEAN
col3	BOOLEAN
col4	BOOLEAN

Value

`JSON`

Overloads

- `json_serialize_plan(col0 = VARCHAR)`
- `json_serialize_plan(col0 = VARCHAR, col1 = BOOLEAN)`
- `json_serialize_plan(col0 = VARCHAR, col1 = BOOLEAN, col2 = BOOLEAN)`
- `json_serialize_plan(col0 = VARCHAR, col1 = BOOLEAN, col2 = BOOLEAN, col3 = BOOLEAN)`
- `json_serialize_plan(col0 = VARCHAR, col1 = BOOLEAN, col2 = BOOLEAN, col3 = BOOLEAN, col4 = BOOLEAN)`

Provided by

The `json` extension (`LOAD json;`).

`json_serialize_sql` *DuckDB function json_serialize_sql*

Description

DuckDB function `json_serialize_sql()`.

Usage

```
json_serialize_sql(col0, col1, col2, col3, col4)
```

Arguments

`col0`	`VARCHAR`
`col1`	`BOOLEAN`
`col2`	`BOOLEAN`
`col3`	`BOOLEAN`
`col4`	`BOOLEAN`

Value

`JSON`

Overloads

- `json_serialize_sql(col0 = VARCHAR)`
- `json_serialize_sql(col0 = VARCHAR, col1 = BOOLEAN)`
- `json_serialize_sql(col0 = VARCHAR, col1 = BOOLEAN, col2 = BOOLEAN)`
- `json_serialize_sql(col0 = VARCHAR, col1 = BOOLEAN, col2 = BOOLEAN, col3 = BOOLEAN)`
- `json_serialize_sql(col0 = VARCHAR, col1 = BOOLEAN, col2 = BOOLEAN, col3 = BOOLEAN, col4 = BOOLEAN)`

Provided by

The `json` extension (`LOAD json;`).

`json_structure` *DuckDB function json_structure*

Description

DuckDB function `json_structure()`.

Usage

```
json_structure(col0)
```

Arguments

`col0`	`VARCHAR | JSON`

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

`json_transform` *DuckDB function json_transform*

Description

DuckDB function `json_transform()`.

Usage

```
json_transform(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR`

Value

`ANY`

Provided by

The `json` extension (`LOAD json;`).

`json_transform_strict` *DuckDB function json_transform_strict*

Description

DuckDB function `json_transform_strict()`.

Usage

```
json_transform_strict(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR`

Value

`ANY`

Provided by

The `json` extension (`LOAD json;`).

`json_tree` *DuckDB function json_tree*

Description

DuckDB function `json_tree()`.

Usage

```
json_tree(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR`

Value

Unspecified.

Overloads

- `json_tree(col0 = VARCHAR)`
- `json_tree(col0 = VARCHAR, col1 = VARCHAR)`
- `json_tree(col0 = JSON)`
- `json_tree(col0 = JSON, col1 = VARCHAR)`

Provided by

The `json` extension (`LOAD json;`).

`json_type` *DuckDB function json_type*

Description

DuckDB function `json_type()`.

Usage

```
json_type(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`VARCHAR | VARCHAR[]`

Value

`VARCHAR | VARCHAR[]`

Overloads

- `json_type(col0 = VARCHAR)`
- `json_type(col0 = VARCHAR, col1 = VARCHAR)`
- `json_type(col0 = VARCHAR, col1 = ` `VARCHAR[]` `)`
- `json_type(col0 = JSON)`
- `json_type(col0 = JSON, col1 = VARCHAR)`
- `json_type(col0 = JSON, col1 = ` `VARCHAR[]` `)`

Provided by

The `json` extension (`LOAD json;`).

`json_valid` *DuckDB function json_valid*

Description

DuckDB function `json_valid()`.

Usage

```
json_valid(col0)
```

Arguments

`col0`	`VARCHAR | JSON`

Value

`BOOLEAN`

Provided by

The `json` extension (`LOAD json;`).

`json_value` *DuckDB function json_value*

Description

DuckDB function `json_value()`.

Usage

```
json_value(col0, col1)
```

Arguments

`col0`	`VARCHAR | JSON`
`col1`	`BIGINT | VARCHAR | VARCHAR[]`

Value

`VARCHAR | VARCHAR[]`

Provided by

The `json` extension (`LOAD json;`).

`julian` *DuckDB function julian*

Description

Extract the Julian Day number from a date or timestamp.

Usage

```
julian(ts)
```

Arguments

`ts`	`DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`DOUBLE`

SQL examples

```
julian(timestamp '2006-01-01 12:00')
```

`kahan_sum` *DuckDB function kahan_sum*

Description

Calculates the sum using a more accurate floating point summation (Kahan Sum).

Usage

```
kahan_sum(arg)
```

Arguments

`arg`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
kahan_sum(A)
```

`kurtosis` *DuckDB function kurtosis*

Description

Returns the excess kurtosis (Fisher's definition) of all input values, with a bias correction according to the sample size.

Usage

```
kurtosis(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

`kurtosis_pop` *DuckDB function kurtosis_pop*

Description

Returns the excess kurtosis (Fisher's definition) of all input values, without bias correction.

Usage

```
kurtosis_pop(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

lag *DuckDB function lag*

Description

DuckDB function `lag()`.

Usage

```
lag(col0, col1, col2)
```

Arguments

col0	T
col1	BIGINT
col2	T

Value

T

lance_fts *DuckDB function lance_fts*

Description

DuckDB function `lance_fts()`.

Usage

```
lance_fts(col0, col1, col2, filter, prefilter, k)
```

Arguments

col0	VARCHAR
col1	VARCHAR
col2	VARCHAR
filter	VARCHAR
prefilter	BOOLEAN
k	BIGINT

Value

Unspecified.

Provided by

The `lance` extension (`LOAD lance;`).

`lance_hybrid_search` *DuckDB function lance_hybrid_search*

Description

DuckDB function `lance_hybrid_search()`.

Usage

```
lance_hybrid_search(
  col0,
  col1,
  col2,
  col3,
  col4,
  alpha,
  use_index,
  prefilter,
  refine_factor,
  nprobs,
  oversample_factor,
  k
)
```

Arguments

col0	VARCHAR
col1	VARCHAR
col2	FLOAT[] | DOUBLE[]
col3	VARCHAR
col4	VARCHAR
alpha	FLOAT
use_index	BOOLEAN
prefilter	BOOLEAN
refine_factor	BIGINT
nprobs	BIGINT
oversample_factor	INTEGER
k	BIGINT

Value

Unspecified.

Provided by

The `lance` extension (`LOAD lance;`).

`lance_vector_search` *DuckDB function lance_vector_search*

Description

DuckDB function `lance_vector_search()`.

Usage

```
lance_vector_search(
  col0,
  col1,
  col2,
  filter,
  explain_verbose,
  use_index,
  prefilter,
  refine_factor,
  nprobs,
  k
)
```

Arguments

col0	VARCHAR
col1	VARCHAR
col2	FLOAT[] | DOUBLE[]
filter	VARCHAR
explain_verbose	BOOLEAN
use_index	BOOLEAN
prefilter	BOOLEAN
refine_factor	BIGINT
nprobs	BIGINT
k	BIGINT

Value

Unspecified.

Provided by

The `lance` extension (`LOAD lance;`).

`last` *DuckDB function last*

Description

Returns the last value of a column. This function is affected by ordering.

Usage

```
last(arg)
```

Arguments

`arg`	`DECIMAL | ANY`

Value

`DECIMAL | ANY | T`

Overloads

- `last(arg = DECIMAL)`
- `last(arg = ANY)`
- `last(col0 = T)`

SQL examples

```
last(A)
```

`last_day` *DuckDB function last_day*

Description

Returns the last day of the month.

Usage

```
last_day(ts)
```

Arguments

`ts`	`DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

DATE

SQL examples

```
last_day(TIMESTAMP '1992-03-22 01:02:03.1234')
```

---

last_value *DuckDB function last_value*

Description

DuckDB function `last_value()`.

Usage

```
last_value(col0)
```

Arguments

col0	T

Value

T

---

lead *DuckDB function lead*

Description

DuckDB function `lead()`.

Usage

```
lead(col0, col1, col2)
```

Arguments

col0	T
col1	BIGINT
col2	T

Value

T

`least` *DuckDB function least*

Description

Returns the smallest value. For strings lexicographical ordering is used. Note that uppercase characters are considered "smaller" than lowercase characters, and collations are not supported.

Usage

```
least(arg1)
```

Arguments

`arg1`	`ANY`

Value

`ANY`

SQL examples

```
least(42, 84)
least('abc', 'bcd', 'cde', 'EFG')
```

See Also

Other aggregate: `greatest()`

Other date: `greatest()`

Other numeric: `bin()`, `bit_length()`, `formatReadableDecimalSize()`, `format_bytes()`, `greatest()`, `hex()`, `len()`, `parse_formatted_bytes()`, `to_base()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

Other timestamp: `greatest()`

`least_common_multiple`
DuckDB function least_common_multiple

Description

Computes the least common multiple of x and y.

Usage

```
least_common_multiple(x, y)
```

Arguments

x, y	BIGINT | HUGEINT

Value

```
BIGINT | HUGEINT
```

SQL examples

```
least_common_multiple(42, 57)
```

`left` *DuckDB function left*

Description

Extracts the left-most count characters.

Usage

```
left(string, count)
```

Arguments

string	VARCHAR
count	BIGINT

Value

```
VARCHAR
```

SQL examples

```
left('Hello ', 2)
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

left_grapheme *DuckDB function left_grapheme*

Description

Extracts the left-most count grapheme clusters.

Usage

```
left_grapheme(string, count)
```

Arguments

string	VARCHAR
count	BIGINT

Value

VARCHAR

SQL examples

```
left_grapheme('    ', 1)
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`len` *DuckDB function len*

Description

- `len(string = VARCHAR)`: Number of characters in `string`.
- `len(bit = BIT)`: Returns the bit-length of the `bit` argument.
- `len(list = ``ANY[]``)`: Returns the length of the `list`.

Usage

```
len(string)
```

Arguments

`string`	`VARCHAR`

Value

`BIGINT`

Overloads

- `len(string = VARCHAR)`
- `len(bit = BIT)`
- `len(list = ``ANY[]``)`

SQL examples

```
length('Hello ')
length(42::TINYINT::BIT)
length([1,2,3])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

Other numeric: `bin()`, `bit_length()`, `formatReadableDecimalSize()`, `format_bytes()`, `greatest()`, `hex()`, `least()`, `parse_formatted_bytes()`, `to_base()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`,

from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`length_grapheme` *DuckDB function length_grapheme*

Description

Number of grapheme clusters in `string`.

Usage

```
length_grapheme(string)
```

Arguments

`string`	`VARCHAR`

Value

`BIGINT`

SQL examples

```
length_grapheme('    ')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

levenshtein *DuckDB function levenshtein*

Description

The minimum number of single-character edits (insertions, deletions or substitutions) required to change one string to the other. Characters of different cases (e.g., `a` and `A`) are considered different.

Usage

```
levenshtein(s1, s2)
```

Arguments

`s1`, `s2`	`VARCHAR`

Value

`BIGINT`

SQL examples

```
levenshtein('duck', 'db')
```

See Also

Other text_similarity: `damerau_levenshtein()`, `hamming()`, `jaccard()`, `jaro_similarity()`, `jaro_winkler_similarity()`

lgamma *DuckDB function lgamma*

Description

Computes the log of the gamma function.

Usage

```
lgamma(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
lgamma(2)
```

`like_escape` *DuckDB function like_escape*

Description

Returns `true` if the `string` matches the `like_specifier` (see Pattern Matching) using case-sensitive matching. `escape_character` is used to search for wildcard characters in the `string`.

Usage

```
like_escape(string, like_specifier, escape_character)
```

Arguments

`string`, `like_specifier`, `escape_character`	`VARCHAR`

Value

`BOOLEAN`

SQL examples

```
like_escape('a%c', 'a$%c', '$')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`list_aggregate` *DuckDB function list_aggregate*

Description

Executes the aggregate function `function_name` on the elements of `list`.

Usage

```
list_aggregate(list, function_name)
```

Arguments

`list`	`ANY[]`
`function_name`	`VARCHAR`

Value

`ANY`

SQL examples

```
list_aggregate([1, 2, NULL], 'min')
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_any_value` *DuckDB function list_any_value*

Description

DuckDB macro `list_any_value()`.

Usage

```
list_any_value(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_append` *DuckDB function list_append*

Description

DuckDB macro `list_append()`.

Usage

```
list_append(l, e)
```

Arguments

`l`, `e`	Unspecified.

Value

Unspecified.

`list_approx_count_distinct` *DuckDB function list_approx_count_distinct*

Description

DuckDB macro `list_approx_count_distinct()`.

Usage

```
list_approx_count_distinct(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_avg *DuckDB function list_avg*

Description

DuckDB macro `list_avg()`.

Usage

```
list_avg(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_bit_and *DuckDB function list_bit_and*

Description

DuckDB macro `list_bit_and()`.

Usage

```
list_bit_and(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_bit_or *DuckDB function list_bit_or*

Description

DuckDB macro `list_bit_or()`.

Usage

```
list_bit_or(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_bit_xor *DuckDB function list_bit_xor*

Description

DuckDB macro `list_bit_xor()`.

Usage

```
list_bit_xor(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_bool_and *DuckDB function list_bool_and*

Description

DuckDB macro `list_bool_and()`.

Usage

```
list_bool_and(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_bool_or *DuckDB function list_bool_or*

Description

DuckDB macro `list_bool_or()`.

Usage

```
list_bool_or(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_concat` *DuckDB function list_concat*

Description

Concatenates lists. `NULL` inputs are skipped. See also operator `||`.

Usage

```
list_concat()
```

Value

`ANY[]`

SQL examples

```
list_concat([2, 3], [4, 5, 6], [7])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product` `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_contains` *DuckDB function list_contains*

Description

Returns true if the list contains the element.

Usage

```
list_contains(list, element)
```

Arguments

`list`	`T[]`
`element`	`T`

Value

BOOLEAN

SQL examples

```
list_contains([1, 2, NULL], 1)
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

---

list_cosine_distance *DuckDB function list_cosine_distance*

Description

Computes the cosine distance between two same-sized lists.

Usage

```
list_cosine_distance(list1, list2)
```

Arguments

list1, list2	FLOAT[] | DOUBLE[]

Value

FLOAT | DOUBLE

SQL examples

```
list_cosine_distance([1, 2, 3], [1, 2, 3])
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

`list_cosine_similarity`
DuckDB function list_cosine_similarity

Description

Computes the cosine similarity between two same-sized lists.

Usage

```
list_cosine_similarity(list1, list2)
```

Arguments

`list1`, `list2` `FLOAT[] | DOUBLE[]`

Value

`FLOAT | DOUBLE`

SQL examples

```
list_cosine_similarity([1, 2, 3], [1, 2, 3])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_count` *DuckDB function list_count*

Description

DuckDB macro `list_count()`.

Usage

```
list_count(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_distance` *DuckDB function list_distance*

Description

Calculates the Euclidean distance between two points with coordinates given in two inputs lists of equal length.

Usage

```
list_distance(list1, list2)
```

Arguments

`list1`, `list2`	`FLOAT[] | DOUBLE[]`

Value

`FLOAT | DOUBLE`

SQL examples

```
list_distance([1, 2, 3], [1, 2, 5])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_distinct` *DuckDB function list_distinct*

Description

Removes all duplicates and `NULL` values from a list. Does not preserve the original order.

Usage

```
list_distinct(list)
```

Arguments

`list`	`T[]`

Value

`T[]`

SQL examples

```
list_distinct([1, 1, NULL, -3, 1, 5])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_entropy` *DuckDB function list_entropy*

Description

DuckDB macro `list_entropy()`.

Usage

```
list_entropy(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_extract` *DuckDB function list_extract*

Description

Extract the `index`th (1-based) value from the list.

Usage

```
list_extract(list, index)
```

Arguments

`list`	`T[] | VARCHAR`
`index`	`BIGINT`

Value

`T | VARCHAR`

SQL examples

```
list_extract([4, 5, 6], 3)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_filter` *DuckDB function list_filter*

Description

Constructs a list from those elements of the input `list` for which the `lambda` function returns `true`. DuckDB must be able to cast the `lambda` function's return type to `BOOL`. The return type of `list_filter` is the same as the input list's.

Usage

```
list_filter(list, lambda)
```

Arguments

`list`	`ANY[]`
`lambda`	`LAMBDA`

Value

`ANY[]`

SQL examples

```
list_filter([3, 4, 5], lambda x : x > 4)
```

See Also

Other lambda: `list_reduce()`, `list_transform()`

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_first` *DuckDB function list_first*

Description

DuckDB macro `list_first()`.

Usage

```
list_first(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_grade_up` *DuckDB function list_grade_up*

Description

Works like list_sort, but the results are the indexes that correspond to the position in the original list instead of the actual values.

Usage

```
list_grade_up(list, col1, col2)
```

Arguments

`list`	`ANY[]`
`col1`	`VARCHAR`
`col2`	`VARCHAR`

Value

`ANY[]`

Overloads

- `list_grade_up(list = `ANY[]`)`
- `list_grade_up(list = `ANY[]`, col1 = VARCHAR)`
- `list_grade_up(list = `ANY[]`, col1 = VARCHAR, col2 = VARCHAR)`

SQL examples

```
list_grade_up([3, 6, 1, 2])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_has_all` *DuckDB function list_has_all*

Description

Returns true if all elements of list2 are in list1. NULLs are ignored.

Usage

```
list_has_all(list1, list2)
```

Arguments

`list1`, `list2` `T[]`

Value

`BOOLEAN`

SQL examples

```
list_has_all([1, 2, 3], [2, 3])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_has_any` *DuckDB function list_has_any*

Description

Returns true if the lists have any element in common. NULLs are ignored.

Usage

```
list_has_any(list1, list2)
```

Arguments

`list1`, `list2`	`T[]`

Value

`BOOLEAN`

SQL examples

```
list_has_any([1, 2, 3], [2, 3, 4])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_produc`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_histogram` *DuckDB function list_histogram*

Description

DuckDB macro `list_histogram()`.

Usage

```
list_histogram(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_inner_product` *DuckDB function list_inner_product*

Description

Computes the inner product between two same-sized lists.

Usage

```
list_inner_product(list1, list2)
```

Arguments

`list1`, `list2`	`FLOAT[] | DOUBLE[]`

Value

`FLOAT | DOUBLE`

SQL examples

```
list_inner_product([1, 2, 3], [1, 2, 3])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

list_intersect *DuckDB function list_intersect*

Description

Returns a list containing the distinct elements that are present in both `list1` and `list2`.

Usage

```
list_intersect(list1, list2)
```

Arguments

`list1`, `list2`	`T[]`

Value

`T[]`

SQL examples

```
list_intersect([1, 2, 3], [2, 3, 4])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

list_kurtosis *DuckDB function list_kurtosis*

Description

DuckDB macro `list_kurtosis()`.

Usage

```
list_kurtosis(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_kurtosis_pop` *DuckDB function list_kurtosis_pop*

Description

DuckDB macro `list_kurtosis_pop()`.

Usage

```
list_kurtosis_pop(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_last` *DuckDB function list_last*

Description

DuckDB macro `list_last()`.

Usage

```
list_last(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_mad *DuckDB function list_mad*

Description

DuckDB macro `list_mad()`.

Usage

```
list_mad(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_max *DuckDB function list_max*

Description

DuckDB macro `list_max()`.

Usage

```
list_max(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_median *DuckDB function list_median*

Description

DuckDB macro `list_median()`.

Usage

```
list_median(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_min *DuckDB function list_min*

Description

DuckDB macro `list_min()`.

Usage

```
list_min(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_mode` *DuckDB function list_mode*

Description

DuckDB macro `list_mode()`.

Usage

```
list_mode(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_negative_inner_product`
DuckDB function list_negative_inner_product

Description

Computes the negative inner product between two same-sized lists.

Usage

```
list_negative_inner_product(list1, list2)
```

Arguments

`list1`, `list2`	`FLOAT[] | DOUBLE[]`

Value

`FLOAT | DOUBLE`

SQL examples

```
list_negative_inner_product([1, 2, 3], [1, 2, 3])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_position` *DuckDB function list_position*

Description

Returns the index of the `element` if the `list` contains the `element`. If the `element` is not found, it returns `NULL`.

Usage

```
list_position(list, element)
```

Arguments

`list`	`T[]`
`element`	`T`

Value

`INTEGER`

SQL examples

```
list_position([1, 2, NULL], 2)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_prepend` *DuckDB function list_prepend*

Description

DuckDB macro `list_prepend()`.

Usage

```
list_prepend(e, l)
```

Arguments

`e`, `l`	Unspecified.

Value

Unspecified.

`list_product` *DuckDB function list_product*

Description

DuckDB macro `list_product()`.

Usage

```
list_product(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_reduce` *DuckDB function list_reduce*

Description

Reduces all elements of the input `list` into a single scalar value by executing the `lambda` function on a running result and the next list element. The `lambda` function has an optional `initial_value` argument.

Usage

```
list_reduce(list, lambda, initial_value)
```

Arguments

`list`	`ANY[]`
`lambda`	`LAMBDA`
`initial_value`	`ANY`

Value

`ANY`

Overloads

- `list_reduce(list = ``ANY[]``, lambda = LAMBDA)`
- `list_reduce(list = ``ANY[]``, lambda = LAMBDA, initial_value = ANY)`

SQL examples

```
list_reduce([1, 2, 3], lambda x, y : x + y)
```

See Also

Other lambda: `list_filter()`, `list_transform()`

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_resize` *DuckDB function list_resize*

Description

Resizes the `list` to contain `size` elements. Initializes new elements with `value` or `NULL` if `value` is not set.

Usage

```
list_resize(list, size, value)
```

Arguments

`list`	`ANY[]`
`size`	`ANY`
`value`	`ANY`

Value

`ANY[]`

Overloads

- `list_resize(list = `ANY[]`, size = ANY)`
- `list_resize(list = `ANY[]`, size = ANY, value = ANY)`

SQL examples

```
list_resize([1, 2, 3], 5, 0)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_reverse` *DuckDB function list_reverse*

Description

DuckDB macro `list_reverse()`.

Usage

```
list_reverse(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_reverse_sort` *DuckDB function list_reverse_sort*

Description

Sorts the elements of the list in reverse order.

Usage

```
list_reverse_sort(list, col1)
```

Arguments

`list`	`ANY[]`
`col1`	`VARCHAR`

Value

`ANY[]`

Overloads

- `list_reverse_sort(list = ``ANY[]``)`
- `list_reverse_sort(list = ``ANY[]``, col1 = VARCHAR)`

SQL examples

```
list_reverse_sort([3, 6, 1, 2])
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

---

list_select *DuckDB function list_select*

Description

Returns a list based on the elements selected by the index_list.

Usage

```
list_select(value_list, index_list)
```

Arguments

value_list	T[]
index_list	BIGINT[]

Value

T[]

SQL examples

```
list_select([10, 20, 30, 40], [1, 4])
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

list_sem *DuckDB function list_sem*

Description

DuckDB macro `list_sem()`.

Usage

```
list_sem(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_skewness *DuckDB function list_skewness*

Description

DuckDB macro `list_skewness()`.

Usage

```
list_skewness(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_slice` *DuckDB function list_slice*

Description

- `list_slice(list = ANY, begin = ANY, end = ANY)`: Extracts a sublist or substring using slice conventions. Negative values are accepted.
- `list_slice(list = ANY, begin = ANY, end = ANY, step = BIGINT)`: list_slice with added step feature.

Usage

```
list_slice(list, begin, end, step)
```

Arguments

`list`	`ANY`
`begin`	`ANY`
`end`	`ANY`
`step`	`BIGINT`

Value

`ANY`

Overloads

- `list_slice(list = ANY, begin = ANY, end = ANY)`
- `list_slice(list = ANY, begin = ANY, end = ANY, step = BIGINT)`

SQL examples

```
list_slice([4, 5, 6], 2, 3)
list_slice([4, 5, 6], 1, 3, 2)
array_slice('DuckDB', 3, 4)
array_slice('DuckDB', 3, NULL)
array_slice('DuckDB', 0, -3)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`list_sort` *DuckDB function list_sort*

Description

Sorts the elements of the list.

Usage

```
list_sort(list, col1, col2)
```

Arguments

list	ANY[]
col1	VARCHAR
col2	VARCHAR

Value

ANY[]

Overloads

- list_sort(list = `ANY[]`)
- list_sort(list = `ANY[]`, col1 = VARCHAR)
- list_sort(list = `ANY[]`, col1 = VARCHAR, col2 = VARCHAR)

SQL examples

```
list_sort([3, 6, 1, 2])
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

---

list_stddev_pop *DuckDB function list_stddev_pop*

Description

DuckDB macro list_stddev_pop().

Usage

```
list_stddev_pop(l)
```

Arguments

l	Unspecified.

Value

Unspecified.

---

list_stddev_samp *DuckDB function list_stddev_samp*

Description

DuckDB macro list_stddev_samp().

Usage

```
list_stddev_samp(l)
```

Arguments

l	Unspecified.

Value

Unspecified.

list_string_agg *DuckDB function list_string_agg*

Description

DuckDB macro `list_string_agg()`.

Usage

```
list_string_agg(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

list_sum *DuckDB function list_sum*

Description

DuckDB macro `list_sum()`.

Usage

```
list_sum(l)
```

Arguments

`l`	Unspecified.

Value

Unspecified.

`list_transform` *DuckDB function list_transform*

Description

Returns a list that is the result of applying the `lambda` function to each element of the input `list`. The return type is defined by the return type of the `lambda` function.

Usage

```
list_transform(list, lambda)
```

Arguments

`list`	`ANY[]`
`lambda`	`LAMBDA`

Value

`ANY[]`

SQL examples

```
list_transform([1, 2, 3], lambda x : x + 1)
```

See Also

Other lambda: `list_filter()`, `list_reduce()`

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_unique` *DuckDB function list_unique*

Description

Counts the unique elements of a `list`.

Usage

```
list_unique(list)
```

Arguments

`list`	`ANY[]`

Value

`UBIGINT`

SQL examples

```
list_unique([1, 1, NULL, -3, 1, 5])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_value` *DuckDB function list_value*

Description

Creates a LIST containing the argument values.

Usage

```
list_value(any)
```

Arguments

`any`	`T`

Value

`"NULL"[] | T[]`

Overloads

- `list_value()`
- `list_value(any = T)`

SQL examples

```
list_value(4, 5, 6)
```

See Also

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_where(), list_zip(), or--or, range(), repeat(), unpivot_list()

---

list_var_pop *DuckDB function list_var_pop*

Description

DuckDB macro list_var_pop().

Usage

```
list_var_pop(l)
```

Arguments

l	Unspecified.

Value

Unspecified.

---

list_var_samp *DuckDB function list_var_samp*

Description

DuckDB macro list_var_samp().

Usage

```
list_var_samp(l)
```

Arguments

l	Unspecified.

Value

Unspecified.

`list_where` *DuckDB function list_where*

Description

Returns a list with the `BOOLEAN`s in `mask_list` applied as a mask to the `value_list`.

Usage

```
list_where(value_list, mask_list)
```

Arguments

`value_list`	`T[]`
`mask_list`	`BOOLEAN[]`

Value

`T[]`

SQL examples

```
list_where([10, 20, 30, 40], [true, false, false, true])
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_zip()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`list_zip` *DuckDB function list_zip*

Description

Zips n `LIST`s to a new `LIST` whose length will be that of the longest list. Its elements are structs of n elements from each list `list_1`, …, `list_n`, missing elements are replaced with `NULL`. If `truncate` is set, all lists are truncated to the smallest list length.

Usage

```
list_zip()
```

Value

`STRUCT[]`

SQL examples

```
list_zip([1, 2], [3, 4], [5, 6])
list_zip([1, 2], [3, 4], [5, 6, 7])
list_zip([1, 2], [3, 4], [5, 6, 7], true)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `or--or`, `range()`, `repeat()`, `unpivot_list()`

`ln` *DuckDB function ln*

Description

Computes the natural logarithm of x.

Usage

```
ln(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
ln(2)
```

`load_aws_credentials` *DuckDB function load_aws_credentials*

Description

DuckDB function `load_aws_credentials()`.

Usage

```
load_aws_credentials(col0, redact_secret, set_region)
```

Arguments

`col0`	`VARCHAR`
`redact_secret`	`BOOLEAN`
`set_region`	`BOOLEAN`

Value

Unspecified.

Overloads

- `load_aws_credentials(redact_secret = BOOLEAN, set_region = BOOLEAN)`
- `load_aws_credentials(col0 = VARCHAR, redact_secret = BOOLEAN, set_region = BOOLEAN)`

Provided by

The `aws` extension (`LOAD aws;`).

`log` *DuckDB function log*

Description

Computes the logarithm of x to base b. b may be omitted, in which case the default 10.

Usage

```
log(b, x)
```

Arguments

`b, x`	`DOUBLE`

Value

DOUBLE

Overloads

- log(b = DOUBLE)
- log(b = DOUBLE, x = DOUBLE)

SQL examples

```
log(2, 64)
```

---

log10 *DuckDB function log10*

Description

Computes the 10-log of x.

Usage

```
log10(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
log10(1000)
```

`log2` *DuckDB function log2*

Description

Computes the 2-log of x.

Usage

```
log2(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
log2(8)
```

`lower` *DuckDB function lower*

Description

Converts `string` to lower case.

Usage

```
lower(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
lower('Hello')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`lpad` *DuckDB function lpad*

Description

Pads the `string` with the `character` on the left until it has `count` characters. Truncates the `string` on the right if it has more than `count` characters.

Usage

```
lpad(string, count, character)
```

Arguments

`string`	`VARCHAR`
`count`	`INTEGER`
`character`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
lpad('hello', 8, '>')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(),

rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`ltrim` *DuckDB function ltrim*

Description

Removes any occurrences of any of the `characters` from the left side of the `string`. `characters` defaults to `space`.

Usage

```
ltrim(string, characters)
```

Arguments

`string, characters`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `ltrim(string = VARCHAR)`
- `ltrim(string = VARCHAR, characters = VARCHAR)`

SQL examples

```
ltrim('    test  ')
ltrim('>>>>test<<', '><')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`mad` *DuckDB function mad*

Description

Returns the median absolute deviation for the values within x. NULL values are ignored. Temporal types return a positive INTERVAL. .

Usage

```
mad(x)
```

Arguments

x	DECIMAL | FLOAT | DOUBLE | DATE | TIMESTAMP | TIME | TIMESTAMP WITH TIME ZONE |

Value

`DECIMAL | FLOAT | DOUBLE | INTERVAL`

SQL examples

```
mad(x)
```

`make_date` *DuckDB function make_date*

Description

- `make_date(year = BIGINT, month = BIGINT, day = BIGINT)`: The date for the given parts.
- `make_date(date.struct = `STRUCT("year" BIGINT, "month" BIGINT, "day" BIGINT)`)`: The date for the given struct.
- `make_date(col0 = INTEGER)`: (Description missing.)

Usage

```
make_date(year, month, day)
```

Arguments

year, month, day	BIGINT

Value

`DATE`

Overloads

- `make_date(col0 = INTEGER)`
- `make_date(year = BIGINT, month = BIGINT, day = BIGINT)`
- `make_date(date.struct = `STRUCT("year" BIGINT, "month" BIGINT, "day" BIGINT)`)`

SQL examples

```
make_date(1992, 9, 20)
make_date({'year': 2024, 'month': 11, 'day': 14})
```

---

make_time *DuckDB function make_time*

Description

The time for the given parts.

Usage

```
make_time(hour, minute, seconds)
```

Arguments

hour	BIGINT
minute	BIGINT
seconds	DOUBLE

Value

TIME

SQL examples

```
make_time(13, 34, 27.123456)
```

`make_timestamp` *DuckDB function make_timestamp*

Description

The timestamp for the given parts.

Usage

```
make_timestamp(year, month, day, hour, minute, seconds)
```

Arguments

year	BIGINT
month	BIGINT
day	BIGINT
hour	BIGINT
minute	BIGINT
seconds	DOUBLE

Value

TIMESTAMP

Overloads

- `make_timestamp(year = BIGINT, month = BIGINT, day = BIGINT, hour = BIGINT, minute = BIGINT, seconds = DOUBLE)`
- `make_timestamp(year = BIGINT)`

SQL examples

```
make_timestamp(1992, 9, 20, 13, 34, 27.123456)
```

`make_timestamp_ms` *DuckDB function make_timestamp_ms*

Description

The timestamp for the given microseconds since the epoch.

Usage

```
make_timestamp_ms(nanos)
```

Arguments

nanos	BIGINT

Value

TIMESTAMP

SQL examples

```
make_timestamp_ms(1732117793000000)
```

---

make_timestamp_ns *DuckDB function make_timestamp_ns*

Description

The timestamp for the given nanoseconds since epoch.

Usage

```
make_timestamp_ns(nanos)
```

Arguments

nanos	BIGINT

Value

TIMESTAMP_NS

SQL examples

```
make_timestamp_ns(1732117793000000000)
```

---

make_timestamptz *DuckDB function make_timestamptz*

Description

DuckDB function `make_timestamptz()`.

Usage

```
make_timestamptz(col0, col1, col2, col3, col4, col5, col6)
```

Arguments

col0	BIGINT
col1	BIGINT
col2	BIGINT
col3	BIGINT
col4	BIGINT
col5	DOUBLE
col6	VARCHAR

Value

TIMESTAMP WITH TIME ZONE

Overloads

- make_timestamptz(col0 = BIGINT, col1 = BIGINT, col2 = BIGINT, col3 = BIGINT, col4 = BIGINT, col5 = DOUBLE)
- make_timestamptz(col0 = BIGINT, col1 = BIGINT, col2 = BIGINT, col3 = BIGINT, col4 = BIGINT, col5 = DOUBLE, col6 = VARCHAR)
- make_timestamptz(col0 = BIGINT)

Provided by

The icu extension (LOAD icu;).

---

make_type *DuckDB function make_type*

Description

Construct a type from its name and optional parameters.

Usage

```
make_type(name)
```

Arguments

name	VARCHAR

Value

TYPE

SQL examples

```
make_type('DECIMAL', 10, 2)
```

map — *DuckDB function map*

Description

Creates a map from a set of keys and values.

Usage

```
map(keys, values)
```

Arguments

keys	K[]
values	V[]

Value

```
MAP("NULL", "NULL") | MAP(K, V)
```

Overloads

- `map()`
- `map(keys = \`K[]\`, values = \`V[]\`)`

SQL examples

```
map(['key1', 'key2'], ['val1', 'val2'])
```

map_concat — *DuckDB function map_concat*

Description

Returns a map created from merging the input maps, on key collision the value is taken from the last map with that key.

Usage

```
map_concat()
```

Value

```
LIST
```

SQL examples

```
map_concat(map([1, 2], ['a', 'b']), map([2, 3], ['c', 'd']));
```

`map_contains` *DuckDB function map_contains*

Description

Checks if a map contains a given key.

Usage

```
map_contains(map, key)
```

Arguments

`map`	`MAP(K, V)`
`key`	`K`

Value

`BOOLEAN`

SQL examples

```
map_contains(MAP {'key1': 10, 'key2': 20, 'key3': 30}, 'key2')
```

`map_contains_entry` *DuckDB function map_contains_entry*

Description

DuckDB macro `map_contains_entry()`.

Usage

```
map_contains_entry(map, key, value)
```

Arguments

`map`, `key`, `value`	Unspecified.

Value

Unspecified.

map_contains_value *DuckDB function map_contains_value*

Description

DuckDB macro `map_contains_value()`.

Usage

```
map_contains_value(map, value)
```

Arguments

`map`, `value`	Unspecified.

Value

Unspecified.

map_entries *DuckDB function map_entries*

Description

Returns the map entries as a list of keys/values.

Usage

```
map_entries(map)
```

Arguments

`map`	`MAP(K, V)`

Value

`STRUCT("key" K, "value" V)[]`

SQL examples

```
map_entries(map(['key'], ['val']))
```

`map_extract` *DuckDB function map_extract*

Description

Returns a list containing the value for a given key or an empty list if the key is not contained in the map. The type of the key provided in the second parameter must match the type of the map's keys else an error is returned.

Usage

```
map_extract(map, key)
```

Arguments

map	MAP(K, V)
key	K

Value

`V[]`

SQL examples

```
map_extract(map(['key'], ['val']), 'key')
```

`map_extract_value` *DuckDB function map_extract_value*

Description

Returns the value for a given key or NULL if the key is not contained in the map. The type of the key provided in the second parameter must match the type of the map's keys else an error is returned.

Usage

```
map_extract_value(map, key)
```

Arguments

map	MAP(K, V)
key	K

Value

`V`

SQL examples

```
map_extract_value(map(['key'], ['val']), 'key')
```

`map_from_entries` *DuckDB function map_from_entries*

Description

Returns a map created from the entries of the array.

Usage

```
map_from_entries(map)
```

Arguments

map	STRUCT(K, V)[]

Value

```
MAP(K, V)
```

SQL examples

```
map_from_entries([{k: 5, v: 'val1'}, {k: 3, v: 'val2'}]);
```

`map_keys` *DuckDB function map_keys*

Description

Returns the keys of a map as a list.

Usage

```
map_keys(map)
```

Arguments

map	MAP(K, V)

Value

```
K[]
```

SQL examples

```
map_keys(map(['key'], ['val']))
```

map_to_pg_oid — *DuckDB function map_to_pg_oid*

Description

DuckDB macro `map_to_pg_oid()`.

Usage

```
map_to_pg_oid(type_name)
```

Arguments

`type_name`	Unspecified.

Value

Unspecified.

map_values — *DuckDB function map_values*

Description

Returns the values of a map as a list.

Usage

```
map_values(map)
```

Arguments

`map`	`MAP(K, V)`

Value

`V[]`

SQL examples

```
map_values(map(['key'], ['val']))
```

`max` *DuckDB function max*

Description

Returns the maximum value present in arg.

Usage

```
max(arg, col1)
```

Arguments

`arg`	`ANY`
`col1`	`BIGINT`

Value

`ANY | ANY[]`

Overloads

- `max(arg = ANY)`
- `max(arg = ANY, col1 = BIGINT)`

SQL examples

```
max(A)
```

`md5` *DuckDB function md5*

Description

- `md5(string = VARCHAR)`: Returns the MD5 hash of the `string` as a `VARCHAR`.
- `md5(blob = BLOB)`: Returns the MD5 hash of the `blob` as a `VARCHAR`.

Usage

```
md5(string)
```

Arguments

`string`	`VARCHAR`

Value

VARCHAR

Overloads

- md5(string = VARCHAR)
- md5(blob = BLOB)

SQL examples

```
md5('abc')
md5('\xAA\xBB'::BLOB)
```

See Also

Other blob: decode(), encode(), from_base64(), hex(), md5_number(), octet_length(), or--or, repeat(), sha1(), sha256(), to_base64(), unbin(), unhex()

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

md5_number *DuckDB function md5_number*

Description

- md5_number(string = VARCHAR): Returns the MD5 hash of the string as a HUGEINT.
- md5_number(blob = BLOB): Returns the MD5 hash of the blob as a HUGEINT.

Usage

```
md5_number(string)
```

Arguments

string	VARCHAR

Value

UHUGEINT

Overloads

- `md5_number(string = VARCHAR)`
- `md5_number(blob = BLOB)`

SQL examples

```
md5_number('abc')
md5_number('\xAA\xBB'::BLOB)
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unbin()`, `unhex()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`md5_number_lower` *DuckDB function md5_number_lower*

Description

DuckDB macro `md5_number_lower()`.

Usage

```
md5_number_lower(param)
```

Arguments

`param`	Unspecified.

Value

Unspecified.

`md5_number_upper` *DuckDB function md5_number_upper*

Description

DuckDB macro `md5_number_upper()`.

Usage

```
md5_number_upper(param)
```

Arguments

`param`	Unspecified.

Value

Unspecified.

`median` *DuckDB function median*

Description

Returns the middle value of the set. NULL values are ignored. For even value counts, interpolate-able types (numeric, date/time) return the average of the two middle values. Non-interpolate-able types (everything else) return the lower of the two middle values.

Usage

```
median(x)
```

Arguments

`x`	`ANY`

Value

`ANY`

SQL examples

```
median(x)
```

metadata_info *DuckDB function metadata_info*

Description

DuckDB function `metadata_info()`.

Usage

```
metadata_info()
```

Value

Unspecified.

microsecond *DuckDB function microsecond*

Description

Extract the microsecond component from a date or timestamp.

Usage

```
microsecond(ts)
```

Arguments

ts	DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W

Value

BIGINT

SQL examples

```
microsecond(timestamp '2021-08-03 11:59:44.123456')
```

`millennium` *DuckDB function millennium*

Description

Extract the millennium component from a date or timestamp.

Usage

```
millennium(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
millennium(timestamp '2021-08-03 11:59:44.123456')
```

`millisecond` *DuckDB function millisecond*

Description

Extract the millisecond component from a date or timestamp.

Usage

```
millisecond(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W`

Value

`BIGINT`

SQL examples

```
millisecond(timestamp '2021-08-03 11:59:44.123456')
```

`min` *DuckDB function min*

Description

Returns the minimum value present in arg.

Usage

```
min(arg, col1)
```

Arguments

`arg`	`ANY`
`col1`	`BIGINT`

Value

`ANY | ANY[]`

Overloads

- `min(arg = ANY)`
- `min(arg = ANY, col1 = BIGINT)`

SQL examples

```
min(A)
```

`minute` *DuckDB function minute*

Description

Extract the minute component from a date or timestamp.

Usage

```
minute(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W`

Value

`BIGINT`

SQL examples

```
minute(timestamp '2021-08-03 11:59:44.123456')
```

`mod` *DuckDB function mod*

Description

DuckDB function `mod()`.

Usage

```
mod(col0, col1)
```

Arguments

`col0`, `col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT`

`mode` *DuckDB function mode*

Description

Returns the most frequent value for the values within x. NULL values are ignored.

Usage

```
mode(x)
```

Arguments

`x`	`ANY`

Value

`ANY`

`month` *DuckDB function month*

Description

Extract the month component from a date or timestamp.

Usage

```
month(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
month(timestamp '2021-08-03 11:59:44.123456')
```

`monthname` *DuckDB function monthname*

Description

The (English) name of the month.

Usage

```
monthname(ts)
```

Arguments

`ts`	`DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`VARCHAR`

SQL examples

```
monthname(TIMESTAMP '1992-09-20')
```

`multiply` *DuckDB function multiply*

Description

DuckDB function `multiply()`.

Usage

```
multiply(col0, col1)
```

Arguments

`col0`, `col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | INTERVAL`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | INTERVAL`

`murmur3_32` *DuckDB function murmur3_32*

Description

DuckDB function `murmur3_32()`.

Usage

```
murmur3_32(col0)
```

Arguments

`col0`	`ANY`

Value

`INTEGER`

Provided by

The `ducklake` extension (`LOAD ducklake;`).

`mysql_clear_cache` *DuckDB function mysql_clear_cache*

Description

DuckDB function `mysql_clear_cache()`.

Usage

```
mysql_clear_cache()
```

Value

Unspecified.

Provided by

The `mysql_scanner` extension (`LOAD mysql_scanner;`).

`mysql_debug_execution_plan`
DuckDB function mysql_debug_execution_plan

Description

DuckDB function `mysql_debug_execution_plan()`.

Usage

```
mysql_debug_execution_plan()
```

Value

Unspecified.

Provided by

The `mysql_scanner` extension (`LOAD mysql_scanner;`).

`mysql_execute` *DuckDB function mysql_execute*

Description

DuckDB function `mysql_execute()`.

Usage

```
mysql_execute(col0, col1, params)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`params`	`ANY`

Value

Unspecified.

Provided by

The `mysql_scanner` extension (`LOAD mysql_scanner;`).

`mysql_explain_federated`
DuckDB function mysql_explain_federated

Description

DuckDB function `mysql_explain_federated()`.

Usage

```
mysql_explain_federated()
```

Value

Unspecified.

Provided by

The `mysql_scanner` extension (`LOAD mysql_scanner;`).

`mysql_query` *DuckDB function mysql_query*

Description

DuckDB function `mysql_query()`.

Usage

```
mysql_query(col0, col1, stream_results, params)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`stream_results`	`BOOLEAN`
`params`	`ANY`

Value

Unspecified.

Provided by

The `mysql_scanner` extension (`LOAD mysql_scanner;`).

`nanosecond` *DuckDB function nanosecond*

Description

Extract the nanosecond component from a date or timestamp.

Usage

```
nanosecond(tsns)
```

Arguments

`tsns`	`DATE | TIMESTAMP | INTERVAL | TIME | TIME_NS | TIME WITH TIME ZONE | TIMESTAMP_N`

Value

`BIGINT`

SQL examples

```
nanosecond(timestamp_ns '2021-08-03 11:59:44.123456789')
```

`netmask` *DuckDB function netmask*

Description

DuckDB function `netmask()`.

Usage

```
netmask(col0)
```

Arguments

`col0`	`INET`

Value

`INET`

Provided by

The `inet` extension (`LOAD inet;`).

`network` *DuckDB function network*

Description

DuckDB function `network()`.

Usage

```
network(col0)
```

Arguments

`col0`	`INET`

Value

`INET`

Provided by

The `inet` extension (`LOAD inet;`).

`nextafter` *DuckDB function nextafter*

Description

Returns the next floating point value after x in the direction of y.

Usage

```
nextafter(x, y)
```

Arguments

x, y	DOUBLE | FLOAT

Value

```
DOUBLE | FLOAT
```

SQL examples

```
nextafter(1::float, 2::float)
```

`nextval` *DuckDB function nextval*

Description

Return the following value of the sequence.

Usage

```
nextval(sequence_name)
```

Arguments

sequence_name	VARCHAR

Value

```
BIGINT
```

SQL examples

```
nextval('my_sequence_name')
```

nfc_normalize *DuckDB function nfc_normalize*

Description

Converts `string` to Unicode NFC normalized string. Useful for comparisons and ordering if text data is mixed between NFC normalized and not.

Usage

```
nfc_normalize(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
nfc_normalize('ardèch')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

normalized_interval *DuckDB function normalized_interval*

Description

Normalizes an INTERVAL to an equivalent interval.

Usage

```
normalized_interval(interval)
```

Arguments

`interval` `INTERVAL`

Value

`INTERVAL`

SQL examples

```
normalized_interval(INTERVAL '30 days')
```

`not-~~*` *DuckDB function !~~**

Description

DuckDB function `!~~*()`.

Usage

```
`!~~*`(col0, col1)
```

Arguments

`col0`, `col1` `VARCHAR`

Value

`BOOLEAN`

`not-~~` *DuckDB function !~~*

Description

DuckDB function `!~~()`.

Usage

```
`!~~`(col0, col1)
```

Arguments

`col0`, `col1` `VARCHAR`

Value

`BOOLEAN`

`not_ilike_escape` *DuckDB function not_ilike_escape*

Description

Returns `false` if the `string` matches the `like_specifier` (see Pattern Matching) using case-insensitive matching. `escape_character` is used to search for wildcard characters in the `string`.

Usage

```
not_ilike_escape(string, like_specifier, escape_character)
```

Arguments

`string`, `like_specifier`, `escape_character`	`VARCHAR`

Value

`BOOLEAN`

SQL examples

```
not_ilike_escape('A%c', 'a$%C', '$')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`not_like_escape` *DuckDB function not_like_escape*

Description

Returns `false` if the `string` matches the `like_specifier` (see Pattern Matching) using case-sensitive matching. `escape_character` is used to search for wildcard characters in the `string`.

Usage

```
not_like_escape(string, like_specifier, escape_character)
```

Arguments

`string`, `like_specifier`, `escape_character`	`VARCHAR`

Value

`BOOLEAN`

SQL examples

```
not_like_escape('a%c', 'a$%c', '$')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

nth_value *DuckDB function nth_value*

Description

DuckDB function `nth_value()`.

Usage

```
nth_value(col0, col1)
```

Arguments

col0	T
col1	BIGINT

Value

T

ntile *DuckDB function ntile*

Description

DuckDB function `ntile()`.

Usage

```
ntile(col0)
```

Arguments

col0	BIGINT

Value

BIGINT

nullif *DuckDB function nullif*

Description

DuckDB macro `nullif()`.

Usage

```
nullif(a, b)
```

Arguments

`a`, `b`	Unspecified.

Value

Unspecified.

obj_description *DuckDB function obj_description*

Description

DuckDB macro `obj_description()`.

Usage

```
obj_description(object_oid, catalog_name)
```

Arguments

`object_oid`, `catalog_name`	Unspecified.

Value

Unspecified.

`octet_length` *DuckDB function octet_length*

Description

- `octet_length(blob = BLOB)`: Number of bytes in `blob`.
- `octet_length(bitstring = BIT)`: Returns the number of bytes in the `bitstring`.

Usage

```
octet_length(blob)
```

Arguments

`blob`	`BLOB`

Value

`BIGINT`

Overloads

- `octet_length(blob = BLOB)`
- `octet_length(bitstring = BIT)`

SQL examples

```
octet_length('\xAA\xBB'::BLOB)
octet_length('1101011'::BITSTRING)
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unbin()`, `unhex()`

`or-` *DuckDB function |*

Description

Bitwise OR.

Usage

```
`|`(left, right)
```

Arguments

`left`, `right`	`TINYINT` | `SMALLINT` | `INTEGER` | `BIGINT` | `HUGEINT` | `UTINYINT` | `USMALLINT` | `UINTEGER` | `UBIGINT` | `UHUGEINT` | `BIT`

Value

`TINYINT` | `SMALLINT` | `INTEGER` | `BIGINT` | `HUGEINT` | `UTINYINT` | `USMALLINT` | `UINTEGER` | `UBIGINT` | `UHUGEINT` | `BIT`

SQL examples

```
32 | 3
```

`or--or` *DuckDB function* //

Description

Concatenates two strings, lists, or blobs. Any `NULL` input results in `NULL`. See also `concat(arg1, arg2, ...)` and `list_concat(list1, list2, ...)`.

Usage

```
`||`(arg1, arg2)
```

Arguments

`arg1`, `arg2`	`ANY`

Value

`ANY`

SQL examples

```
'Duck' || 'DB'
[1, 2, 3] || [4, 5, 6]
'\xAA'::BLOB || '\xBB'::BLOB
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unbin()`, `unhex()`

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`,

list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), range(), repeat(), unpivot_list()

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

parquet_bloom_probe *DuckDB function parquet_bloom_probe*

Description

DuckDB function `parquet_bloom_probe()`.

Usage

```
parquet_bloom_probe(col0, col1, col2)
```

Arguments

col0	VARCHAR | VARCHAR[]
col1	VARCHAR
col2	ANY

Value

Unspecified.

---

parquet_file_metadata *DuckDB function parquet_file_metadata*

Description

DuckDB function `parquet_file_metadata()`.

Usage

```
parquet_file_metadata(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

`parquet_full_metadata` *DuckDB function parquet_full_metadata*

Description

DuckDB function `parquet_full_metadata()`.

Usage

```
parquet_full_metadata(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

`parquet_kv_metadata` *DuckDB function parquet_kv_metadata*

Description

DuckDB function `parquet_kv_metadata()`.

Usage

```
parquet_kv_metadata(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

parquet_metadata *DuckDB function parquet_metadata*

Description

DuckDB function `parquet_metadata()`.

Usage

```
parquet_metadata(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

parquet_scan *DuckDB function parquet_scan*

Description

DuckDB function `parquet_scan()`.

Usage

```
parquet_scan(
  col0,
  can_have_nan,
  filename,
  union_by_name,
  debug_use_openssl,
  hive_partitioning,
  parquet_version,
  encryption_config,
  hive_types_autocast,
  binary_as_string,
  explicit_cardinality,
  compression,
  file_row_number,
  hive_types,
  schema
)
```

Arguments

col0	VARCHAR | VARCHAR[]
can_have_nan	BOOLEAN
filename	ANY
union_by_name	BOOLEAN
debug_use_openssl	BOOLEAN
hive_partitioning	BOOLEAN
parquet_version	VARCHAR
encryption_config	ANY
hive_types_autocast	BOOLEAN
binary_as_string	BOOLEAN
explicit_cardinality	UBIGINT
compression	VARCHAR
file_row_number	BOOLEAN
hive_types	ANY
schema	ANY

Value

Unspecified.

---

parquet_schema *DuckDB function parquet_schema*

Description

DuckDB function `parquet_schema()`.

Usage

```
parquet_schema(col0)
```

Arguments

col0	VARCHAR | VARCHAR[]

Value

Unspecified.

`parse_delta_filter_logline` *DuckDB function parse_delta_filter_logline*

Description

DuckDB macro `parse_delta_filter_logline()`.

Usage

```
parse_delta_filter_logline(x)
```

Arguments

`x`	Unspecified.

Value

Unspecified.

Provided by

The `delta` extension (`LOAD delta;`).

`parse_dirname` *DuckDB function parse_dirname*

Description

Returns the top-level directory name from the given `path`. `separator` options: `system`, `both_slash` (default), `forward_slash`, `backslash`.

Usage

```
parse_dirname(path, separator)
```

Arguments

`path`, `separator`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `parse_dirname(path = VARCHAR)`
- `parse_dirname(path = VARCHAR, separator = VARCHAR)`

SQL examples

```
parse_dirname('path/to/file.csv', 'system')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`parse_dirpath` *DuckDB function parse_dirpath*

Description

Returns the head of the `path` (the pathname until the last slash) similarly to Python's `os.path.dirname`. `separator` options: `system`, `both_slash` (default), `forward_slash`, `backslash`.

Usage

```
parse_dirpath(path, separator)
```

Arguments

`path`, `separator`
VARCHAR

Value

VARCHAR

Overloads

- `parse_dirpath(path = VARCHAR)`
- `parse_dirpath(path = VARCHAR, separator = VARCHAR)`

SQL examples

```
parse_dirpath('path/to/file.csv', 'forward_slash')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

parse_duckdb_log_message

DuckDB function parse_duckdb_log_message

Description

Parse the message into the expected logical type.

Usage

```
parse_duckdb_log_message(type, message)
```

Arguments

type, message	VARCHAR

Value

ANY

SQL examples

```
parse_duckdb_log_message('FileSystem', log_message)
```

`parse_filename` *DuckDB function parse_filename*

Description

Returns the last component of the `path` similarly to Python's `os.path.basename` function. If `trim_extension` is `true`, the file extension will be removed (defaults to `false`). `separator` options: `system`, `both_slash` (default), `forward_slash`, `backslash`.

Usage

```
parse_filename(string, trim_extension)
```

Arguments

`string`	`VARCHAR`
`trim_extension`	`VARCHAR | BOOLEAN`

Value

`VARCHAR`

Overloads

- `parse_filename(string = VARCHAR)`
- `parse_filename(string = VARCHAR, trim_extension = VARCHAR)`
- `parse_filename(string = VARCHAR, trim_extension = BOOLEAN)`
- `parse_filename(string = VARCHAR, trim_extension = BOOLEAN, separator = VARCHAR)`

SQL examples

```
parse_filename('path/to/file.csv', true, 'forward_slash')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`parse_formatted_bytes`

DuckDB function parse_formatted_bytes

Description

Parses a human-readable representation of a size in bytes into an integer.

Usage

```
parse_formatted_bytes(string)
```

Arguments

`string`	`VARCHAR`

Value

`UBIGINT`

SQL examples

```
parse_formatted_bytes('16 KiB')
```

See Also

Other numeric: `bin()`, `bit_length()`, `formatReadableDecimalSize()`, `format_bytes()`, `greatest()`, `hex()`, `least()`, `len()`, `to_base()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`parse_path` *DuckDB function parse_path*

Description

Returns a list of the components (directories and filename) in the `path` similarly to Python's `pathlib.parts` function. `separator` options: `system`, `both_slash` (default), `forward_slash`, `backslash`.

Usage

```
parse_path(path, separator)
```

Arguments

`path`, `separator`
VARCHAR

Value

`VARCHAR[]`

Overloads

- `parse_path(path = VARCHAR)`
- `parse_path(path = VARCHAR, separator = VARCHAR)`

SQL examples

```
parse_path('path/to/file.csv', 'system')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

percent_rank — *DuckDB function percent_rank*

Description

DuckDB function `percent_rank()`.

Usage

```
percent_rank()
```

Value

`DOUBLE`

pg_clear_cache — *DuckDB function pg_clear_cache*

Description

DuckDB function `pg_clear_cache()`.

Usage

```
pg_clear_cache()
```

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`pg_collation_is_visible`
DuckDB function pg_collation_is_visible

Description

DuckDB macro `pg_collation_is_visible()`.

Usage

```
pg_collation_is_visible(collation_oid)
```

Arguments

`collation_oid` Unspecified.

Value

Unspecified.

`pg_conf_load_time` *DuckDB function pg_conf_load_time*

Description

DuckDB macro `pg_conf_load_time()`.

Usage

```
pg_conf_load_time()
```

Value

Unspecified.

`pg_conversion_is_visible`
DuckDB function pg_conversion_is_visible

Description

DuckDB macro `pg_conversion_is_visible()`.

Usage

```
pg_conversion_is_visible(conversion_oid)
```

Arguments

`conversion_oid`
Unspecified.

Value

Unspecified.

`pg_function_is_visible`
DuckDB function pg_function_is_visible

Description

DuckDB macro `pg_function_is_visible()`.

Usage

```
pg_function_is_visible(function_oid)
```

Arguments

`function_oid` Unspecified.

Value

Unspecified.

`pg_get_constraintdef` *DuckDB function pg_get_constraintdef*

Description

DuckDB function `pg_get_constraintdef()`.

Usage

```
pg_get_constraintdef(constraint_oid, pretty_bool)
```

Arguments

`constraint_oid`, `pretty_bool`
: Unspecified.

Value

Unspecified.

Overloads

- `pg_get_constraintdef(constraint_oid)`
- `pg_get_constraintdef(constraint_oid, pretty_bool)`

`pg_get_expr` *DuckDB function pg_get_expr*

Description

DuckDB macro `pg_get_expr()`.

Usage

```
pg_get_expr(pg_node_tree, relation_oid)
```

Arguments

`pg_node_tree`, `relation_oid`
: Unspecified.

Value

Unspecified.

`pg_get_viewdef` *DuckDB function pg_get_viewdef*

Description

DuckDB macro `pg_get_viewdef()`.

Usage

```
pg_get_viewdef(oid)
```

Arguments

`oid`	Unspecified.

Value

Unspecified.

`pg_has_role` *DuckDB function pg_has_role*

Description

DuckDB function `pg_has_role()`.

Usage

```
pg_has_role(user, role, privilege)
```

Arguments

`user`, `role`, `privilege`	Unspecified.

Value

Unspecified.

Overloads

- `pg_has_role(user, role, privilege)`
- `pg_has_role(role, privilege)`

`pg_is_other_temp_schema` *DuckDB function pg_is_other_temp_schema*

Description

DuckDB macro `pg_is_other_temp_schema()`.

Usage

```
pg_is_other_temp_schema(schema_id)
```

Arguments

`schema_id`	Unspecified.

Value

Unspecified.

`pg_my_temp_schema` *DuckDB function pg_my_temp_schema*

Description

DuckDB macro `pg_my_temp_schema()`.

Usage

```
pg_my_temp_schema()
```

Value

Unspecified.

`pg_opclass_is_visible`

DuckDB function pg_opclass_is_visible

Description

DuckDB macro `pg_opclass_is_visible()`.

Usage

```
pg_opclass_is_visible(opclass_oid)
```

Arguments

`opclass_oid`	Unspecified.

Value

Unspecified.

`pg_operator_is_visible`

DuckDB function pg_operator_is_visible

Description

DuckDB macro `pg_operator_is_visible()`.

Usage

```
pg_operator_is_visible(operator_oid)
```

Arguments

`operator_oid`	Unspecified.

Value

Unspecified.

`pg_opfamily_is_visible`
DuckDB function pg_opfamily_is_visible

Description

DuckDB macro `pg_opfamily_is_visible()`.

Usage

```
pg_opfamily_is_visible(opclass_oid)
```

Arguments

`opclass_oid`	Unspecified.

Value

Unspecified.

`pg_postmaster_start_time`
DuckDB function pg_postmaster_start_time

Description

DuckDB macro `pg_postmaster_start_time()`.

Usage

```
pg_postmaster_start_time()
```

Value

Unspecified.

pg_size_pretty *DuckDB function pg_size_pretty*

Description

DuckDB macro `pg_size_pretty()`.

Usage

```
pg_size_pretty(bytes)
```

Arguments

`bytes`	Unspecified.

Value

Unspecified.

pg_sleep *DuckDB function pg_sleep*

Description

DuckDB macro `pg_sleep()`.

Usage

```
pg_sleep(seconds)
```

Arguments

`seconds`	Unspecified.

Value

Unspecified.

`pg_table_is_visible` *DuckDB function pg_table_is_visible*

Description

DuckDB macro `pg_table_is_visible()`.

Usage

```
pg_table_is_visible(table_oid)
```

Arguments

`table_oid`	Unspecified.

Value

Unspecified.

`pg_timezone_names` *DuckDB function pg_timezone_names*

Description

DuckDB function `pg_timezone_names()`.

Usage

```
pg_timezone_names()
```

Value

Unspecified.

Provided by

The `icu` extension (`LOAD icu;`).

`pg_ts_config_is_visible`

DuckDB function pg_ts_config_is_visible

Description

DuckDB macro `pg_ts_config_is_visible()`.

Usage

```
pg_ts_config_is_visible(config_oid)
```

Arguments

`config_oid`	Unspecified.

Value

Unspecified.

`pg_ts_dict_is_visible`

DuckDB function pg_ts_dict_is_visible

Description

DuckDB macro `pg_ts_dict_is_visible()`.

Usage

```
pg_ts_dict_is_visible(dict_oid)
```

Arguments

`dict_oid`	Unspecified.

Value

Unspecified.

`pg_ts_parser_is_visible`
DuckDB function pg_ts_parser_is_visible

Description

DuckDB macro `pg_ts_parser_is_visible()`.

Usage

```
pg_ts_parser_is_visible(parser_oid)
```

Arguments

`parser_oid`	Unspecified.

Value

Unspecified.

`pg_ts_template_is_visible`
DuckDB function pg_ts_template_is_visible

Description

DuckDB macro `pg_ts_template_is_visible()`.

Usage

```
pg_ts_template_is_visible(template_oid)
```

Arguments

`template_oid`	Unspecified.

Value

Unspecified.

pg_type_is_visible *DuckDB function pg_type_is_visible*

Description

DuckDB macro `pg_type_is_visible()`.

Usage

```
pg_type_is_visible(type_oid)
```

Arguments

`type_oid`	Unspecified.

Value

Unspecified.

pg_typeof *DuckDB function pg_typeof*

Description

DuckDB macro `pg_typeof()`.

Usage

```
pg_typeof(expression)
```

Arguments

`expression`	Unspecified.

Value

Unspecified.

pi *DuckDB function pi*

Description

Returns the value of pi.

Usage

```
pi()
```

Value

```
DOUBLE
```

SQL examples

```
pi()
```

platform *DuckDB function platform*

Description

DuckDB function `platform()`.

Usage

```
platform()
```

Value

Unspecified.

`postgres_attach` *DuckDB function postgres_attach*

Description

DuckDB function `postgres_attach()`.

Usage

```
postgres_attach(col0, suffix, sink_schema, filter_pushdown, source_schema, overwrite)
```

Arguments

`col0`	`VARCHAR`
`suffix`	`VARCHAR`
`sink_schema`	`VARCHAR`
`filter_pushdown`	`BOOLEAN`
`source_schema`	`VARCHAR`
`overwrite`	`BOOLEAN`

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_configure_pool`
DuckDB function postgres_configure_pool

Description

DuckDB function `postgres_configure_pool()`.

Usage

```
postgres_configure_pool(
  health_check_query,
  max_lifetime_millis,
  enable_reaper_thread,
  enable_thread_local_cache,
  wait_timeout_millis,
  max_connections,
  idle_timeout_millis,
  acquire_mode,
  catalog_name
)
```

Arguments

```
health_check_query
               VARCHAR
max_lifetime_millis
               UBIGINT
enable_reaper_thread
               BOOLEAN
enable_thread_local_cache
               BOOLEAN
wait_timeout_millis
               UBIGINT
max_connections
               UBIGINT
idle_timeout_millis
               UBIGINT
acquire_mode   VARCHAR
catalog_name   VARCHAR
```

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_execute` *DuckDB function postgres_execute*

Description

DuckDB function `postgres_execute()`.

Usage

```
postgres_execute(col0, col1, use_transaction)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`use_transaction`	`BOOLEAN`

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_hstore_get` *DuckDB function postgres_hstore_get*

Description

DuckDB function `postgres_hstore_get()`.

Usage

```
postgres_hstore_get(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_hstore_to_json` *DuckDB function postgres_hstore_to_json*

Description

DuckDB function `postgres_hstore_to_json()`.

Usage

```
postgres_hstore_to_json(col0)
```

Arguments

`col0`	`VARCHAR`

Value

`JSON`

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_query` *DuckDB function postgres_query*

Description

DuckDB function `postgres_query()`.

Usage

```
postgres_query(col0, col1, params, use_transaction)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`params`	`ANY`
`use_transaction`	`BOOLEAN`

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_scan` *DuckDB function postgres_scan*

Description

DuckDB function `postgres_scan()`.

Usage

```
postgres_scan(col0, col1, col2)
```

Arguments

`col0`, `col1`, `col2`
: `VARCHAR`

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`postgres_scan_pushdown` *DuckDB function postgres_scan_pushdown*

Description

DuckDB function `postgres_scan_pushdown()`.

Usage

```
postgres_scan_pushdown(col0, col1, col2)
```

Arguments

`col0`, `col1`, `col2`
: `VARCHAR`

Value

Unspecified.

Provided by

The `postgres_scanner` extension (`LOAD postgres_scanner;`).

`pow` *DuckDB function pow*

Description

Computes x to the power of y.

Usage

```
pow(x, y)
```

Arguments

x, y	DOUBLE

Value

`DOUBLE`

SQL examples

```
pow(2, 3)
power(2, 3)
2 ** 3
2 ^ 3
```

`pragma_collations` *DuckDB function pragma_collations*

Description

DuckDB function `pragma_collations()`.

Usage

```
pragma_collations()
```

Value

Unspecified.

`pragma_database_size` *DuckDB function pragma_database_size*

Description

DuckDB function `pragma_database_size()`.

Usage

```
pragma_database_size()
```

Value

Unspecified.

`pragma_hnsw_index_info`
DuckDB function pragma_hnsw_index_info

Description

DuckDB function `pragma_hnsw_index_info()`.

Usage

```
pragma_hnsw_index_info()
```

Value

Unspecified.

Provided by

The `vss` extension (`LOAD vss;`).

pragma_metadata_info *DuckDB function pragma_metadata_info*

Description

DuckDB function `pragma_metadata_info()`.

Usage

```
pragma_metadata_info(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Overloads

- `pragma_metadata_info()`
- `pragma_metadata_info(col0 = VARCHAR)`

pragma_platform *DuckDB function pragma_platform*

Description

DuckDB function `pragma_platform()`.

Usage

```
pragma_platform()
```

Value

Unspecified.

`pragma_rtree_index_info` *DuckDB function pragma_rtree_index_info*

Description

DuckDB function `pragma_rtree_index_info()`.

Usage

```
pragma_rtree_index_info()
```

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

`pragma_show` *DuckDB function pragma_show*

Description

DuckDB function `pragma_show()`.

Usage

```
pragma_show(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`pragma_storage_info` *DuckDB function pragma_storage_info*

Description

DuckDB function `pragma_storage_info()`.

Usage

```
pragma_storage_info(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`pragma_table_info` *DuckDB function pragma_table_info*

Description

DuckDB function `pragma_table_info()`.

Usage

```
pragma_table_info(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`pragma_user_agent` *DuckDB function pragma_user_agent*

Description

DuckDB function `pragma_user_agent()`.

Usage

```
pragma_user_agent()
```

Value

Unspecified.

`pragma_version` *DuckDB function pragma_version*

Description

DuckDB function `pragma_version()`.

Usage

```
pragma_version()
```

Value

Unspecified.

`prefix` *DuckDB function prefix*

Description

Returns `true` if `string` starts with `search_string`.

Usage

```
prefix(string, search_string)
```

Arguments

`string`, `search_string`
: VARCHAR

Value

BOOLEAN

SQL examples

```
prefix('abc', 'ab')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

printf *DuckDB function printf*

Description

Formats a `string` using printf syntax.

Usage

```
printf(format)
```

Arguments

format	VARCHAR

Value

VARCHAR

SQL examples

```
printf('Benchmark "%s" took %d seconds', 'CSV', 42)
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

product *DuckDB function product*

Description

Calculates the product of all tuples in arg.

Usage

```
product(arg)
```

Arguments

arg	DOUBLE

Value

DOUBLE

SQL examples

```
product(A)
```

---

quack_active_connections *DuckDB function quack_active_connections*

Description

DuckDB function quack_active_connections().

Usage

```
quack_active_connections()
```

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`quack_check_token` *DuckDB function quack_check_token*

Description

DuckDB function `quack_check_token()`.

Usage

```
quack_check_token(col0, col1, col2)
```

Arguments

`col0`, `col1`, `col2`	`VARCHAR`

Value

`BOOLEAN`

Provided by

The `quack` extension (`LOAD quack;`).

`quack_clear_cache` *DuckDB function quack_clear_cache*

Description

DuckDB function `quack_clear_cache()`.

Usage

```
quack_clear_cache()
```

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`quack_identify` *DuckDB function quack_identify*

Description

DuckDB function `quack_identify()`.

Usage

```
quack_identify(meta, region, hostname, provider, name)
```

Arguments

`meta`, `region`, `hostname`, `provider`, `name`	`VARCHAR`

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`quack_nop_authorization` *DuckDB function quack_nop_authorization*

Description

DuckDB function `quack_nop_authorization()`.

Usage

```
quack_nop_authorization(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `quack` extension (`LOAD quack;`).

`quack_query` *DuckDB function quack_query*

Description

DuckDB function `quack_query()`.

Usage

```
quack_query(col0, col1, token, disable_ssl)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`token`	`VARCHAR`
`disable_ssl`	`BOOLEAN`

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`quack_query_by_name` *DuckDB function quack_query_by_name*

Description

DuckDB function `quack_query_by_name()`.

Usage

```
quack_query_by_name(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

quack_serve *DuckDB function quack_serve*

Description

DuckDB function `quack_serve()`.

Usage

```
quack_serve(col0, token, allow_other_hostname, disable_ssl)
```

Arguments

col0	VARCHAR
token	VARCHAR
allow_other_hostname	BOOLEAN
disable_ssl	BOOLEAN

Value

Unspecified.

Overloads

- `quack_serve(col0 = VARCHAR, token = VARCHAR, allow_other_hostname = BOOLEAN, disable_ssl = BOOLEAN)`
- `quack_serve(token = VARCHAR, allow_other_hostname = BOOLEAN, disable_ssl = BOOLEAN)`

Provided by

The `quack` extension (`LOAD quack;`).

quack_server_list *DuckDB function quack_server_list*

Description

DuckDB function `quack_server_list()`.

Usage

```
quack_server_list()
```

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`quack_stop` *DuckDB function quack_stop*

Description

DuckDB function `quack_stop()`.

Usage

```
quack_stop(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`quack_uri_parser` *DuckDB function quack_uri_parser*

Description

DuckDB function `quack_uri_parser()`.

Usage

```
quack_uri_parser(col0, col1)
```

Arguments

`col0`	`VARCHAR`
`col1`	`BOOLEAN`

Value

```
STRUCT(host VARCHAR, port USMALLINT, ipv6 BOOLEAN, ssl BOOLEAN, url VARCHAR)
```

Provided by

The `quack` extension (`LOAD quack;`).

`quantile_cont` *DuckDB function quantile_cont*

Description

Returns the interpolated quantile number between 0 and 1 . If pos is a LIST of FLOATs, then the result is a LIST of the corresponding interpolated quantiles. .

Usage

```
quantile_cont(x, pos)
```

Arguments

x	DECIMAL | TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DAT
pos	DOUBLE | DOUBLE[]

Value

```
DECIMAL | TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DATE | TIMESTAMP
```

SQL examples

```
quantile_cont(x, 0.5)
```

`quantile_disc` *DuckDB function quantile_disc*

Description

Returns the exact quantile number between 0 and 1 . If pos is a LIST of FLOATs, then the result is a LIST of the corresponding exact quantiles.

Usage

```
quantile_disc(x, pos)
```

Arguments

x	ANY
pos	DOUBLE | DOUBLE[]

Value

ANY

Overloads

- quantile_disc(x = ANY, pos = DOUBLE)
- quantile_disc(x = ANY, pos = `DOUBLE[]`)
- quantile_disc(x = ANY)

SQL examples

```
quantile_disc(x, 0.5)
```

---

quarter *DuckDB function quarter*

Description

Extract the quarter component from a date or timestamp.

Usage

```
quarter(ts)
```

Arguments

ts	DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE

Value

BIGINT

SQL examples

```
quarter(timestamp '2021-08-03 11:59:44.123456')
```

`query` *DuckDB function query*

Description

DuckDB function `query()`.

Usage

```
query(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`query_table` *DuckDB function query_table*

Description

DuckDB function `query_table()`.

Usage

```
query_table(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

Overloads

- `query_table(col0 = VARCHAR)`
- `query_table(col0 = \`VARCHAR[]\`)`
- `query_table(col0 = \`VARCHAR[]\`, col1 = BOOLEAN)`

r_dataframe_scan *DuckDB function r_dataframe_scan*

Description

DuckDB function `r_dataframe_scan()`.

Usage

```
r_dataframe_scan(col0, map_list_of, experimental, integer64)
```

Arguments

col0	POINTER
map_list_of	BOOLEAN
experimental	BOOLEAN
integer64	BOOLEAN

Value

Unspecified.

radians *DuckDB function radians*

Description

Converts degrees to radians.

Usage

```
radians(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
radians(90)
```

`random` *DuckDB function random*

Description

Returns a random number between 0 and 1.

Usage

```
random()
```

Value

`DOUBLE`

SQL examples

```
random()
```

`range` *DuckDB function range*

Description

Creates a list of values between `start` and `stop` - the stop parameter is exclusive.

Usage

```
range(start, stop, step)
```

Arguments

`start`	`BIGINT | TIMESTAMP | TIMESTAMP WITH TIME ZONE`
`stop`	`BIGINT | TIMESTAMP | TIMESTAMP WITH TIME ZONE`
`step`	`BIGINT | INTERVAL`

Value

`BIGINT[] | TIMESTAMP[] | TIMESTAMP WITH TIME ZONE[]`

Overloads

- `range(col0 = BIGINT)`
- `range(col0 = BIGINT, col1 = BIGINT)`
- `range(col0 = BIGINT, col1 = BIGINT, col2 = BIGINT)`
- `range(col0 = TIMESTAMP, col1 = TIMESTAMP, col2 = INTERVAL)`
- `range(col0 = `TIMESTAMP WITH TIME ZONE`, col1 = `TIMESTAMP WITH TIME ZONE`, col2 = INTERVAL)`
- `range(start = BIGINT)`
- `range(start = BIGINT, stop = BIGINT)`
- `range(start = BIGINT, stop = BIGINT, step = BIGINT)`
- `range(start = TIMESTAMP, stop = TIMESTAMP, step = INTERVAL)`
- `range(start = `TIMESTAMP WITH TIME ZONE`, stop = `TIMESTAMP WITH TIME ZONE`, step = INTERVAL)`

Provided by

- `range(col0 = BIGINT)`: built in
- `range(col0 = BIGINT, col1 = BIGINT)`: built in
- `range(col0 = BIGINT, col1 = BIGINT, col2 = BIGINT)`: built in
- `range(col0 = TIMESTAMP, col1 = TIMESTAMP, col2 = INTERVAL)`: built in
- `range(col0 = `TIMESTAMP WITH TIME ZONE`, col1 = `TIMESTAMP WITH TIME ZONE`, col2 = INTERVAL)`: `icu`
- `range(start = BIGINT)`: built in
- `range(start = BIGINT, stop = BIGINT)`: built in
- `range(start = BIGINT, stop = BIGINT, step = BIGINT)`: built in
- `range(start = TIMESTAMP, stop = TIMESTAMP, step = INTERVAL)`: built in
- `range(start = `TIMESTAMP WITH TIME ZONE`, stop = `TIMESTAMP WITH TIME ZONE`, step = INTERVAL)`: built in

SQL examples

```
range(2, 5, 3)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `repeat()`, `unpivot_list()`

`rank` *DuckDB function rank*

Description

DuckDB function `rank()`.

Usage

```
rank()
```

Value

`BIGINT`

`rank_dense` *DuckDB function rank_dense*

Description

DuckDB function `rank_dense()`.

Usage

```
rank_dense()
```

Value

`BIGINT`

`read_avro` *DuckDB function read_avro*

Description

DuckDB function `read_avro()`.

Usage

```
read_avro(
  col0,
  hive_types_autocast,
  hive_types,
  hive_partitioning,
  union_by_name,
  filename
)
```

Arguments

col0	VARCHAR | VARCHAR[]
hive_types_autocast	BOOLEAN
hive_types	ANY
hive_partitioning	BOOLEAN
union_by_name	BOOLEAN
filename	ANY

Value

Unspecified.

Provided by

The `avro` extension (`LOAD avro;`).

`read_blob` *DuckDB function read_blob*

Description

DuckDB function `read_blob()`.

Usage

```
read_blob(col0)
```

Arguments

col0	VARCHAR | VARCHAR[]

Value

Unspecified.

`read_csv` *DuckDB function read_csv*

Description

DuckDB function `read_csv()`.

Usage

```
read_csv(
  col0,
  thousands,
  strict_mode,
  dtypes,
  column_types,
  null_padding,
  column_names,
  buffer_size,
  parallel,
  force_not_null,
  hive_types,
  new_line,
  files_to_sniff,
  dateformat,
  delim,
  sep,
  decimal_separator,
  nullstr,
  escape,
  compression,
  encoding,
  hive_types_autocast,
  all_varchar,
  columns,
  hive_partitioning,
  auto_detect,
  comment,
  quote,
  max_line_size,
  store_rejects,
  union_by_name,
  header,
  types,
  skip,
  filename,
  sample_size,
  timestampformat,
```

```
  normalize_names,
  ignore_errors,
  names,
  allow_quoted_nulls,
  maximum_line_size,
  rejects_table,
  auto_type_candidates,
  rejects_scan,
  rejects_limit
)
```

Arguments

col0	VARCHAR | VARCHAR[]
thousands	VARCHAR
strict_mode	BOOLEAN
dtypes	ANY
column_types	ANY
null_padding	BOOLEAN
column_names	VARCHAR[]
buffer_size	UBIGINT
parallel	BOOLEAN
force_not_null	VARCHAR[]
hive_types	ANY
new_line	VARCHAR
files_to_sniff	BIGINT
dateformat	VARCHAR
delim	VARCHAR
sep	VARCHAR
decimal_separator	VARCHAR
nullstr	ANY
escape	VARCHAR
compression	VARCHAR
encoding	VARCHAR
hive_types_autocast	BOOLEAN
all_varchar	BOOLEAN
columns	ANY
hive_partitioning	BOOLEAN

auto_detect	BOOLEAN
comment	VARCHAR
quote	VARCHAR
max_line_size	VARCHAR
store_rejects	BOOLEAN
union_by_name	BOOLEAN
header	BOOLEAN
types	ANY
skip	BIGINT
filename	ANY
sample_size	BIGINT
timestampformat	VARCHAR
normalize_names	BOOLEAN
ignore_errors	BOOLEAN
names	VARCHAR[]
allow_quoted_nulls	BOOLEAN
maximum_line_size	VARCHAR
rejects_table	VARCHAR
auto_type_candidates	ANY
rejects_scan	VARCHAR
rejects_limit	BIGINT

Value

Unspecified.

`read_csv_auto` *DuckDB function read_csv_auto*

Description

DuckDB function `read_csv_auto()`.

Usage

```
read_csv_auto(
  col0,
  thousands,
  strict_mode,
  dtypes,
  column_types,
  null_padding,
  column_names,
  buffer_size,
  parallel,
  force_not_null,
  hive_types,
  new_line,
  files_to_sniff,
  dateformat,
  delim,
  sep,
  decimal_separator,
  nullstr,
  escape,
  compression,
  encoding,
  hive_types_autocast,
  all_varchar,
  columns,
  hive_partitioning,
  auto_detect,
  comment,
  quote,
  max_line_size,
  store_rejects,
  union_by_name,
  header,
  types,
  skip,
  filename,
  sample_size,
  timestampformat,
  normalize_names,
  ignore_errors,
  names,
  allow_quoted_nulls,
  maximum_line_size,
  rejects_table,
  auto_type_candidates,
  rejects_scan,
  rejects_limit
```

```
)
```

Arguments

col0	VARCHAR | VARCHAR[]
thousands	VARCHAR
strict_mode	BOOLEAN
dtypes	ANY
column_types	ANY
null_padding	BOOLEAN
column_names	VARCHAR[]
buffer_size	UBIGINT
parallel	BOOLEAN
force_not_null	VARCHAR[]
hive_types	ANY
new_line	VARCHAR
files_to_sniff	BIGINT
dateformat	VARCHAR
delim	VARCHAR
sep	VARCHAR
decimal_separator	VARCHAR
nullstr	ANY
escape	VARCHAR
compression	VARCHAR
encoding	VARCHAR
hive_types_autocast	BOOLEAN
all_varchar	BOOLEAN
columns	ANY
hive_partitioning	BOOLEAN
auto_detect	BOOLEAN
comment	VARCHAR
quote	VARCHAR
max_line_size	VARCHAR
store_rejects	BOOLEAN
union_by_name	BOOLEAN

```
header          BOOLEAN
types           ANY
skip            BIGINT
filename        ANY
sample_size     BIGINT
timestampformat
                VARCHAR
normalize_names
                BOOLEAN
ignore_errors   BOOLEAN
names           VARCHAR[]
allow_quoted_nulls
                BOOLEAN
maximum_line_size
                VARCHAR
rejects_table   VARCHAR
auto_type_candidates
                ANY
rejects_scan    VARCHAR
rejects_limit   BIGINT
```

Value

Unspecified.

`read_duckdb` *DuckDB function read_duckdb*

Description

DuckDB function `read_duckdb()`.

Usage

```
read_duckdb(
  col0,
  table_name,
  schema_name,
  hive_types_autocast,
  hive_types,
  hive_partitioning,
  union_by_name,
  filename
)
```

Arguments

col0	VARCHAR | VARCHAR[]
table_name	VARCHAR
schema_name	VARCHAR
hive_types_autocast	BOOLEAN
hive_types	ANY
hive_partitioning	BOOLEAN
union_by_name	BOOLEAN
filename	ANY

Value

Unspecified.

---

read_json *DuckDB function read_json*

Description

DuckDB function `read_json()`.

Usage

```
read_json(
  col0,
  convert_strings_to_integers,
  maximum_sample_files,
  timestamp_format,
  field_appearance_threshold,
  timestampformat,
  map_inference_threshold,
  date_format,
  filename,
  union_by_name,
  compression,
  maximum_depth,
  columns,
  sample_size,
  hive_types,
  hive_types_autocast,
  maximum_object_size,
  format,
  ignore_errors,
```

```
  hive_partitioning,
  auto_detect,
  records,
  dateformat
)
```

Arguments

col0	VARCHAR | VARCHAR[]
convert_strings_to_integers	BOOLEAN
maximum_sample_files	BIGINT
timestamp_format	VARCHAR
field_appearance_threshold	DOUBLE
timestampformat	VARCHAR
map_inference_threshold	BIGINT
date_format	VARCHAR
filename	ANY
union_by_name	BOOLEAN
compression	VARCHAR
maximum_depth	BIGINT
columns	ANY
sample_size	BIGINT
hive_types	ANY
hive_types_autocast	BOOLEAN
maximum_object_size	UINTEGER
format	VARCHAR
ignore_errors	BOOLEAN
hive_partitioning	BOOLEAN
auto_detect	BOOLEAN
records	VARCHAR
dateformat	VARCHAR

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_json_auto` *DuckDB function read_json_auto*

Description

DuckDB function `read_json_auto()`.

Usage

```
read_json_auto(
  col0,
  convert_strings_to_integers,
  maximum_sample_files,
  timestamp_format,
  field_appearance_threshold,
  timestampformat,
  map_inference_threshold,
  date_format,
  filename,
  union_by_name,
  compression,
  maximum_depth,
  columns,
  sample_size,
  hive_types,
  hive_types_autocast,
  maximum_object_size,
  format,
  ignore_errors,
  hive_partitioning,
  auto_detect,
  records,
  dateformat
)
```

Arguments

```
col0                VARCHAR | VARCHAR[]
convert_strings_to_integers
                    BOOLEAN
maximum_sample_files
                    BIGINT
timestamp_format
                    VARCHAR
field_appearance_threshold
                    DOUBLE
```

```
timestampformat
               VARCHAR
map_inference_threshold
               BIGINT
date_format    VARCHAR
filename       ANY
union_by_name  BOOLEAN
compression    VARCHAR
maximum_depth  BIGINT
columns        ANY
sample_size    BIGINT
hive_types     ANY
hive_types_autocast
               BOOLEAN
maximum_object_size
               UINTEGER
format         VARCHAR
ignore_errors  BOOLEAN
hive_partitioning
               BOOLEAN
auto_detect    BOOLEAN
records        VARCHAR
dateformat     VARCHAR
```

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_json_objects` *DuckDB function read_json_objects*

Description

DuckDB function `read_json_objects()`.

Usage

```
read_json_objects(
  col0,
  ignore_errors,
  format,
  maximum_object_size,
  hive_types_autocast,
  hive_types,
  hive_partitioning,
  compression,
  union_by_name,
  filename
)
```

Arguments

col0	VARCHAR | VARCHAR[]
ignore_errors	BOOLEAN
format	VARCHAR
maximum_object_size	UINTEGER
hive_types_autocast	BOOLEAN
hive_types	ANY
hive_partitioning	BOOLEAN
compression	VARCHAR
union_by_name	BOOLEAN
filename	ANY

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_json_objects_auto`
DuckDB function read_json_objects_auto

Description

DuckDB function `read_json_objects_auto()`.

Usage

```
read_json_objects_auto(
  col0,
  ignore_errors,
  format,
  maximum_object_size,
  hive_types_autocast,
  hive_types,
  hive_partitioning,
  compression,
  union_by_name,
  filename
)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`
`ignore_errors`	`BOOLEAN`
`format`	`VARCHAR`
`maximum_object_size`	`UINTEGER`
`hive_types_autocast`	`BOOLEAN`
`hive_types`	`ANY`
`hive_partitioning`	`BOOLEAN`
`compression`	`VARCHAR`
`union_by_name`	`BOOLEAN`
`filename`	`ANY`

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_ndjson` *DuckDB function read_ndjson*

Description

DuckDB function `read_ndjson()`.

Usage

```
read_ndjson(
  col0,
  convert_strings_to_integers,
  maximum_sample_files,
  timestamp_format,
  field_appearance_threshold,
  timestampformat,
  map_inference_threshold,
  date_format,
  filename,
  union_by_name,
  compression,
  maximum_depth,
  columns,
  sample_size,
  hive_types,
  hive_types_autocast,
  maximum_object_size,
  format,
  ignore_errors,
  hive_partitioning,
  auto_detect,
  records,
  dateformat
)
```

Arguments

```
col0            VARCHAR | VARCHAR[]
convert_strings_to_integers
                BOOLEAN
maximum_sample_files
                BIGINT
timestamp_format
                VARCHAR
field_appearance_threshold
                DOUBLE
```

```
timestampformat
               VARCHAR
map_inference_threshold
               BIGINT
date_format    VARCHAR
filename       ANY
union_by_name  BOOLEAN
compression    VARCHAR
maximum_depth  BIGINT
columns        ANY
sample_size    BIGINT
hive_types     ANY
hive_types_autocast
               BOOLEAN
maximum_object_size
               UINTEGER
format         VARCHAR
ignore_errors  BOOLEAN
hive_partitioning
               BOOLEAN
auto_detect    BOOLEAN
records        VARCHAR
dateformat     VARCHAR
```

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_ndjson_auto` *DuckDB function read_ndjson_auto*

Description

DuckDB function `read_ndjson_auto()`.

Usage

```
read_ndjson_auto(
  col0,
  convert_strings_to_integers,
  maximum_sample_files,
  timestamp_format,
  field_appearance_threshold,
  timestampformat,
  map_inference_threshold,
  date_format,
  filename,
  union_by_name,
  compression,
  maximum_depth,
  columns,
  sample_size,
  hive_types,
  hive_types_autocast,
  maximum_object_size,
  format,
  ignore_errors,
  hive_partitioning,
  auto_detect,
  records,
  dateformat
)
```

Arguments

```
col0            VARCHAR | VARCHAR[]
convert_strings_to_integers
                BOOLEAN
maximum_sample_files
                BIGINT
timestamp_format
                VARCHAR
field_appearance_threshold
                DOUBLE
timestampformat
                VARCHAR
map_inference_threshold
                BIGINT
date_format     VARCHAR
filename        ANY
union_by_name   BOOLEAN
compression     VARCHAR
maximum_depth   BIGINT
```

```
columns         ANY
sample_size     BIGINT
hive_types      ANY
hive_types_autocast
                BOOLEAN
maximum_object_size
                UINTEGER
format          VARCHAR
ignore_errors   BOOLEAN
hive_partitioning
                BOOLEAN
auto_detect     BOOLEAN
records         VARCHAR
dateformat      VARCHAR
```

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_ndjson_objects` *DuckDB function read_ndjson_objects*

Description

DuckDB function `read_ndjson_objects()`.

Usage

```
read_ndjson_objects(
  col0,
  ignore_errors,
  format,
  maximum_object_size,
  hive_types_autocast,
  hive_types,
  hive_partitioning,
  compression,
  union_by_name,
  filename
)
```

Arguments

col0	VARCHAR | VARCHAR[]
ignore_errors	BOOLEAN
format	VARCHAR
maximum_object_size	UINTEGER
hive_types_autocast	BOOLEAN
hive_types	ANY
hive_partitioning	BOOLEAN
compression	VARCHAR
union_by_name	BOOLEAN
filename	ANY

Value

Unspecified.

Provided by

The `json` extension (`LOAD json;`).

`read_parquet` *DuckDB function read_parquet*

Description

DuckDB function `read_parquet()`.

Usage

```
read_parquet(
  col0,
  can_have_nan,
  filename,
  union_by_name,
  debug_use_openssl,
  hive_partitioning,
  parquet_version,
  encryption_config,
  hive_types_autocast,
  binary_as_string,
  explicit_cardinality,
  compression,
```

```
  file_row_number,
  hive_types,
  schema
)
```

Arguments

col0	VARCHAR | VARCHAR[]
can_have_nan	BOOLEAN
filename	ANY
union_by_name	BOOLEAN
debug_use_openssl	BOOLEAN
hive_partitioning	BOOLEAN
parquet_version	VARCHAR
encryption_config	ANY
hive_types_autocast	BOOLEAN
binary_as_string	BOOLEAN
explicit_cardinality	UBIGINT
compression	VARCHAR
file_row_number	BOOLEAN
hive_types	ANY
schema	ANY

Value

Unspecified.

`read_postgres_binary` *DuckDB function read_postgres_binary*

Description

DuckDB function `read_postgres_binary()`.

Usage

```
read_postgres_binary(col0, buffer_size, columns)
```

Arguments

col0	VARCHAR
buffer_size	UBIGINT
columns	ANY

Value

Unspecified.

Provided by

The postgres_scanner extension (LOAD postgres_scanner;).

---

read_text *DuckDB function read_text*

Description

DuckDB function read_text().

Usage

```
read_text(col0)
```

Arguments

col0	VARCHAR | VARCHAR[]

Value

Unspecified.

---

read_vortex *DuckDB function read_vortex*

Description

DuckDB function read_vortex().

Usage

```
read_vortex(col0)
```

Arguments

col0	VARCHAR | VARCHAR[]

Value

Unspecified.

Provided by

The `vortex` extension (`LOAD vortex;`).

`read_xlsx` *DuckDB function read_xlsx*

Description

DuckDB function `read_xlsx()`.

Usage

```
read_xlsx(
  col0,
  normalize_names,
  empty_as_varchar,
  stop_at_empty,
  sheet,
  range,
  ignore_errors,
  all_varchar,
  header
)
```

Arguments

`col0`	`VARCHAR`
`normalize_names`	`BOOLEAN`
`empty_as_varchar`	`BOOLEAN`
`stop_at_empty`	`BOOLEAN`
`sheet`	`VARCHAR`
`range`	`VARCHAR`
`ignore_errors`	`BOOLEAN`
`all_varchar`	`BOOLEAN`
`header`	`BOOLEAN`

Value

Unspecified.

Provided by

The `excel` extension (`LOAD excel;`).

`regexp_escape` *DuckDB function regexp_escape*

Description

Escapes special patterns to turn `string` into a regular expression similarly to Python's `re.escape` function.

Usage

```
regexp_escape(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
regexp_escape('https://duckdb.org')
```

See Also

Other regex: `regexp_extract()`, `regexp_extract_all()`, `regexp_full_match()`, `regexp_matches()`, `regexp_replace()`, `string_split_regex()`

`regexp_extract` *DuckDB function regexp_extract*

Description

- `regexp_extract(string = VARCHAR, regex = VARCHAR)`, `regexp_extract(string = VARCHAR, regex = VARCHAR, group = INTEGER)`, `regexp_extract(string = VARCHAR, regex = VARCHAR, group = INTEGER, options = VARCHAR)`: If `string` contains the `regex` pattern, returns the capturing group specified by optional parameter `group`; otherwise, returns the empty string. The `group` must be a constant value. If no `group` is given, it defaults to 0. A set of optional regex `options` can be set.
- `regexp_extract(string = VARCHAR, regex = VARCHAR, name_list = \`VARCHAR[]\`)`, `regexp_extract(string = VARCHAR, regex = VARCHAR, name_list = \`VARCHAR[]\`, options = VARCHAR)`: If `string` contains the `regex` pattern, returns the capturing groups as a struct with corresponding names from `name_list`; otherwise, returns a struct with the same keys and empty strings as values. A set of optional regex `options` can be set.

Usage

```
regexp_extract(string, regex, group, options)
```

Arguments

`string`	`VARCHAR`
`regex`	`VARCHAR`
`group`	`INTEGER`
`options`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `regexp_extract(string = VARCHAR, regex = VARCHAR)`
- `regexp_extract(string = VARCHAR, regex = VARCHAR, group = INTEGER)`
- `regexp_extract(string = VARCHAR, regex = VARCHAR, group = INTEGER, options = VARCHAR)`
- `regexp_extract(string = VARCHAR, regex = VARCHAR, name_list = ` `` `VARCHAR[]` `` `)`
- `regexp_extract(string = VARCHAR, regex = VARCHAR, name_list = ` `` `VARCHAR[]` `` `, options = VARCHAR)`

SQL examples

```
regexp_extract('abcde', '[a-z]{3}')
regexp_extract('abc', '([a-z])(b)', 1)
regexp_extract('ABC', '([a-z])(b)', 1, 'i')
regexp_extract('2023-04-15', '(\d+)-(\d+)-(\d+)', ['y', 'm', 'd'])
regexp_extract('John Doe', '([a-z]+) ([a-z]+)', ['first_name', 'last_name'], 'i')
```

See Also

Other regex: `regexp_escape()`, `regexp_extract_all()`, `regexp_full_match()`, `regexp_matches()`, `regexp_replace()`, `string_split_regex()`

`regexp_extract_all` *DuckDB function regexp_extract_all*

Description

Finds non-overlapping occurrences of the `regex` in the `string` and returns the corresponding values of the capturing `group`. A set of optional regex `options` can be set.

Usage

```
regexp_extract_all(string, regex, group, options)
```

Arguments

`string`	`VARCHAR`
`regex`	`VARCHAR`
`group`	`INTEGER | VARCHAR[]`
`options`	`VARCHAR`

Value

`VARCHAR[]`

Overloads

- `regexp_extract_all(string = VARCHAR, regex = VARCHAR)`
- `regexp_extract_all(string = VARCHAR, regex = VARCHAR, group = INTEGER)`
- `regexp_extract_all(string = VARCHAR, regex = VARCHAR, group = INTEGER, options = VARCHAR)`
- `regexp_extract_all(string = VARCHAR, regex = VARCHAR, group = ``VARCHAR[]``)`
- `regexp_extract_all(string = VARCHAR, regex = VARCHAR, group = ``VARCHAR[]``, options = VARCHAR)`

SQL examples

```
regexp_extract_all('Peter: 33, Paul:14', '(\w+):\s*(\d+)', 2)
```

See Also

Other regex: `regexp_escape()`, `regexp_extract()`, `regexp_full_match()`, `regexp_matches()`, `regexp_replace()`, `string_split_regex()`

`regexp_full_match` *DuckDB function regexp_full_match*

Description

Returns `true` if the entire `string` matches the `regex`. A set of optional regex `options` can be set.

Usage

```
regexp_full_match(string, regex, col2)
```

Arguments

`string`, `regex`, `col2`	`VARCHAR`

Value

`BOOLEAN`

Overloads

- `regexp_full_match(string = VARCHAR, regex = VARCHAR)`
- `regexp_full_match(string = VARCHAR, regex = VARCHAR, col2 = VARCHAR)`

SQL examples

```
regexp_full_match('anabanana', '(an)*')
```

See Also

Other regex: `regexp_escape()`, `regexp_extract()`, `regexp_extract_all()`, `regexp_matches()`, `regexp_replace()`, `string_split_regex()`

`regexp_matches` *DuckDB function regexp_matches*

Description

Returns `true` if `string` contains the `regex`, `false` otherwise. A set of optional regex `options` can be set.

Usage

```
regexp_matches(string, regex, options)
```

Arguments

string, regex, options
VARCHAR

Value

BOOLEAN

Overloads

- regexp_matches(string = VARCHAR, regex = VARCHAR)
- regexp_matches(string = VARCHAR, regex = VARCHAR, options = VARCHAR)

SQL examples

```
regexp_matches('anabanana', '(an)*')
```

See Also

Other regex: regexp_escape(), regexp_extract(), regexp_extract_all(), regexp_full_match(), regexp_replace(), string_split_regex()

---

regexp_replace *DuckDB function regexp_replace*

Description

If string contains the regex, replaces the matching part with replacement. A set of optional regex options can be set.

Usage

```
regexp_replace(string, regex, replacement, options)
```

Arguments

string, regex, replacement, options
VARCHAR

Value

VARCHAR

Overloads

- regexp_replace(string = VARCHAR, regex = VARCHAR, replacement = VARCHAR)
- regexp_replace(string = VARCHAR, regex = VARCHAR, replacement = VARCHAR, options = VARCHAR)

SQL examples

```
regexp_replace('hello', '[lo]', '-')
```

See Also

Other regex: `regexp_escape()`, `regexp_extract()`, `regexp_extract_all()`, `regexp_full_match()`, `regexp_matches()`, `string_split_regex()`

`regexp_split_to_table` *DuckDB function regexp_split_to_table*

Description

DuckDB macro `regexp_split_to_table()`.

Usage

```
regexp_split_to_table(text, pattern)
```

Arguments

`text`, `pattern`	Unspecified.

Value

Unspecified.

`regr_avgx` *DuckDB function regr_avgx*

Description

Returns the average of the independent variable for non-NULL pairs in a group, where x is the independent variable and y is the dependent variable.

Usage

```
regr_avgx(y, x)
```

Arguments

`y`, `x`	`DOUBLE`

Value

`DOUBLE`

`regr_avgy` *DuckDB function regr_avgy*

Description

Returns the average of the dependent variable for non-NULL pairs in a group, where x is the independent variable and y is the dependent variable.

Usage

```
regr_avgy(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

`regr_count` *DuckDB function regr_count*

Description

Returns the number of non-NULL number pairs in a group.

Usage

```
regr_count(y, x)
```

Arguments

y, x	DOUBLE

Value

UINTEGER

SQL examples

```
(SUM(x*y) - SUM(x) * SUM(y) / COUNT(*)) / COUNT(*)
```

`regr_intercept` *DuckDB function regr_intercept*

Description

Returns the intercept of the univariate linear regression line for non-NULL pairs in a group.

Usage

```
regr_intercept(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
AVG(y)-REGR_SLOPE(y, x)*AVG(x)
```

`regr_r2` *DuckDB function regr_r2*

Description

Returns the coefficient of determination for non-NULL pairs in a group.

Usage

```
regr_r2(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

regr_slope — *DuckDB function regr_slope*

Description

Returns the slope of the linear regression line for non-NULL pairs in a group.

Usage

```
regr_slope(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
COVAR_POP(x, y) / VAR_POP(x)
```

regr_sxx — *DuckDB function regr_sxx*

Description

DuckDB function `regr_sxx()`.

Usage

```
regr_sxx(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
REGR_COUNT(y, x) * VAR_POP(x)
```

`regr_sxy` *DuckDB function regr_sxy*

Description

Returns the population covariance of input values.

Usage

```
regr_sxy(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
REGR_COUNT(y, x) * COVAR_POP(y, x)
```

`regr_syy` *DuckDB function regr_syy*

Description

DuckDB function `regr_syy()`.

Usage

```
regr_syy(y, x)
```

Arguments

y, x	DOUBLE

Value

DOUBLE

SQL examples

```
REGR_COUNT(y, x) * VAR_POP(y)
```

`remap_struct` *DuckDB function remap_struct*

Description

Map a struct to another struct type, potentially re-ordering, renaming and casting members and filling in defaults for missing values.

Usage

```
remap_struct(input, target_type, mapping, defaults)
```

Arguments

`input`, `target_type`, `mapping`, `defaults`	`ANY`

Value

`ANY`

SQL examples

```
remap_struct({'i': 1, 'j': 2}, NULL::ROW(v1 INT, v2 INT, v3 INT), {'v1': 'j', 'v3': 'i'}, {'v2
```

`remove_iceberg_schema_properties`
DuckDB function remove_iceberg_schema_properties

Description

DuckDB function `remove_iceberg_schema_properties()`.

Usage

```
remove_iceberg_schema_properties(col0, col1)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR[]`

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`remove_iceberg_table_properties`
DuckDB function remove_iceberg_table_properties

Description

DuckDB function `remove_iceberg_table_properties()`.

Usage

```
remove_iceberg_table_properties(col0, col1)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR[]`

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`repeat` *DuckDB function repeat*

Description

- `` `repeat`(string = VARCHAR, count = BIGINT) ``: Repeats the `string` `count` number of times.
- `` `repeat`(blob = BLOB, count = BIGINT) ``: Repeats the `blob` `count` number of times.
- `` `repeat`(list = `T[]`, count = BIGINT) ``: Repeats the `list` `count` number of times.
- `` `repeat`(col0 = ANY, col1 = BIGINT) ``: (Description missing.)

Usage

```
`repeat`(string, count)
```

Arguments

`string`	`VARCHAR`
`count`	`BIGINT`

Value

VARCHAR | BLOB | T[]

Overloads

- `repeat`(col0 = ANY, col1 = BIGINT)
- `repeat`(string = VARCHAR, count = BIGINT)
- `repeat`(blob = BLOB, count = BIGINT)
- `repeat`(list = `T[]`, count = BIGINT)

SQL examples

```
repeat('A', 5)
repeat('\xAA\xBB'::BLOB, 5)
repeat([1, 2, 3], 5)
```

See Also

Other blob: decode(), encode(), from_base64(), hex(), md5(), md5_number(), octet_length(), or--or, sha1(), sha256(), to_base64(), unbin(), unhex()

Other list: array_extract(), array_length(), concat(), contains(), flatten(), generate_series(), len(), list_aggregate(), list_concat(), list_contains(), list_cosine_distance(), list_cosine_similarity(), list_distance(), list_distinct(), list_extract(), list_filter(), list_grade_up(), list_has_all(), list_has_any(), list_inner_product(), list_intersect(), list_negative_inner_product(), list_position(), list_reduce(), list_resize(), list_reverse_sort(), list_select(), list_slice(), list_sort(), list_transform(), list_unique(), list_value(), list_where(), list_zip(), or--or, range(), unpivot_list()

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

repeat_row *DuckDB function repeat_row*

Description

DuckDB function repeat_row().

Usage

```
repeat_row(num_rows)
```

Arguments

num_rows	BIGINT

Value

Unspecified.

`replace` *DuckDB function replace*

Description

Replaces any occurrences of the `source` with `target` in `string`.

Usage

```
replace(string, source, target)
```

Arguments

string, source, target	VARCHAR

Value

VARCHAR

SQL examples

```
replace('hello', 'l', '-')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`replace_type` *DuckDB function replace_type*

Description

Casts all fields of type1 to type2.

Usage

```
replace_type(param, type1, type2)
```

Arguments

param, type1, type2	ANY

Value

ANY

SQL examples

```
replace_type({duck: 3.141592653589793::DOUBLE}, NULL::DOUBLE, NULL::DECIMAL(15,2))
```

`reservoir_quantile` *DuckDB function reservoir_quantile*

Description

Gives the approximate quantile using reservoir sampling, the sample size is optional and uses 8192 as a default size.

Usage

```
reservoir_quantile(x, quantile, sample_size)
```

Arguments

x	DECIMAL | TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE
quantile	DOUBLE | DOUBLE[]
sample_size	INTEGER

Value

DECIMAL | DECIMAL[] | TINYINT | TINYINT[] | SMALLINT | SMALLINT[] | INTEGER | INTEGER[] | BIGINT | BIGINT[] | HUGEINT | HUGEINT[] | FLOAT | FLOAT[] | DOUBLE | DOUBLE[]

Overloads

- `reservoir_quantile(x = DECIMAL, quantile = DOUBLE)`
- `reservoir_quantile(x = DECIMAL, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = DECIMAL, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = DECIMAL, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = TINYINT, quantile = DOUBLE)`
- `reservoir_quantile(x = TINYINT, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = TINYINT, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = TINYINT, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = SMALLINT, quantile = DOUBLE)`
- `reservoir_quantile(x = SMALLINT, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = SMALLINT, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = SMALLINT, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = INTEGER, quantile = DOUBLE)`
- `reservoir_quantile(x = INTEGER, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = INTEGER, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = INTEGER, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = BIGINT, quantile = DOUBLE)`
- `reservoir_quantile(x = BIGINT, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = BIGINT, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = BIGINT, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = HUGEINT, quantile = DOUBLE)`
- `reservoir_quantile(x = HUGEINT, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = HUGEINT, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = HUGEINT, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = FLOAT, quantile = DOUBLE)`
- `reservoir_quantile(x = FLOAT, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = FLOAT, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = FLOAT, quantile = `DOUBLE[]`, sample_size = INTEGER)`
- `reservoir_quantile(x = DOUBLE, quantile = DOUBLE)`
- `reservoir_quantile(x = DOUBLE, quantile = DOUBLE, sample_size = INTEGER)`
- `reservoir_quantile(x = DOUBLE, quantile = `DOUBLE[]`)`
- `reservoir_quantile(x = DOUBLE, quantile = `DOUBLE[]`, sample_size = INTEGER)`

SQL examples

```
reservoir_quantile(A, 0.5, 1024)
```

`reverse` *DuckDB function reverse*

Description

Reverses the `string`.

Usage

```
reverse(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
reverse('hello')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`right` *DuckDB function right*

Description

Extract the right-most `count` characters.

Usage

```
right(string, count)
```

Arguments

`string`	`VARCHAR`
`count`	`BIGINT`

Value

`VARCHAR`

SQL examples

```
right('Hello ', 3)
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`right_grapheme` *DuckDB function right_grapheme*

Description

Extracts the right-most `count` grapheme clusters.

Usage

```
right_grapheme(string, count)
```

Arguments

`string`	`VARCHAR`
`count`	`BIGINT`

Value

`VARCHAR`

SQL examples

```
right_grapheme('    ', 1)
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

round *DuckDB function round*

Description

Rounds x to s decimal places.

Usage

```
round(x, precision)
```

Arguments

x	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL
precision	INTEGER

Value

TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL

Overloads

- round(x = TINYINT)
- round(x = TINYINT, precision = INTEGER)
- round(x = SMALLINT)
- round(x = SMALLINT, precision = INTEGER)
- round(x = INTEGER)
- round(x = INTEGER, precision = INTEGER)
- round(x = BIGINT)
- round(x = BIGINT, precision = INTEGER)
- round(x = HUGEINT)

- round(x = HUGEINT, precision = INTEGER)
- round(x = FLOAT)
- round(x = FLOAT, precision = INTEGER)
- round(x = DOUBLE)
- round(x = DOUBLE, precision = INTEGER)
- round(x = DECIMAL)
- round(x = DECIMAL, precision = INTEGER)

SQL examples

```
round(42.4332, 2)
```

---

round_even *DuckDB function round_even*

Description

DuckDB macro round_even().

Usage

```
round_even(x, n)
```

Arguments

x, n	Unspecified.

Value

Unspecified.

---

roundbankers *DuckDB function roundbankers*

Description

DuckDB macro roundbankers().

Usage

```
roundbankers(x, n)
```

Arguments

x, n	Unspecified.

Value

Unspecified.

`row` *DuckDB function row*

Description

Create an unnamed STRUCT (tuple) containing the argument values.

Usage

```
row()
```

Value

`STRUCT`

SQL examples

```
row(i, i % 4, i / 4)
```

`row_number` *DuckDB function row_number*

Description

DuckDB function `row_number()`.

Usage

```
row_number()
```

Value

`BIGINT`

row_to_json — *DuckDB function row_to_json*

Description

DuckDB function `row_to_json()`.

Usage

```
row_to_json()
```

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

rpad — *DuckDB function rpad*

Description

Pads the `string` with the `character` on the right until it has `count` characters. Truncates the `string` on the right if it has more than `count` characters.

Usage

```
rpad(string, count, character)
```

Arguments

`string`	`VARCHAR`
`count`	`INTEGER`
`character`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
rpad('hello', 10, '<')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

rtree_index_dump *DuckDB function rtree_index_dump*

Description

DuckDB function rtree_index_dump().

Usage

```
rtree_index_dump(col0)
```

Arguments

col0	VARCHAR

Value

Unspecified.

Provided by

The spatial extension (LOAD spatial;).

---

rtree_index_scan *DuckDB function rtree_index_scan*

Description

DuckDB function rtree_index_scan().

Usage

```
rtree_index_scan()
```

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

`rtrim` *DuckDB function rtrim*

Description

Removes any occurrences of any of the `characters` from the right side of the `string`. `characters` defaults to `space`.

Usage

```
rtrim(string, characters)
```

Arguments

`string, characters`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `rtrim(string = VARCHAR)`
- `rtrim(string = VARCHAR, characters = VARCHAR)`

SQL examples

```
rtrim('    test  ')
rtrim('>>>>test<<', '><')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`second` *DuckDB function second*

Description

Extract the second component from a date or timestamp.

Usage

```
second(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIME | TIMESTAMP | TIME WITH TIME ZONE | TIME_NS | TIMESTAMP W`

Value

`BIGINT`

SQL examples

```
second(timestamp '2021-08-03 11:59:44.123456')
```

`sem` *DuckDB function sem*

Description

Returns the standard error of the mean.

Usage

```
sem(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

seq_scan *DuckDB function seq_scan*

Description

DuckDB function `seq_scan()`.

Usage

```
seq_scan()
```

Value

Unspecified.

session_user *DuckDB function session_user*

Description

DuckDB macro `session_user()`.

Usage

```
session_user()
```

Value

Unspecified.

set_bit *DuckDB function set_bit*

Description

Sets the nth bit in bitstring to newvalue; the first (leftmost) bit is indexed 0. Returns a new bitstring.

Usage

```
set_bit(bitstring, index, new_value)
```

Arguments

bitstring	BIT
index	INTEGER
new_value	INTEGER

Value

BIT

SQL examples

```
set_bit('0110010'::BIT, 2, 0)
```

---

set_iceberg_schema_properties

DuckDB function set_iceberg_schema_properties

Description

DuckDB function `set_iceberg_schema_properties()`.

Usage

```
set_iceberg_schema_properties(col0, col1)
```

Arguments

col0	VARCHAR
col1	ANY

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`set_iceberg_table_properties` *DuckDB function set_iceberg_table_properties*

Description

DuckDB function `set_iceberg_table_properties()`.

Usage

```
set_iceberg_table_properties(col0, col1)
```

Arguments

col0	VARCHAR
col1	ANY

Value

Unspecified.

Provided by

The `iceberg` extension (`LOAD iceberg;`).

`setseed` *DuckDB function setseed*

Description

Sets the seed to be used for the random function.

Usage

```
setseed(col0)
```

Arguments

col0	DOUBLE

Value

`"NULL"`

SQL examples

```
setseed(0.42)
```

`sha1` *DuckDB function sha1*

Description

- `sha1(value = VARCHAR)`: Returns a `VARCHAR` with the SHA-1 hash of the `value`.
- `sha1(blob = BLOB)`: Returns a `VARCHAR` with the SHA-1 hash of the `blob`.

Usage

```
sha1(value)
```

Arguments

`value`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `sha1(value = VARCHAR)`
- `sha1(blob = BLOB)`

SQL examples

```
sha1(' ')
sha1('\xAA\xBB'::BLOB)
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha256()`, `to_base64()`, `unbin()`, `unhex()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`sha256` *DuckDB function sha256*

Description

- `sha256(value = VARCHAR)`: Returns a `VARCHAR` with the SHA-256 hash of the `value`.
- `sha256(blob = BLOB)`: Returns a `VARCHAR` with the SHA-256 hash of the `blob`.

Usage

```
sha256(value)
```

Arguments

`value`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `sha256(value = VARCHAR)`
- `sha256(blob = BLOB)`

SQL examples

```
sha256(' ')
sha256('\xAA\xBB'::BLOB)
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `to_base64()`, `unbin()`, `unhex()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

shapefile_meta *DuckDB function shapefile_meta*

Description

DuckDB function `shapefile_meta()`.

Usage

```
shapefile_meta(col0)
```

Arguments

`col0` `VARCHAR | VARCHAR[]`

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

shobj_description *DuckDB function shobj_description*

Description

DuckDB macro `shobj_description()`.

Usage

```
shobj_description(object_oid, catalog_name)
```

Arguments

`object_oid`, `catalog_name`
: Unspecified.

Value

Unspecified.

`show` *DuckDB function show*

Description

DuckDB function `show()`.

Usage

```
show(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`show_databases` *DuckDB function show_databases*

Description

DuckDB function `show_databases()`.

Usage

```
show_databases()
```

Value

Unspecified.

`show_tables` *DuckDB function show_tables*

Description

DuckDB function `show_tables()`.

Usage

```
show_tables()
```

Value

Unspecified.

`show_tables_expanded` *DuckDB function show_tables_expanded*

Description

DuckDB function `show_tables_expanded()`.

Usage

```
show_tables_expanded()
```

Value

Unspecified.

`sign` *DuckDB function sign*

Description

Returns the sign of x as -1, 0 or 1.

Usage

```
sign(x)
```

Arguments

x	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT

Value

`TINYINT`

SQL examples

```
sign(-349)
```

`signbit` *DuckDB function signbit*

Description

Returns whether the signbit is set or not.

Usage

```
signbit(x)
```

Arguments

x	FLOAT | DOUBLE

Value

BOOLEAN

SQL examples

```
signbit(-0.0)
```

`sin` *DuckDB function sin*

Description

Computes the sin of x.

Usage

```
sin(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
sin(90)
```

`sinh` *DuckDB function sinh*

Description

Computes the hyperbolic sin of x.

Usage

```
sinh(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
sinh(1)
```

`skewness` *DuckDB function skewness*

Description

Returns the skewness of all input values.

Usage

```
skewness(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
skewness(A)
```

sleep_ms *DuckDB function sleep_ms*

Description

Sleeps for the specified number of milliseconds and returns NULL.

Usage

```
sleep_ms(milliseconds)
```

Arguments

```
milliseconds   BIGINT
```

Value

```
"NULL"
```

SQL examples

```
sleep_ms(100)
```

sniff_csv *DuckDB function sniff_csv*

Description

DuckDB function `sniff_csv()`.

Usage

```
sniff_csv(
  col0,
  force_match,
  hive_types,
  union_by_name,
  hive_partitioning,
  filename,
  thousands,
  strict_mode,
  hive_types_autocast,
  encoding,
  dtypes,
  column_types,
  auto_detect,
```

```
    sample_size,
    files_to_sniff,
    dateformat,
    columns,
    all_varchar,
    new_line,
    header,
    types,
    skip,
    compression,
    comment,
    quote,
    max_line_size,
    store_rejects,
    delim,
    sep,
    column_names,
    buffer_size,
    timestampformat,
    normalize_names,
    ignore_errors,
    names,
    allow_quoted_nulls,
    maximum_line_size,
    rejects_table,
    force_not_null,
    parallel,
    auto_type_candidates,
    rejects_scan,
    rejects_limit,
    nullstr,
    escape,
    decimal_separator,
    null_padding
)
```

Arguments

col0	VARCHAR
force_match	BOOLEAN
hive_types	ANY
union_by_name	BOOLEAN
hive_partitioning	BOOLEAN
filename	ANY
thousands	VARCHAR
strict_mode	BOOLEAN

```
hive_types_autocast
               BOOLEAN
encoding       VARCHAR
dtypes         ANY
column_types   ANY
auto_detect    BOOLEAN
sample_size    BIGINT
files_to_sniff
               BIGINT
dateformat     VARCHAR
columns        ANY
all_varchar    BOOLEAN
new_line       VARCHAR
header         BOOLEAN
types          ANY
skip           BIGINT
compression    VARCHAR
comment        VARCHAR
quote          VARCHAR
max_line_size  VARCHAR
store_rejects  BOOLEAN
delim          VARCHAR
sep            VARCHAR
column_names   VARCHAR[]
buffer_size    UBIGINT
timestampformat
               VARCHAR
normalize_names
               BOOLEAN
ignore_errors  BOOLEAN
names          VARCHAR[]
allow_quoted_nulls
               BOOLEAN
maximum_line_size
               VARCHAR
rejects_table  VARCHAR
force_not_null
               VARCHAR[]
parallel       BOOLEAN
auto_type_candidates
               ANY
```

```
rejects_scan   VARCHAR
rejects_limit  BIGINT
nullstr        ANY
escape         VARCHAR
decimal_separator
               VARCHAR
null_padding   BOOLEAN
```

Value

Unspecified.

`split_part` *DuckDB function split_part*

Description

DuckDB macro `split_part()`.

Usage

```
split_part(string, delimiter, position)
```

Arguments

`string`, `delimiter`, `position`
: Unspecified.

Value

Unspecified.

`sql_auto_complete` *DuckDB function sql_auto_complete*

Description

DuckDB function `sql_auto_complete()`.

Usage

```
sql_auto_complete(
  col0,
  max_exact_suggestion_count,
  max_file_suggestion_count,
  max_suggestion_count
)
```

Arguments

col0	VARCHAR
max_exact_suggestion_count	UBIGINT
max_file_suggestion_count	UBIGINT
max_suggestion_count	UBIGINT

Value

Unspecified.

Provided by

The `autocomplete` extension (`LOAD autocomplete;`).

`sqlite_attach` *DuckDB function sqlite_attach*

Description

DuckDB function `sqlite_attach()`.

Usage

```
sqlite_attach(col0, overwrite)
```

Arguments

col0	VARCHAR
overwrite	BOOLEAN

Value

Unspecified.

Provided by

The `sqlite_scanner` extension (`LOAD sqlite_scanner;`).

`sqlite_query` *DuckDB function sqlite_query*

Description

DuckDB function `sqlite_query()`.

Usage

```
sqlite_query(col0, col1, params)
```

Arguments

`col0`	`VARCHAR`
`col1`	`VARCHAR`
`params`	`ANY`

Value

Unspecified.

Provided by

The `sqlite_scanner` extension (`LOAD sqlite_scanner;`).

`sqlite_scan` *DuckDB function sqlite_scan*

Description

DuckDB function `sqlite_scan()`.

Usage

```
sqlite_scan(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

Unspecified.

Provided by

The `sqlite_scanner` extension (`LOAD sqlite_scanner;`).

`sqrt` *DuckDB function sqrt*

Description

Returns the square root of x.

Usage

```
sqrt(x)
```

Arguments

x	DOUBLE

Value

`DOUBLE`

SQL examples

```
sqrt(4)
```

`ST_Affine` *DuckDB function ST_Affine*

Description

Applies an affine transformation to a geometry.

For the 2D variant, the transformation matrix is defined as follows:

```
| a b xoff |
| d e yoff |
| 0 0 1    |
```

For the 3D variant, the transformation matrix is defined as follows:

```
| a b c xoff |
| d e f yoff |
| g h i zoff |
| 0 0 0 1    |
```

The transformation is applied to all vertices of the geometry.

Usage

```
ST_Affine(geom, a, b, c, d, e, f, g, h, i, xoff, yoff, zoff)
```

Arguments

geom	GEOMETRY
a	DOUBLE
b	DOUBLE
c	DOUBLE
d	DOUBLE
e	DOUBLE
f	DOUBLE
g	DOUBLE
h	DOUBLE
i	DOUBLE
xoff	DOUBLE
yoff	DOUBLE
zoff	DOUBLE

Value

GEOMETRY

Overloads

- ST_Affine(geom = GEOMETRY, a = DOUBLE, b = DOUBLE, c = DOUBLE, d = DOUBLE, e = DOUBLE, f = DOUBLE, g = DOUBLE, h = DOUBLE, i = DOUBLE, xoff = DOUBLE, yoff = DOUBLE, zoff = DOUBLE)
- ST_Affine(geom = GEOMETRY, a = DOUBLE, b = DOUBLE, d = DOUBLE, e = DOUBLE, xoff = DOUBLE, yoff = DOUBLE)

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
-- Translate a point by (2, 3)
SELECT ST_Affine(ST_Point(1, 1),
                 1, 0,   -- a, b
                 0, 1,   -- d, e
                 2, 3);  -- xoff, yoff
-- POINT (3 4)
--
-- -- Scale a geometry by factor 2 in X and Y
```

```
-- SELECT ST_Affine(ST_Point(1, 1),
--                  2, 0, 0,   -- a, b, c
--                  0, 2, 0,   -- d, e, f
--                  0, 0, 1,   -- g, h, i
--                  0, 0, 0);  -- xoff, yoff, zoff
-- POINT (2 2)
```

`ST_Area` *DuckDB function ST_Area*

Description

Compute the area of a geometry.

Returns `0.0` for any geometry that is not a `POLYGON`, `MULTIPOLYGON` or `GEOMETRYCOLLECTION` containing polygon geometries.

The area is in the same units as the spatial reference system of the geometry.

The `POINT_2D` and `LINESTRING_2D` overloads of this function always return `0.0` but are included for completeness.

Usage

```
ST_Area(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Area(geom = GEOMETRY)`
- `ST_Area(polygon = POLYGON_2D)`
- `ST_Area(linestring = LINESTRING_2D)`
- `ST_Area(point = POINT_2D)`
- `ST_Area(box = BOX_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Area('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::GEOMETRY);
-- 1.0
```

`ST_Area_Spheroid` *DuckDB function ST_Area_Spheroid*

Description

Returns the area of a geometry in meters, using an ellipsoidal model of the earth

The input geometry is assumed to be in the EPSG:4326 coordinate system (WGS84), with [latitude, longitude] axis order and the area is returned in square meters. This function uses the GeographicLib library, calculating the area using an ellipsoidal model of the earth. This is a highly accurate method for calculating the area of a polygon taking the curvature of the earth into account, but is also the slowest.

Returns `0.0` for any geometry that is not a `POLYGON`, `MULTIPOLYGON` or `GEOMETRYCOLLECTION` containing polygon geometries.

Usage

```
ST_Area_Spheroid(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Area_Spheroid(geom = GEOMETRY)`
- `ST_Area_Spheroid(poly = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_AsGeoJSON` *DuckDB function ST_AsGeoJSON*

Description

Returns the geometry as a GeoJSON fragment

This does not return a complete GeoJSON document, only the geometry fragment. To construct a complete GeoJSON document or feature, look into using the DuckDB JSON extension in conjunction with this function. This function supports geometries with Z values, but not M values. M values are ignored.

Usage

```
ST_AsGeoJSON(geom)
```

Arguments

geom	GEOMETRY

Value

JSON

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_AsGeoJSON('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::GEOMETRY);
-- {"type":"Polygon","coordinates":[[[0.0, 0.0], [0.0, 1.0], [1.0, 1.0], [1.0, 0.0], [0.0, 0.0
--
-- -- Convert a geometry into a full GeoJSON feature (requires the JSON extension to be loaded)
-- SELECT CAST({
--     type: 'Feature',
--     geometry: ST_AsGeoJSON(ST_Point(1, 2)),
--     properties: {
--         name: 'my_point'
--     }
-- } AS JSON);
-- {"type":"Feature","geometry":{"type":"Point","coordinates":[1.0, 2.0]},"properties":{"name"
```

ST_AsHEXWKB *DuckDB function ST_AsHEXWKB*

Description

Returns the geometry as a HEXWKB string.

Usage

```
ST_AsHEXWKB(geom)
```

Arguments

geom	GEOMETRY

Value

VARCHAR

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_AsHexWKB('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::GEOMETRY);
-- 0103000000010000000500000000000000000000000000000000...
```

`ST_AsMVT` *DuckDB function ST_AsMVT*

Description

Make a Mapbox Vector Tile from a set of geometries and properties The function takes as input a row type (STRUCT) containing a geometry column and any number of property columns. It returns a single binary BLOB containing the Mapbox Vector Tile.

The function has the following signature:
`ST_AsMVT(row STRUCT, layer_name VARCHAR DEFAULT 'layer', extent INTEGER DEFAULT 4096, geom_colu`

- The first argument is a struct containing the geometry and properties.
- The second argument is the name of the layer in the vector tile. This argument is optional and defaults to 'layer'.
- The third argument is the extent of the tile. This argument is optional and defaults to 4096.
- The fourth argument is the name of the geometry column in the input row. This argument is optional. If not provided, the first geometry column in the input row will be used. If multiple geometry columns are present, an error will be raised.
- The fifth argument is the name of the feature id column in the input row. This argument is optional. If provided, the values in this column will be used as feature ids in the vector tile. The column must be of type INTEGER or BIGINT. If set to negative or NULL, a feature id will not be assigned to the corresponding feature.

The input struct must contain exactly one geometry column of type GEOMETRY. It can contain any number of property columns of types VARCHAR, FLOAT, DOUBLE, INTEGER, BIGINT, or BOOLEAN.

Example:

```
SELECT ST_AsMVT({'geom': geom, 'id': id, 'name': name}, 'cities', 4096, 'geom', 'id') AS tile
FROM cities;
```

This example creates a vector tile named 'cities' with an extent of 4096 from the 'cities' table, using 'geom' as the geometry column and 'id' as the feature id column.

However, you probably want to use the ST_AsMVTGeom function to first transform and clip your geometries to the tile extent. The following example assumes the geometry is in WebMercator ("EPSG:3857") coordinates. Replace `{z}`, `{x}`, and `{y}` with the appropriate tile coordinates, `{your table}` with your table name, and `{tile_path}` with the path to write the tile to.

```
COPY (
    SELECT ST_AsMVT({{
        "geometry": ST_AsMVTGeom(
            geometry,
            ST_Extent(ST_TileEnvelope({z}, {x}, {y})),
            4096,
            256,
            false
        )
    }})
    FROM {your table} WHERE ST_Intersects(geometry, ST_TileEnvelope({z}, {x}, {y}))
) to {tile_path} (FORMAT 'BLOB');
```

Usage

```
ST_AsMVT(col0, col1, col2, col3, col4)
```

Arguments

col0	ANY
col1	VARCHAR
col2	INTEGER
col3	VARCHAR
col4	VARCHAR

Value

```
BLOB
```

Overloads

- `ST_AsMVT(col0 = ANY)`
- `ST_AsMVT(col0 = ANY, col1 = VARCHAR)`
- `ST_AsMVT(col0 = ANY, col1 = VARCHAR, col2 = INTEGER)`
- `ST_AsMVT(col0 = ANY, col1 = VARCHAR, col2 = INTEGER, col3 = VARCHAR)`
- `ST_AsMVT(col0 = ANY, col1 = VARCHAR, col2 = INTEGER, col3 = VARCHAR, col4 = VARCHAR)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_AsMVTGeom` *DuckDB function ST_AsMVTGeom*

Description

Transform and clip geometry to a tile boundary

See "ST_AsMVT" for more details.

Usage

```
ST_AsMVTGeom(geom, bounds, extent, buffer, clip_geom)
```

Arguments

geom	GEOMETRY
bounds	BOX_2D
extent	BIGINT
buffer	BIGINT
clip_geom	BOOLEAN

Value

GEOMETRY

Overloads

- `ST_AsMVTGeom(geom = GEOMETRY, bounds = BOX_2D, extent = BIGINT, buffer = BIGINT, clip_geom = BOOLEAN)`
- `ST_AsMVTGeom(geom = GEOMETRY, bounds = BOX_2D, extent = BIGINT, buffer = BIGINT)`
- `ST_AsMVTGeom(geom = GEOMETRY, bounds = BOX_2D, extent = BIGINT)`
- `ST_AsMVTGeom(geom = GEOMETRY, bounds = BOX_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_AsSVG` *DuckDB function ST_AsSVG*

Description

Convert the geometry into a SVG fragment or path

The SVG fragment is returned as a string. The fragment is a path element that can be used in an SVG document. The second boolean argument specifies whether the path should be relative or absolute. The third argument specifies the maximum number of digits to use for the coordinates.

Points are formatted as cx/cy using absolute coordinates or x/y using relative coordinates.

Usage

```
ST_AsSVG(geom, relative, precision)
```

Arguments

`geom`	`GEOMETRY`
`relative`	`BOOLEAN`
`precision`	`INTEGER`

Value

`VARCHAR`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_AsSVG('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::GEOMETRY, false, 15);
-- M 0 0 L 0 -1 1 -1 1 0 Z
```

`st_astext` *DuckDB function st_astext*

Description

- `st_astext(geom = GEOMETRY)`: Returns the Well-Known Text (WKT) representation of the geometry.
- `st_astext(point = POINT_2D)`, `st_astext(linestring = LINESTRING_2D)`, `st_astext(polygon = POLYGON_2D)`, `st_astext(box = BOX_2D)`: Returns the geometry as a WKT string.

Usage

```
st_astext(geom)
```

Arguments

geom	GEOMETRY

Value

VARCHAR

Overloads

- st_astext(geom = GEOMETRY)
- st_astext(point = POINT_2D)
- st_astext(linestring = LINESTRING_2D)
- st_astext(polygon = POLYGON_2D)
- st_astext(box = BOX_2D)

Provided by

- st_astext(geom = GEOMETRY): built in
- st_astext(point = POINT_2D): spatial
- st_astext(linestring = LINESTRING_2D): spatial
- st_astext(polygon = POLYGON_2D): spatial
- st_astext(box = BOX_2D): spatial

SQL examples

```
ST_AsText(ST_GeomFromWKB(X'0101000000000000000000000000000000000000000000000000000000'))
SELECT ST_MakeEnvelope(0, 0, 1, 1);
-- POLYGON ((0 0, 0 1, 1 1, 1 0, 0 0))
```

See Also

Other geometry: st_aswkb(), st_crs(), st_geomfromwkb()

`st_aswkb` *DuckDB function st_aswkb*

Description

Returns the Well-Known Binary (WKB) representation of the geometry.

Usage

```
st_aswkb(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`BLOB`

SQL examples

```
st_aswkb(ST_GeomFromWKB(X'0101000000000000000000000000000000000000000000000000000000'))
```

See Also

Other geometry: `st_astext()`, `st_crs()`, `st_geomfromwkb()`

`ST_Azimuth` *DuckDB function ST_Azimuth*

Description

Returns the azimuth (a clockwise angle measured from north) of two points in radian.

Usage

```
ST_Azimuth(origin, target)
```

Arguments

`origin`, `target`	`GEOMETRY | POINT_2D`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT degrees(ST_Azimuth(ST_Point(0, 0), ST_Point(0, 1)));
-- 90.0
```

`ST_Boundary` *DuckDB function ST_Boundary*

Description

Returns the ”boundary” of a geometry.

Usage

```
ST_Boundary(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Buffer` *DuckDB function ST_Buffer*

Description

Returns a buffer around the input geometry at the target distance

`geom` is the input geometry.

`distance` is the target distance for the buffer, using the same units as the input geometry.

`num_triangles` represents how many triangles that will be produced to approximate a quarter circle. The larger the number, the smoother the resulting geometry. The default value is 8.

`cap_style` must be one of ”CAP_ROUND”, ”CAP_FLAT”, ”CAP_SQUARE”. This parameter is case-insensitive.

`join_style` must be one of "JOIN_ROUND", "JOIN_MITRE", "JOIN_BEVEL". This parameter is case-insensitive.

`mitre_limit` only applies when `join_style` is "JOIN_MITRE". It is the ratio of the distance from the corner to the mitre point to the corner radius. The default value is 1.0.

This is a planar operation and will not take into account the curvature of the earth.

Usage

```
ST_Buffer(geom, distance, num_triangles, cap_style, join_style, mitre_limit)
```

Arguments

`geom`	`GEOMETRY`
`distance`	`DOUBLE`
`num_triangles`	`INTEGER`
`cap_style`	`VARCHAR`
`join_style`	`VARCHAR`
`mitre_limit`	`DOUBLE`

Value

`GEOMETRY`

Overloads

- `ST_Buffer(geom = GEOMETRY, distance = DOUBLE)`
- `ST_Buffer(geom = GEOMETRY, distance = DOUBLE, num_triangles = INTEGER)`
- `ST_Buffer(geom = GEOMETRY, distance = DOUBLE, num_triangles = INTEGER, cap_style = VARCHAR, join_style = VARCHAR, mitre_limit = DOUBLE)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_BuildArea` *DuckDB function ST_BuildArea*

Description

Creates a polygonal geometry by attempting to "fill in" the input geometry.

Unlike ST_Polygonize, this function does not fill in holes.

Usage

```
ST_BuildArea(geom)
```

Arguments

geom	GEOMETRY

Value

GEOMETRY

Provided by

The spatial extension (LOAD spatial;).

ST_Centroid — *DuckDB function ST_Centroid*

Description

Returns the centroid of a geometry.

Usage

```
ST_Centroid(box)
```

Arguments

box	BOX_2D | BOX_2DF

Value

GEOMETRY | POINT_2D

Overloads

- ST_Centroid(geom = GEOMETRY)
- ST_Centroid(point = POINT_2D)
- ST_Centroid(linestring = LINESTRING_2D)
- ST_Centroid(polygon = POLYGON_2D)
- ST_Centroid(box = BOX_2D)
- ST_Centroid(box = BOX_2DF)

Provided by

The spatial extension (LOAD spatial;).

`ST_ClosestPoint` *DuckDB function ST_ClosestPoint*

Description

Returns the closest point on the first geometry to the second geometry.

Usage

```
ST_ClosestPoint(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Collect` *DuckDB function ST_Collect*

Description

Collects a list of geometries into a collection geometry.

- If all geometries are `POINT`'s, a `MULTIPOINT` is returned.
- If all geometries are `LINESTRING`'s, a `MULTILINESTRING` is returned.
- If all geometries are `POLYGON`'s, a `MULTIPOLYGON` is returned.
- Otherwise if the input collection contains a mix of geometry types, a `GEOMETRYCOLLECTION` is returned.

Empty and `NULL` geometries are ignored. If all geometries are empty or `NULL`, a `GEOMETRYCOLLECTION EMPTY` is returned.

Usage

```
ST_Collect(geoms)
```

Arguments

`geoms`	`GEOMETRY[]`

Value

GEOMETRY

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
-- With all POINT's, a MULTIPOINT is returned
SELECT ST_Collect([ST_Point(1, 2), ST_Point(3, 4)]);
-- MULTIPOINT (1 2, 3 4)
--
-- -- With mixed geometry types, a GEOMETRYCOLLECTION is returned
-- SELECT ST_Collect([ST_Point(1, 2), ST_GeomFromText('LINESTRING(3 4, 5 6)')]);
-- GEOMETRYCOLLECTION (POINT (1 2), LINESTRING (3 4, 5 6))
--
-- -- Note that the empty geometry is ignored, so the result is a MULTIPOINT
-- SELECT ST_Collect([ST_Point(1, 2), NULL, ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')]);
-- MULTIPOINT (1 2)
--
-- -- If all geometries are empty or NULL, a GEOMETRYCOLLECTION EMPTY is returned
-- SELECT ST_Collect([NULL, ST_GeomFromText('GEOMETRYCOLLECTION EMPTY')]);
-- GEOMETRYCOLLECTION EMPTY
--
-- -- Tip: You can use the `ST_Collect` function together with the `list()` aggregate function
--
-- CREATE TABLE points (geom GEOMETRY);
--
-- INSERT INTO points VALUES (ST_Point(1, 2)), (ST_Point(3, 4));
--
-- SELECT ST_Collect(list(geom)) FROM points;
-- MULTIPOINT (1 2, 3 4)
```

ST_CollectionExtract *DuckDB function ST_CollectionExtract*

Description

Extracts geometries from a GeometryCollection into a typed multi geometry.

If the input geometry is a GeometryCollection, the function will return a multi geometry, determined by the type parameter.

- if type = 1, returns a MultiPoint containing all the Points in the collection
- if type = 2, returns a MultiLineString containing all the LineStrings in the collection
- if type = 3, returns a MultiPolygon containing all the Polygons in the collection

If no `type` parameters is provided, the function will return a multi geometry matching the highest "surface dimension" of the contained geometries. E.g. if the collection contains only Points, a MultiPoint will be returned. But if the collection contains both Points and LineStrings, a MultiLineString will be returned. Similarly, if the collection contains Polygons, a MultiPolygon will be returned. Contained geometries of a lower surface dimension will be ignored.

If the input geometry contains nested GeometryCollections, their geometries will be extracted recursively and included into the final multi geometry as well.

If the input geometry is not a GeometryCollection, the function will return the input geometry as is.

Usage

```
ST_CollectionExtract(geom, type)
```

Arguments

`geom`	`GEOMETRY`
`type`	`INTEGER`

Value

`GEOMETRY`

Overloads

- `ST_CollectionExtract(geom = GEOMETRY, type = INTEGER)`
- `ST_CollectionExtract(geom = GEOMETRY)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_CollectionExtract('MULTIPOINT(1 2, 3 4)'::GEOMETRY, 1);
-- MULTIPOINT (1 2, 3 4)
```

`ST_ConcaveHull` — *DuckDB function ST_ConcaveHull*

Description

Returns the 'concave' hull of the input geometry, containing all of the source input's points, and which can be used to create polygons from points. The ratio parameter dictates the level of concavity; 1.0 returns the convex hull; and 0 indicates to return the most concave hull possible. Set allowHoles to a non-zero value to allow output containing holes.

Usage

```
ST_ConcaveHull(geom, ratio, allowHoles)
```

Arguments

geom	GEOMETRY
ratio	DOUBLE
allowHoles	BOOLEAN

Value

GEOMETRY

Provided by

The `spatial` extension (`LOAD spatial;`).

ST_Contains — *DuckDB function ST_Contains*

Description

Returns true if the first geometry contains the second geometry

In contrast to `ST_ContainsProperly`, this function will also return true if `geom2` is contained strictly on the boundary of `geom1`. A geometry always `ST_Contains` itself, but does not `ST_ContainsProperly` itself.

Usage

```
ST_Contains(geom1, geom2)
```

Arguments

geom1	POLYGON_2D | GEOMETRY
geom2	POINT_2D | GEOMETRY

Value

BOOLEAN

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_ContainsProperly` *DuckDB function ST_ContainsProperly*

Description

Returns true if the first geometry \"properly\" contains the second geometry

In contrast to `ST_Contains`, this function does not return true if `geom2` is contained strictly on the boundary of `geom1`. A geometry always `ST_Contains` itself, but does not `ST_ContainsProperly` itself.

Usage

```
ST_ContainsProperly(geom1, geom2)
```

Arguments

`geom1`, `geom2` `GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_ConvexHull` *DuckDB function ST_ConvexHull*

Description

Returns the convex hull enclosing the geometry.

Usage

```
ST_ConvexHull(geom)
```

Arguments

`geom` `GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoverageClean` *DuckDB function ST_CoverageClean*

Description

Aligns the edges of a list of polygons whose edges are meant to align but are in fact exact matches.

Returns a collection of fixed polygons with the same size and order as the input polygons. EMPTY will be used in place of collapsed polygons.

Usage

```
ST_CoverageClean(geoms, snapping_distance, gap_maximum_width)
```

Arguments

`geoms`	`GEOMETRY[]`
`snapping_distance`	`DOUBLE`
`gap_maximum_width`	`DOUBLE`

Value

`GEOMETRY`

Overloads

- `ST_CoverageClean(geoms = `GEOMETRY[]`, snapping_distance = DOUBLE, gap_maximum_width = DOUBLE)`
- `ST_CoverageClean(geoms = `GEOMETRY[]`, snapping_distance = DOUBLE)`
- `ST_CoverageClean(geoms = `GEOMETRY[]`)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoverageInvalidEdges`

DuckDB function ST_CoverageInvalidEdges

Description

Returns the invalid edges in a polygonal coverage, which are edges that are not shared by two polygons. Returns NULL if the input is not a polygonal coverage, or if the input is valid. Tolerance is 0 by default.

Usage

```
ST_CoverageInvalidEdges(geoms, tolerance)
```

Arguments

`geoms`	`GEOMETRY[]`
`tolerance`	`DOUBLE`

Value

`GEOMETRY`

Overloads

- `ST_CoverageInvalidEdges(geoms = \`GEOMETRY[]\`, tolerance = DOUBLE)`
- `ST_CoverageInvalidEdges(geoms = \`GEOMETRY[]\`)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoverageInvalidEdges_Agg`

DuckDB function ST_CoverageInvalidEdges_Agg

Description

Returns the invalid edges of a coverage geometry.

Usage

```
ST_CoverageInvalidEdges_Agg(col0, col1)
```

Arguments

col0	GEOMETRY
col1	DOUBLE

Value

GEOMETRY

Overloads

- ST_CoverageInvalidEdges_Agg(col0 = GEOMETRY)
- ST_CoverageInvalidEdges_Agg(col0 = GEOMETRY, col1 = DOUBLE)

Provided by

The spatial extension (LOAD spatial;).

ST_CoverageSimplify *DuckDB function ST_CoverageSimplify*

Description

Simplify the edges in a polygonal coverage, preserving the coverange by ensuring that the there are no seams between the resulting simplified polygons.

By default, the boundary of the coverage is also simplified, but this can be controlled with the optional third 'simplify_boundary' parameter.

Usage

```
ST_CoverageSimplify(geoms, tolerance, simplify_boundary)
```

Arguments

geoms	GEOMETRY[]
tolerance	DOUBLE
simplify_boundary	BOOLEAN

Value

GEOMETRY

Overloads

- ST_CoverageSimplify(geoms = `GEOMETRY[]`, tolerance = DOUBLE, simplify_boundary = BOOLEAN)
- ST_CoverageSimplify(geoms = `GEOMETRY[]`, tolerance = DOUBLE)

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoverageSimplify_Agg`

DuckDB function ST_CoverageSimplify_Agg

Description

Simplifies a set of geometries while maintaining coverage.

Usage

```
ST_CoverageSimplify_Agg(col0, col1, col2)
```

Arguments

`col0`	`GEOMETRY`
`col1`	`DOUBLE`
`col2`	`BOOLEAN`

Value

`GEOMETRY`

Overloads

- `ST_CoverageSimplify_Agg(col0 = GEOMETRY, col1 = DOUBLE)`
- `ST_CoverageSimplify_Agg(col0 = GEOMETRY, col1 = DOUBLE, col2 = BOOLEAN)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoverageUnion` *DuckDB function ST_CoverageUnion*

Description

Union all geometries in a polygonal coverage into a single geometry. This may be faster than using `ST_Union`, but may use more memory.

Usage

```
ST_CoverageUnion(geoms)
```

Arguments

`geoms`	`GEOMETRY[]`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoverageUnion_Agg` *DuckDB function ST_CoverageUnion_Agg*

Description

Unions a set of geometries while maintaining coverage.

Usage

```
ST_CoverageUnion_Agg(col0)
```

Arguments

`col0`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_CoveredBy` *DuckDB function ST_CoveredBy*

Description

Returns true if geom1 is ”covered by” geom2.

Usage

```
ST_CoveredBy(geom1, geom2)
```

Arguments

`geom1`, `geom2` `GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Covers` *DuckDB function ST_Covers*

Description

Returns true if the geom1 ”covers” geom2.

Usage

```
ST_Covers(geom1, geom2)
```

Arguments

`geom1`, `geom2` `GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Crosses` *DuckDB function ST_Crosses*

Description

Returns true if geom1 ”crosses” geom2.

Usage

```
ST_Crosses(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`st_crs` *DuckDB function st_crs*

Description

Returns the Coordinate Reference System (CRS) identifier of the geometry.

Usage

```
st_crs(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`VARCHAR`

See Also

Other geometry: `st_astext()`, `st_aswkb()`, `st_geomfromwkb()`

`ST_Difference` *DuckDB function ST_Difference*

Description

Returns the ”difference” between two geometries.

Usage

```
ST_Difference(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Dimension` *DuckDB function ST_Dimension*

Description

Returns the ”topological dimension” of a geometry.

- For POINT and MULTIPOINT geometries, returns `0`
- For LINESTRING and MULTILINESTRING, returns `1`
- For POLYGON and MULTIPOLYGON, returns `2`
- For GEOMETRYCOLLECTION, returns the maximum dimension of the contained geometries, or 0 if the collection is empty.

Usage

```
ST_Dimension(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`INTEGER`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Dimension('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'::GEOMETRY);
-- 2
```

ST_Disjoint *DuckDB function ST_Disjoint*

Description

Returns true if the geometries are disjoint.

Usage

```
ST_Disjoint(geom1, geom2)
```

Arguments

`geom1`, `geom2` `GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

ST_Distance *DuckDB function ST_Distance*

Description

Returns the planar distance between two geometries.

Usage

```
ST_Distance(point1, point2)
```

Arguments

`point1`, `point2` `POINT_2D`

Value

DOUBLE

Overloads

- ST_Distance(point1 = POINT_2D, point2 = POINT_2D)
- ST_Distance(point = POINT_2D, linestring = LINESTRING_2D)
- ST_Distance(linestring = LINESTRING_2D, point = POINT_2D)
- ST_Distance(geom1 = GEOMETRY, geom2 = GEOMETRY)

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
SELECT ST_Distance('POINT (0 0)'::GEOMETRY, 'POINT (3 4)'::GEOMETRY);
-- 5.0
--
-- -- Z coordinates are ignored
-- SELECT ST_Distance('POINT Z (0 0 0)'::GEOMETRY, 'POINT Z (3 4 5)'::GEOMETRY);
-- 5.0
```

ST_Distance_GEOS *DuckDB function ST_Distance_GEOS*

Description

Returns the planar distance between two geometries.

Usage

```
ST_Distance_GEOS(geom1, geom2)
```

Arguments

geom1, geom2 GEOMETRY

Value

DOUBLE

Provided by

The spatial extension (LOAD spatial;).

`ST_Distance_Sphere` *DuckDB function ST_Distance_Sphere*

Description

Returns the haversine (great circle) distance between two geometries.

- Only supports POINT geometries.
- Returns the distance in meters.
- The input is expected to be in WGS84 (EPSG:4326) coordinates, using a [latitude, longitude] axis order.

Usage

```
ST_Distance_Sphere(geom1, geom2)
```

Arguments

`geom1`, `geom2` `GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Distance_Sphere(geom1 = GEOMETRY, geom2 = GEOMETRY)`
- `ST_Distance_Sphere(point1 = POINT_2D, point2 = POINT_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Distance_Spheroid` *DuckDB function ST_Distance_Spheroid*

Description

Returns the distance between two geometries in meters using an ellipsoidal model of the earths surface

The input geometry is assumed to be in the EPSG:4326 coordinate system (WGS84), with [latitude, longitude] axis order and the distance limit is expected to be in meters. This function uses the GeographicLib library to solve the inverse geodesic problem, calculating the distance between two points using an ellipsoidal model of the earth. This is a highly accurate method for calculating the distance between two arbitrary points taking the curvature of the earths surface into account, but is also the slowest.

Usage

```
ST_Distance_Spheroid(p1, p2)
```

Arguments

p1, p2	POINT_2D

Value

```
DOUBLE
```

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Note: the coordinates are in WGS84 and [latitude, longitude] axis order
-- Whats the distance between New York and Amsterdam (JFK and AMS airport)?
SELECT st_distance_spheroid(
st_point(40.6446, -73.7797),
st_point(52.3130, 4.7725)
);
-- 5863418.7459356235
-- -- Roughly 5863km!
```

`ST_Drivers` *DuckDB function ST_Drivers*

Description

Returns the list of supported GDAL drivers and file formats

Note that far from all of these drivers have been tested properly. Some may require additional options to be passed to work as expected. If you run into any issues please first consult the consult the GDAL docs.

Usage

```
ST_Drivers()
```

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT * FROM ST_Drivers();
```

`ST_Dump` *DuckDB function ST_Dump*

Description

Dumps a geometry into a list of sub-geometries and their ”path” in the original geometry.

You can use the `unnest(res, recursive := true)` function to explode the resulting list of structs into multiple rows.

Usage

```
ST_Dump(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

```
STRUCT(geom GEOMETRY, path INTEGER[])[]
```

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Dump('MULTIPOINT(1 2, 3 4)'::GEOMETRY);
-- [{'geom': 'POINT(1 2)', 'path': [0]}, {'geom': 'POINT(3 4)', 'path': [1]}]
--
-- SELECT unnest(ST_Dump('MULTIPOINT(1 2, 3 4)'::GEOMETRY), recursive := true);
-- -- +-------------+---------+
-- -- |    geom     |  path   |
-- -- |  geometry   | int32[] |
-- -- +-------------+---------+
-- -- | POINT (1 2) | [1]     |
-- -- | POINT (3 4) | [2]     |
-- -- +-------------+---------+
```

`ST_DWithin` *DuckDB function ST_DWithin*

Description

Returns if two geometries are within a target distance of each-other.

Usage

```
ST_DWithin(geom1, geom2, distance)
```

Arguments

`geom1`	`GEOMETRY`
`geom2`	`GEOMETRY`
`distance`	`DOUBLE`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_DWithin_GEOS` *DuckDB function ST_DWithin_GEOS*

Description

Returns true if two geometries are within a target distance of each-other.

Usage

```
ST_DWithin_GEOS(geom1, geom2, distance)
```

Arguments

`geom1`	`GEOMETRY`
`geom2`	`GEOMETRY`
`distance`	`DOUBLE`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_DWithin_Spheroid` *DuckDB function ST_DWithin_Spheroid*

Description

Returns if two POINT_2D's are within a target distance in meters, using an ellipsoidal model of the earths surface

The input geometry is assumed to be in the EPSG:4326 coordinate system (WGS84), with [latitude, longitude] axis order and the distance is returned in meters. This function uses the GeographicLib library to solve the inverse geodesic problem, calculating the distance between two points using an ellipsoidal model of the earth. This is a highly accurate method for calculating the distance between two arbitrary points taking the curvature of the earths surface into account, but is also the slowest.

Usage

```
ST_DWithin_Spheroid(p1, p2, distance)
```

Arguments

`p1`	`POINT_2D`
`p2`	`POINT_2D`
`distance`	`DOUBLE`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_EndPoint` *DuckDB function ST_EndPoint*

Description

Returns the end point of a LINESTRING.

Usage

```
ST_EndPoint(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

```
GEOMETRY | POINT_2D
```

Overloads

- `ST_EndPoint(geom = GEOMETRY)`
- `ST_EndPoint(line = LINESTRING_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Envelope` *DuckDB function ST_Envelope*

Description

Returns the minimum bounding rectangle of a geometry as a polygon geometry.

Usage

```
ST_Envelope(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

```
GEOMETRY
```

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Envelope_Agg` *DuckDB function ST_Envelope_Agg*

Description

Alias for ST_Extent_Agg.

Computes the minimal-bounding-box polygon containing the set of input geometries.

Usage

```
ST_Envelope_Agg(col0)
```

Arguments

`col0`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Extent_Agg(geom) FROM UNNEST([ST_Point(1,1), ST_Point(5,5)]) AS _(geom);
-- POLYGON ((1 1, 1 5, 5 5, 5 1, 1 1))
```

`ST_Equals` *DuckDB function ST_Equals*

Description

Returns true if the geometries are ”equal”.

Usage

```
ST_Equals(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Expand` *DuckDB function ST_Expand*

Description

Expand the input geometry by the specified distance, returning a polygon.

`geom` is the input geometry.

`distance` is the target distance for the expansion, using the same units as the input geometry.

This is a planar operation and will not take into account the curvature of the earth.

Usage

```
ST_Expand(geom, distance)
```

Arguments

`geom`	`GEOMETRY`
`distance`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_AsText(ST_Expand(ST_GeomFromText('POINT(20 30)'), 0.1));
```

`ST_Extent` *DuckDB function ST_Extent*

Description

Returns the minimal bounding box enclosing the input geometry.

Usage

```
ST_Extent(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`BOX_2D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Extent_Agg` *DuckDB function ST_Extent_Agg*

Description

Computes the minimal-bounding-box polygon containing the set of input geometries.

Usage

```
ST_Extent_Agg(col0)
```

Arguments

`col0`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Extent_Agg(geom) FROM UNNEST([ST_Point(1,1), ST_Point(5,5)]) AS _(geom);
-- POLYGON ((1 1, 1 5, 5 5, 5 1, 1 1))
```

`ST_Extent_Approx` *DuckDB function ST_Extent_Approx*

Description

Returns the approximate bounding box of a geometry, if available.

This function is only really used internally, and returns the cached bounding box of the geometry if it exists. This function may be removed or renamed in the future.

Usage

```
ST_Extent_Approx(geom)
```

Arguments

geom	GEOMETRY

Value

```
BOX_2DF
```

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_ExteriorRing` *DuckDB function ST_ExteriorRing*

Description

Returns the exterior ring (shell) of a polygon geometry.

Usage

```
ST_ExteriorRing(geom)
```

Arguments

geom	GEOMETRY

Value

```
GEOMETRY | LINESTRING_2D
```

Overloads

- `ST_ExteriorRing(geom = GEOMETRY)`
- `ST_ExteriorRing(polygon = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_FlipCoordinates` *DuckDB function ST_FlipCoordinates*

Description

Returns a new geometry with the coordinates of the input geometry "flipped" so that x = y and y = x.

Usage

```
ST_FlipCoordinates(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY | POINT_2D | LINESTRING_2D | POLYGON_2D | BOX_2D`

Overloads

- `ST_FlipCoordinates(geom = GEOMETRY)`
- `ST_FlipCoordinates(point = POINT_2D)`
- `ST_FlipCoordinates(linestring = LINESTRING_2D)`
- `ST_FlipCoordinates(polygon = POLYGON_2D)`
- `ST_FlipCoordinates(box = BOX_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Force2D` *DuckDB function ST_Force2D*

Description

Forces the vertices of a geometry to have X and Y components

This function will drop any Z and M values from the input geometry, if present. If the input geometry is already 2D, it will be returned as is.

Usage

```
ST_Force2D(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Force3DM` *DuckDB function ST_Force3DM*

Description

Forces the vertices of a geometry to have X, Y and M components

The following cases apply:

- If the input geometry has a Z component but no M component, the Z component will be replaced with the new M value.
- If the input geometry has a M component but no Z component, it will be returned as is.
- If the input geometry has both a Z component and a M component, the Z component will be removed.
- Otherwise, if the input geometry has neither a Z or M component, the new M value will be added to the vertices of the input geometry.

Usage

```
ST_Force3DM(geom, m)
```

Arguments

geom	GEOMETRY
m	DOUBLE

Value

GEOMETRY

Provided by

The spatial extension (LOAD spatial;).

ST_Force3DZ *DuckDB function ST_Force3DZ*

Description

Forces the vertices of a geometry to have X, Y and Z components

The following cases apply:

- If the input geometry has a M component but no Z component, the M component will be replaced with the new Z value.
- If the input geometry has a Z component but no M component, it will be returned as is.
- If the input geometry has both a Z component and a M component, the M component will be removed.
- Otherwise, if the input geometry has neither a Z or M component, the new Z value will be added to the vertices of the input geometry.

Usage

```
ST_Force3DZ(geom, z)
```

Arguments

geom	GEOMETRY
z	DOUBLE

Value

GEOMETRY

Provided by

The spatial extension (LOAD spatial;).

`ST_Force4D` *DuckDB function ST_Force4D*

Description

Forces the vertices of a geometry to have X, Y, Z and M components

The following cases apply:

- If the input geometry has a Z component but no M component, the new M value will be added to the vertices of the input geometry.
- If the input geometry has a M component but no Z component, the new Z value will be added to the vertices of the input geometry.
- If the input geometry has both a Z component and a M component, the geometry will be returned as is.
- Otherwise, if the input geometry has neither a Z or M component, the new Z and M values will be added to the vertices of the input geometry.

Usage

```
ST_Force4D(geom, z, m)
```

Arguments

`geom`	`GEOMETRY`
`z`	`DOUBLE`
`m`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_GeneratePoints` *DuckDB function ST_GeneratePoints*

Description

Generates a set of random points within the specified bounding box.

Takes a bounding box (min_x, min_y, max_x, max_y), a count of points to generate, and optionally a seed for the random number generator.

Usage

```
ST_GeneratePoints(col0, col1, col2)
```

Arguments

`col0`	`BOX_2D`
`col1`	`BIGINT`
`col2`	`BIGINT`

Value

Unspecified.

Overloads

- `ST_GeneratePoints(col0 = BOX_2D, col1 = BIGINT)`
- `ST_GeneratePoints(col0 = BOX_2D, col1 = BIGINT, col2 = BIGINT)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT * FROM ST_GeneratePoints({min_x: 0, min_y:0, max_x:10, max_y:10}::BOX_2D, 5, 42);
```

`ST_GeometryType` *DuckDB function ST_GeometryType*

Description

Returns a 'GEOMETRY_TYPE' enum identifying the input geometry type. Possible enum return types are: `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, and `GEOMETRYCOLLECTION`.

Usage

```
ST_GeometryType(geom)
```

Arguments

geom	GEOMETRY

Value

```
ANY
```

Overloads

- `ST_GeometryType(geom = GEOMETRY)`
- `ST_GeometryType(point = POINT_2D)`
- `ST_GeometryType(linestring = LINESTRING_2D)`
- `ST_GeometryType(polygon = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT DISTINCT ST_GeometryType(ST_GeomFromText('POINT(1 1)'));
-- POINT
```

`ST_GeomFromGeoJSON` *DuckDB function ST_GeomFromGeoJSON*

Description

Deserializes a GEOMETRY from a GeoJSON fragment.

Usage

```
ST_GeomFromGeoJSON(geojson)
```

Arguments

`geojson`	`JSON | VARCHAR`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_GeomFromGeoJSON('{"type": "Point", "coordinates": [1.0, 2.0]}');
-- POINT (1 2)
```

`ST_GeomFromHEXEWKB` *DuckDB function ST_GeomFromHEXEWKB*

Description

Deserialize a GEOMETRY from a HEX(E)WKB encoded string

DuckDB spatial doesn't currently differentiate between `WKB` and `EWKB`, so `ST_GeomFromHEXWKB` and `ST_GeomFromHEXEWKB` are just aliases of each other.

Usage

```
ST_GeomFromHEXEWKB(hexwkb)
```

Arguments

`hexwkb`	`VARCHAR`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_GeomFromHEXWKB` *DuckDB function ST_GeomFromHEXWKB*

Description

Deserialize a GEOMETRY from a HEX(E)WKB encoded string

DuckDB spatial doesn't currently differentiate between `WKB` and `EWKB`, so `ST_GeomFromHEXWKB` and `ST_GeomFromHEXEWKB` are just aliases of each other.

Usage

```
ST_GeomFromHEXWKB(hexwkb)
```

Arguments

`hexwkb`	`VARCHAR`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_GeomFromText` *DuckDB function ST_GeomFromText*

Description

Deserialize a GEOMETRY from a WKT encoded string.

Usage

```
ST_GeomFromText(wkt, ignore_invalid)
```

Arguments

```
wkt             VARCHAR
ignore_invalid
                BOOLEAN
```

Value

```
GEOMETRY
```

Overloads

- `ST_GeomFromText(wkt = VARCHAR)`
- `ST_GeomFromText(wkt = VARCHAR, ignore_invalid = BOOLEAN)`

Provided by

The `spatial` extension (`LOAD spatial;`).

---

st_geomfromwkb *DuckDB function st_geomfromwkb*

Description

Creates a geometry from Well-Known Binary (WKB) representation.

Usage

```
st_geomfromwkb(wkb)
```

Arguments

```
wkb             BLOB
```

Value

```
GEOMETRY
```

SQL examples

```
ST_GeomFromWKB(X'0101000000000000000000000000000000000000000000000000000000')
```

See Also

Other geometry: `st_astext()`, `st_aswkb()`, `st_crs()`

`ST_HasM` *DuckDB function ST_HasM*

Description

Check if the input geometry has M values.

Usage

```
ST_HasM(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- HasM for a 2D geometry
SELECT ST_HasM(ST_GeomFromText('POINT(1 1)'));
-- false
--
-- -- HasM for a 3DZ geometry
-- SELECT ST_HasM(ST_GeomFromText('POINT Z(1 1 1)'));
-- false
--
-- -- HasM for a 3DM geometry
-- SELECT ST_HasM(ST_GeomFromText('POINT M(1 1 1)'));
-- true
--
-- -- HasM for a 4D geometry
-- SELECT ST_HasM(ST_GeomFromText('POINT ZM(1 1 1 1)'));
-- true
```

`ST_HasZ` *DuckDB function ST_HasZ*

Description

Check if the input geometry has Z values.

Usage

```
ST_HasZ(geom)
```

Arguments

geom	GEOMETRY

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- HasZ for a 2D geometry
SELECT ST_HasZ(ST_GeomFromText('POINT(1 1)'));
-- false
--
-- -- HasZ for a 3DZ geometry
-- SELECT ST_HasZ(ST_GeomFromText('POINT Z(1 1 1)'));
-- true
--
-- -- HasZ for a 3DM geometry
-- SELECT ST_HasZ(ST_GeomFromText('POINT M(1 1 1)'));
-- false
--
-- -- HasZ for a 4D geometry
-- SELECT ST_HasZ(ST_GeomFromText('POINT ZM(1 1 1 1)'));
-- true
```

`ST_Hilbert` *DuckDB function ST_Hilbert*

Description

Encodes the X and Y values as the hilbert curve index for a curve covering the given bounding box. If a geometry is provided, the center of the approximate bounding box is used as the point to encode. If no bounding box is provided, the hilbert curve index is mapped to the full range of a single-precision float. For the BOX_2D and BOX_2DF variants, the center of the box is used as the point to encode.

Usage

```
ST_Hilbert(box, bounds)
```

Arguments

`box`, `bounds` `BOX_2D | BOX_2DF`

Value

`UINTEGER`

Overloads

- `ST_Hilbert(x = DOUBLE, y = DOUBLE, bounds = BOX_2D)`
- `ST_Hilbert(geom = GEOMETRY, bounds = BOX_2D)`
- `ST_Hilbert(geom = GEOMETRY)`
- `ST_Hilbert(box = BOX_2D, bounds = BOX_2D)`
- `ST_Hilbert(box = BOX_2DF, bounds = BOX_2DF)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_InteriorRingN` *DuckDB function ST_InteriorRingN*

Description

Returns the N-th interior ring (hole) of a POLYGON as a LINESTRING. Indexing is 1-based (n = 1 returns the first interior ring). Returns NULL if the polygon is empty or has fewer than N interior rings.

Usage

```
ST_InteriorRingN(geom, n)
```

Arguments

geom	GEOMETRY
n	BIGINT

Value

```
GEOMETRY | LINESTRING_2D
```

Overloads

- `ST_InteriorRingN(geom = GEOMETRY, n = BIGINT)`
- `ST_InteriorRingN(polygon = POLYGON_2D, n = BIGINT)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_AsText(ST_InteriorRingN(ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(2 2,4 2,4
```

ST_InterpolatePoint *DuckDB function ST_InterpolatePoint*

Description

Computes the closest point on a LINESTRING to a given POINT and returns the interpolated M value of that point.

First argument must be a linestring and must have a M dimension. The second argument must be a point. Neither argument can be empty.

Usage

```
ST_InterpolatePoint(line, point)
```

Arguments

line, point	GEOMETRY

Value

```
DOUBLE
```

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Intersection` *DuckDB function ST_Intersection*

Description

Returns the intersection of two geometries.

Usage

```
ST_Intersection(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Intersection_Agg` *DuckDB function ST_Intersection_Agg*

Description

Computes the intersection of a set of geometries.

Usage

```
ST_Intersection_Agg(col0)
```

Arguments

`col0`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Intersects` *DuckDB function ST_Intersects*

Description

Returns true if the geometries intersect.

Usage

```
ST_Intersects(box1, box2)
```

Arguments

`box1`, `box2`	`BOX_2D`

Value

`BOOLEAN`

Overloads

- `ST_Intersects(box1 = BOX_2D, box2 = BOX_2D)`
- `ST_Intersects(geom1 = GEOMETRY, geom2 = GEOMETRY)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_IsClosed` *DuckDB function ST_IsClosed*

Description

Check if a geometry is 'closed'.

Usage

```
ST_IsClosed(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_IsEmpty` *DuckDB function ST_IsEmpty*

Description

Returns true if the geometry is ”empty”.

Usage

```
ST_IsEmpty(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`BOOLEAN`

Overloads

- `ST_IsEmpty(geom = GEOMETRY)`
- `ST_IsEmpty(linestring = LINESTRING_2D)`
- `ST_IsEmpty(polygon = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_IsRing` *DuckDB function ST_IsRing*

Description

Returns true if the geometry is a ring (both ST_IsClosed and ST_IsSimple).

Usage

```
ST_IsRing(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

BOOLEAN

Provided by

The spatial extension (LOAD spatial;).

ST_IsSimple — *DuckDB function ST_IsSimple*

Description

Returns true if the geometry is simple.

Usage

```
ST_IsSimple(geom)
```

Arguments

geom	GEOMETRY

Value

BOOLEAN

Provided by

The spatial extension (LOAD spatial;).

ST_IsValid — *DuckDB function ST_IsValid*

Description

Returns true if the geometry is valid.

Usage

```
ST_IsValid(geom)
```

Arguments

geom	GEOMETRY

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Length` *DuckDB function ST_Length*

Description

Returns the length of the input line geometry.

Usage

```
ST_Length(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Length(geom = GEOMETRY)`
- `ST_Length(linestring = LINESTRING_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Length_Spheroid` *DuckDB function ST_Length_Spheroid*

Description

Returns the length of the input geometry in meters, using an ellipsoidal model of the earth

The input geometry is assumed to be in the EPSG:4326 coordinate system (WGS84), with [latitude, longitude] axis order and the length is returned in meters. This function uses the GeographicLib library, calculating the length using an ellipsoidal model of the earth. This is a highly accurate method for calculating the length of a line geometry taking the curvature of the earth into account, but is also the slowest.

Returns `0.0` for any geometry that is not a `LINESTRING`, `MULTILINESTRING` or `GEOMETRYCOLLECTION` containing line geometries.

Usage

```
ST_Length_Spheroid(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Length_Spheroid(geom = GEOMETRY)`
- `ST_Length_Spheroid(line = LINESTRING_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_LineInterpolatePoint`
DuckDB function ST_LineInterpolatePoint

Description

Returns a point interpolated along a line at a fraction of total 2D length.

Usage

```
ST_LineInterpolatePoint(line, fraction)
```

Arguments

`line`	`GEOMETRY`
`fraction`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_LineInterpolatePoints`
DuckDB function ST_LineInterpolatePoints

Description

Returns a multi-point interpolated along a line at a fraction of total 2D length.

if repeat is false, the result is a single point, (and equivalent to ST_LineInterpolatePoint), otherwise, the result is a multi-point with points repeated at the fraction interval.

Usage

```
ST_LineInterpolatePoints(line, fraction, repeat.)
```

Arguments

`line`	`GEOMETRY`
`fraction`	`DOUBLE`
`repeat.`	`BOOLEAN`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

ST_LineLocatePoint *DuckDB function ST_LineLocatePoint*

Description

Returns the location on a line closest to a point as a fraction of the total 2D length of the line.

Usage

```
ST_LineLocatePoint(line, point)
```

Arguments

`line`, `point`	`GEOMETRY`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

ST_LineMerge *DuckDB function ST_LineMerge*

Description

"Merges" the input line geometry, optionally taking direction into account.

Usage

```
ST_LineMerge(geom, preserve_direction)
```

Arguments

`geom`	`GEOMETRY`
`preserve_direction`	`BOOLEAN`

Value

`GEOMETRY`

Overloads

- `ST_LineMerge(geom = GEOMETRY)`
- `ST_LineMerge(geom = GEOMETRY, preserve_direction = BOOLEAN)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_LineString2DFromWKB` *DuckDB function ST_LineString2DFromWKB*

Description

Deserialize a LINESTRING_2D from a WKB encoded blob.

Usage

```
ST_LineString2DFromWKB(blob)
```

Arguments

`blob`	`BLOB`

Value

`LINESTRING_2D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_LineSubstring` *DuckDB function ST_LineSubstring*

Description

Returns a substring of a line between two fractions of total 2D length.

Usage

```
ST_LineSubstring(line, start_fraction, end_fraction)
```

Arguments

line	GEOMETRY
start_fraction	DOUBLE
end_fraction	DOUBLE

Value

GEOMETRY

Provided by

The spatial extension (LOAD spatial;).

ST_LocateAlong *DuckDB function ST_LocateAlong*

Description

Returns a point or multi-point, containing the point(s) at the geometry with the given measure

For a LINESTRING, or MULTILINESTRING, the location is determined by interpolating between M values For a POINT and MULTIPOINT, the point is returned if the measure matches the M value of the vertex, otherwise an empty geometry is returned For a POLYGON, only the exterior ring is considered, and treated as a LINESTRING

If offset is provided, the resulting point(s) is offset by the given amount perpendicular to the line direction.

Usage

```
ST_LocateAlong(line, measure, offset)
```

Arguments

line	GEOMETRY
measure	DOUBLE
offset	DOUBLE

Value

GEOMETRY

Overloads

- ST_LocateAlong(line = GEOMETRY, measure = DOUBLE, offset = DOUBLE)
- ST_LocateAlong(line = GEOMETRY, measure = DOUBLE)

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_LocateBetween` *DuckDB function ST_LocateBetween*

Description

Returns a geometry or geometry collection created by filtering and interpolating vertices within a range of ”M” values

Creates a geometry or geometry collection, containing the parts formed by vertices that have an ”M” value within the ”start_measure” and ”end_measure” range

For LINESTRING or MULTILINESTRING, if a line segment would cross either the upper or lower bound, a vertex is added by interpolating the coordinates at the ”intersection” For a POINT and MULTIPOINT, the point is added to the collection if its vertex has an ”M” value within the range, otherwise it is skipped For a POLYGON, only the exterior ring is considered, and treated like a LINESTRING

If offset is provided, the resulting vertices are offset by the given amount perpendicular to the line direction.

Usage

```
ST_LocateBetween(line, start_measure, end_measure, offset)
```

Arguments

`line`	`GEOMETRY`
`start_measure`	`DOUBLE`
`end_measure`	`DOUBLE`
`offset`	`DOUBLE`

Value

`GEOMETRY`

Overloads

- `ST_LocateBetween(line = GEOMETRY, start_measure = DOUBLE, end_measure = DOUBLE, offset = DOUBLE)`
- `ST_LocateBetween(line = GEOMETRY, start_measure = DOUBLE, end_measure = DOUBLE)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_M` *DuckDB function ST_M*

Description

Returns the M coordinate of a point geometry.

Usage

```
ST_M(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_M(ST_Point(1, 2, 3, 4))
```

`ST_MakeBox2D` *DuckDB function ST_MakeBox2D*

Description

Create a BOX2D from two POINT geometries.

Usage

```
ST_MakeBox2D(point1, point2)
```

Arguments

`point1`, `point2`	`GEOMETRY`

Value

`BOX_2D`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_MakeBox2D(ST_Point(0, 0), ST_Point(1, 1));
-- BOX(0 0, 1 1)
```

`ST_MakeEnvelope` *DuckDB function ST_MakeEnvelope*

Description

Create a rectangular polygon from min/max coordinates.

Usage

```
ST_MakeEnvelope(min_x, min_y, max_x, max_y)
```

Arguments

`min_x`, `min_y`, `max_x`, `max_y`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_MakeLine` *DuckDB function ST_MakeLine*

Description

- `ST_MakeLine(geoms = `GEOMETRY[]`)`: Create a LINESTRING from a list of POINT geometries.
- `ST_MakeLine(start = GEOMETRY, end = GEOMETRY)`: Create a LINESTRING from two POINT geometries.

Usage

```
ST_MakeLine(start, end)
```

Arguments

start, end GEOMETRY

Value

GEOMETRY

Overloads

- ST_MakeLine(geoms = `GEOMETRY[]`)
- ST_MakeLine(start = GEOMETRY, end = GEOMETRY)

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
SELECT ST_MakeLine([ST_Point(0, 0), ST_Point(1, 1)]);
-- LINESTRING(0 0, 1 1)
SELECT ST_MakeLine(ST_Point(0, 0), ST_Point(1, 1));
-- LINESTRING(0 0, 1 1)
```

ST_MakePoint *DuckDB function ST_MakePoint*

Description

- ST_MakePoint(x = DOUBLE, y = DOUBLE): Creates a GEOMETRY point from an pair of floating point numbers.

  For geodetic coordinate systems, x is typically the longitude value and y is the latitude value.

  Note that ST_Point is equivalent. ST_MakePoint is provided for PostGIS compatibility.
- ST_MakePoint(x = DOUBLE, y = DOUBLE, z = DOUBLE): Creates a POINT_3D.
- ST_MakePoint(x = DOUBLE, y = DOUBLE, z = DOUBLE, m = DOUBLE): Creates a POINT_4D.

Usage

```
ST_MakePoint(x, y, z, m)
```

Arguments

x, y, z, m DOUBLE

Value

POINT_2D | POINT_3D | POINT_4D

Overloads

- `ST_MakePoint(x = DOUBLE, y = DOUBLE)`
- `ST_MakePoint(x = DOUBLE, y = DOUBLE, z = DOUBLE)`
- `ST_MakePoint(x = DOUBLE, y = DOUBLE, z = DOUBLE, m = DOUBLE)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_AsText(ST_MakePoint(143.3, -24.2));
-- POINT (143.3 -24.2)
```

`ST_MakePolygon` *DuckDB function ST_MakePolygon*

Description

- `ST_MakePolygon(shell = GEOMETRY)`: Create a POLYGON from a LINESTRING shell.
- `ST_MakePolygon(shell = GEOMETRY, holes = `GEOMETRY[]`)`: Create a POLYGON from a LINESTRING shell and a list of LINESTRING holes.

Usage

```
ST_MakePolygon(shell, holes)
```

Arguments

`shell`	`GEOMETRY`
`holes`	`GEOMETRY[]`

Value

`GEOMETRY`

Overloads

- `ST_MakePolygon(shell = GEOMETRY)`
- `ST_MakePolygon(shell = GEOMETRY, holes = `GEOMETRY[]`)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_MakePolygon(ST_LineString([ST_Point(0, 0), ST_Point(1, 0), ST_Point(1, 1), ST_Point(
SELECT ST_MakePolygon(ST_LineString([ST_Point(0, 0), ST_Point(1, 0), ST_Point(1, 1), ST_Point(
```

`ST_MakeValid` *DuckDB function ST_MakeValid*

Description

Returns a valid representation of the geometry.

Usage

```
ST_MakeValid(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_MaximumInscribedCircle`
DuckDB function ST_MaximumInscribedCircle

Description

Returns the maximum inscribed circle of the input geometry, optionally with a tolerance.

By default, the tolerance is computed as `max(width, height) / 1000`. The return value is a struct with the center of the circle, the nearest point to the center on the boundary of the geometry, and the radius of the circle.

Usage

```
ST_MaximumInscribedCircle(geom, tolerance)
```

Arguments

`geom`	`GEOMETRY`
`tolerance`	`DOUBLE`

Value

```
STRUCT(center GEOMETRY, nearest GEOMETRY, radius DOUBLE)
```

Overloads

- `ST_MaximumInscribedCircle(geom = GEOMETRY)`
- `ST_MaximumInscribedCircle(geom = GEOMETRY, tolerance = DOUBLE)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Find the maximum inscribed circle of a square
SELECT ST_MaximumInscribedCircle(
    ST_GeomFromText('POLYGON((0 0, 10 0, 10 10, 0 10, 0 0))')
);
-- {'center': POINT (5 5), 'nearest': POINT (5 0), 'radius': 5.0}
```

`ST_MemUnion_Agg` *DuckDB function ST_MemUnion_Agg*

Description

Computes the union of a set of input geometries. "Slower, but might be more memory efficient than ST_UnionAgg as each geometry is merged into the union individually rather than all at once.

Usage

```
ST_MemUnion_Agg(col0)
```

Arguments

`col0`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_MinimumRotatedRectangle` *DuckDB function ST_MinimumRotatedRectangle*

Description

Returns the minimum rotated rectangle that bounds the input geometry, finding the surrounding box that has the lowest area by using a rotated rectangle, rather than taking the lowest and highest coordinate values as per ST_Envelope().

Usage

```
ST_MinimumRotatedRectangle(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_MMax` *DuckDB function ST_MMax*

Description

Returns the maximum M coordinate of a geometry.

Usage

```
ST_MMax(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_MMax(ST_Point(1, 2, 3, 4))
```

`ST_MMin` *DuckDB function ST_MMin*

Description

Returns the minimum M coordinate of a geometry.

Usage

```
ST_MMin(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_MMin(ST_Point(1, 2, 3, 4))
```

`ST_Multi` *DuckDB function ST_Multi*

Description

Turns a single geometry into a multi geometry.

If the geometry is already a multi geometry, it is returned as is.

Usage

```
ST_Multi(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

GEOMETRY

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
SELECT ST_Multi(ST_GeomFromText('POINT(1 2)'));
-- MULTIPOINT (1 2)
--
-- SELECT ST_Multi(ST_GeomFromText('LINESTRING(1 1, 2 2)'));
-- MULTILINESTRING ((1 1, 2 2))
--
-- SELECT ST_Multi(ST_GeomFromText('POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))'));
-- MULTIPOLYGON (((0 0, 0 1, 1 1, 1 0, 0 0)))
```

ST_NGeometries *DuckDB function ST_NGeometries*

Description

Returns the number of component geometries in a collection geometry. If the input geometry is not a collection, this function returns 0 or 1 depending on if the geometry is empty or not.

Usage

```
ST_NGeometries(geom)
```

Arguments

geom	GEOMETRY

Value

INTEGER

Provided by

The spatial extension (LOAD spatial;).

`ST_NInteriorRings` *DuckDB function ST_NInteriorRings*

Description

Returns the number of interior rings of a polygon.

Usage

```
ST_NInteriorRings(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`INTEGER`

Overloads

- `ST_NInteriorRings(geom = GEOMETRY)`
- `ST_NInteriorRings(polygon = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Node` *DuckDB function ST_Node*

Description

Returns a "noded" MultiLinestring, produced by combining a collection of input linestrings and adding additional vertices where they intersect.

Usage

```
ST_Node(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Create a noded multilinestring from two intersecting lines
SELECT ST_Node(
    ST_GeomFromText('MULTILINESTRING((0 0, 2 2), (0 2, 2 0))')
);
-- MULTILINESTRING ((0 0, 1 1), (1 1, 2 2), (0 2, 1 1), (1 1, 2 0))
```

`ST_Normalize` *DuckDB function ST_Normalize*

Description

Returns the ”normalized” representation of the geometry.

Usage

```
ST_Normalize(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_NPoints` *DuckDB function ST_NPoints*

Description

Returns the number of vertices within a geometry.

Usage

```
ST_NPoints(geom)
```

Arguments

geom	GEOMETRY

Value

UINTEGER | UBIGINT

Overloads

- ST_NPoints(geom = GEOMETRY)
- ST_NPoints(point = POINT_2D)
- ST_NPoints(linestring = LINESTRING_2D)
- ST_NPoints(polygon = POLYGON_2D)
- ST_NPoints(box = BOX_2D)

Provided by

The spatial extension (LOAD spatial;).

ST_NumGeometries *DuckDB function ST_NumGeometries*

Description

Returns the number of component geometries in a collection geometry. If the input geometry is not a collection, this function returns 0 or 1 depending on if the geometry is empty or not.

Usage

```
ST_NumGeometries(geom)
```

Arguments

geom	GEOMETRY

Value

INTEGER

Provided by

The spatial extension (LOAD spatial;).

`ST_NumInteriorRings` *DuckDB function ST_NumInteriorRings*

Description

Returns the number of interior rings of a polygon.

Usage

```
ST_NumInteriorRings(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`INTEGER`

Overloads

- `ST_NumInteriorRings(geom = GEOMETRY)`
- `ST_NumInteriorRings(polygon = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_NumPoints` *DuckDB function ST_NumPoints*

Description

Returns the number of vertices within a geometry.

Usage

```
ST_NumPoints(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`UINTEGER | UBIGINT`

Overloads

- `ST_NumPoints(geom = GEOMETRY)`
- `ST_NumPoints(point = POINT_2D)`
- `ST_NumPoints(linestring = LINESTRING_2D)`
- `ST_NumPoints(polygon = POLYGON_2D)`
- `ST_NumPoints(box = BOX_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Overlaps` *DuckDB function ST_Overlaps*

Description

Returns true if the geometries overlap.

Usage

```
ST_Overlaps(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Perimeter` *DuckDB function ST_Perimeter*

Description

Returns the length of the perimeter of the geometry.

Usage

```
ST_Perimeter(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Perimeter(geom = GEOMETRY)`
- `ST_Perimeter(polygon = POLYGON_2D)`
- `ST_Perimeter(box = BOX_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Perimeter_Spheroid`
DuckDB function ST_Perimeter_Spheroid

Description

Returns the length of the perimeter in meters using an ellipsoidal model of the earths surface

The input geometry is assumed to be in the EPSG:4326 coordinate system (WGS84), with [latitude, longitude] axis order and the length is returned in meters. This function uses the GeographicLib library, calculating the perimeter using an ellipsoidal model of the earth. This is a highly accurate method for calculating the perimeter of a polygon taking the curvature of the earth into account, but is also the slowest.

Returns `0.0` for any geometry that is not a `POLYGON`, `MULTIPOLYGON` or `GEOMETRYCOLLECTION` containing polygon geometries.

Usage

```
ST_Perimeter_Spheroid(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Perimeter_Spheroid(geom = GEOMETRY)`
- `ST_Perimeter_Spheroid(poly = POLYGON_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Point` *DuckDB function ST_Point*

Description

Creates a GEOMETRY point.

Usage

```
ST_Point(x, y)
```

Arguments

`x`, `y`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Point2D` *DuckDB function ST_Point2D*

Description

Creates a POINT_2D.

Usage

```
ST_Point2D(x, y)
```

Arguments

x, y	DOUBLE

Value

`POINT_2D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Point2DFromWKB` *DuckDB function ST_Point2DFromWKB*

Description

Deserialize a POINT_2D from a WKB encoded blob.

Usage

```
ST_Point2DFromWKB(blob)
```

Arguments

blob	BLOB

Value

`POINT_2D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Point3D` *DuckDB function ST_Point3D*

Description

Creates a POINT_3D.

Usage

```
ST_Point3D(x, y, z)
```

Arguments

x, y, z	DOUBLE

Value

`POINT_3D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Point4D` *DuckDB function ST_Point4D*

Description

Creates a POINT_4D.

Usage

```
ST_Point4D(x, y, z, m)
```

Arguments

x, y, z, m	DOUBLE

Value

`POINT_4D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_PointN` *DuckDB function ST_PointN*

Description

Returns the n'th vertex from the input geometry as a point geometry.

Usage

```
ST_PointN(geom, index)
```

Arguments

`geom`	`GEOMETRY`
`index`	`INTEGER`

Value

`GEOMETRY | POINT_2D`

Overloads

- `ST_PointN(geom = GEOMETRY, index = INTEGER)`
- `ST_PointN(linestring = LINESTRING_2D, index = INTEGER)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_PointOnSurface` *DuckDB function ST_PointOnSurface*

Description

Returns a point guaranteed to lie on the surface of the geometry.

Usage

```
ST_PointOnSurface(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Points` *DuckDB function ST_Points*

Description

Collects all the vertices in the geometry into a MULTIPOINT.

Usage

```
ST_Points(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Points('LINESTRING(1 1, 2 2)'::GEOMETRY);
-- MULTIPOINT (1 1, 2 2)
--
-- SELECT ST_Points('MULTIPOLYGON Z EMPTY'::GEOMETRY);
-- MULTIPOINT Z EMPTY
```

`ST_Polygon2DFromWKB` *DuckDB function ST_Polygon2DFromWKB*

Description

Deserialize a POLYGON_2D from a WKB encoded blob.

Usage

```
ST_Polygon2DFromWKB(blob)
```

Arguments

`blob`	`BLOB`

Value

`POLYGON_2D`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Polygonize` *DuckDB function ST_Polygonize*

Description

Returns a polygonized representation of the input geometries.

Usage

```
ST_Polygonize(geometries)
```

Arguments

`geometries`	`GEOMETRY[]`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Create a polygon from a closed linestring ring
SELECT ST_Polygonize([
    ST_GeomFromText('LINESTRING(0 0, 0 10, 10 10, 10 0, 0 0)')
]);
---
GEOMETRYCOLLECTION (POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0)))
```

`ST_QuadKey` *DuckDB function ST_QuadKey*

Description

Compute the quadkey for a given lon/lat point at a given level. Note that the parameter order is **longitude**, **latitude**.

`level` has to be between 1 and 23, inclusive.

The input coordinates will be clamped to the lon/lat bounds of the earth (longitude between -180 and 180, latitude between -85.05112878 and 85.05112878).

The geometry overload throws an error if the input geometry is not a `POINT`.

Usage

```
ST_QuadKey(longitude, latitude, level)
```

Arguments

`longitude`	`DOUBLE`
`latitude`	`DOUBLE`
`level`	`INTEGER`

Value

`VARCHAR`

Overloads

- `ST_QuadKey(longitude = DOUBLE, latitude = DOUBLE, level = INTEGER)`
- `ST_QuadKey(point = GEOMETRY, level = INTEGER)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_QuadKey(ST_Point(11.08, 49.45), 10);
-- 1333203202
```

`ST_Read` *DuckDB function ST_Read*

Description

Read and import a variety of geospatial file formats using the GDAL library.

The `ST_Read` table function is based on the GDAL translator library and enables reading spatial data from a variety of geospatial vector file formats as if they were DuckDB tables.

See ST_Drivers for a list of supported file formats and drivers.

Except for the `path` parameter, all parameters are optional.

Parameter	Type	Description
`path`	VARCHAR	The path to the file to read. Mandatory
`sequential_layer_scan`	BOOLEAN	If set to true, the table function will scan through all layers sequentiall
`spatial_filter`	WKB_BLOB	If set to a WKB blob, the table function will only return rows that int
`open_options`	VARCHAR[]	A list of key-value pairs that are passed to the GDAL driver to control
`layer`	VARCHAR	The name of the layer to read from the file. If NULL, the first layer is
`allowed_drivers`	VARCHAR[]	A list of GDAL driver names that are allowed to be used to open the f
`sibling_files`	VARCHAR[]	A list of sibling files that are required to open the file. E.g., the ESRI
`spatial_filter_box`	BOX_2D	If set to a BOX_2D, the table function will only return rows that inte
`keep_wkb`	BOOLEAN	If set, the table function will return geometries in a wkb_geometry col

Note that GDAL is single-threaded, so this table function will not be able to make full use of parallelism.

By using `ST_Read`, the spatial extension also provides "replacement scans" for common geospatial file formats, allowing you to query files of these formats as if they were tables directly.

```
SELECT * FROM './path/to/some/shapefile/dataset.shp';
```

In practice this is just syntax-sugar for calling ST_Read, so there is no difference in performance. If you want to pass additional options, you should use the ST_Read table function directly.

The following formats are currently recognized by their file extension:

Format	Extension
ESRI ShapeFile	.shp
GeoPackage	.gpkg
FlatGeoBuf	.fgb

Usage

```
ST_Read(
  col0,
  keep_wkb,
  max_batch_size,
  layer,
  sibling_files,
  allowed_drivers,
  open_options
)
```

Arguments

`col0`	`VARCHAR`
`keep_wkb`	`BOOLEAN`
`max_batch_size`	`INTEGER`
`layer`	`VARCHAR`
`sibling_files`	`VARCHAR[]`
`allowed_drivers`	`VARCHAR[]`
`open_options`	`VARCHAR[]`

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Read a Shapefile
SELECT * FROM ST_Read('some/file/path/filename.shp');

-- Read a GeoJSON file
CREATE TABLE my_geojson_table AS SELECT * FROM ST_Read('some/file/path/filename.json');
```

ST_Read_Meta *DuckDB function ST_Read_Meta*

Description

Read the metadata from a variety of geospatial file formats using the GDAL library.

The `ST_Read_Meta` table function accompanies the `ST_Read` table function, but instead of reading the contents of a file, this function scans the metadata instead. Since the data model of the underlying GDAL library is quite flexible, most of the interesting metadata is within the returned `layers` column, which is a somewhat complex nested structure of DuckDB `STRUCT` and `LIST` types.

Usage

```
ST_Read_Meta(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Find the coordinate reference system authority name and code for the first layers first geor
SELECT
    layers[1].geometry_fields[1].crs.auth_name as name,
    layers[1].geometry_fields[1].crs.auth_code as code
FROM st_read_meta('../../tmp/data/amsterdam_roads.fgb');
```

ST_ReadOSM *DuckDB function ST_ReadOSM*

Description

The `ST_ReadOsm()` table function enables reading compressed OpenStreetMap data directly from a `.osm.pbf` file.

This function uses multithreading and zero-copy protobuf parsing which makes it a lot faster than using the `ST_Read()` OSM driver, however it only outputs the raw OSM data (Nodes, Ways, Relations), without constructing any geometries. For simple node entities (like PoI's) you can trivially construct POINT geometries, but it is also possible to construct LINESTRING and POLYGON geometries by manually joining refs and nodes together in SQL, although with available memory usually being a limiting factor. The `ST_ReadOSM()` function also provides a "replacement scan" to enable reading from a file directly as if it were a table. This is just syntax sugar for calling `ST_ReadOSM()` though. Example:

```
SELECT * FROM 'tmp/data/germany.osm.pbf' LIMIT 5;
```.

Usage

```
ST_ReadOSM(col0)
```

Arguments

```
col0           VARCHAR
```

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT *
FROM ST_ReadOSM('tmp/data/germany.osm.pbf')
WHERE tags['highway'] != []
LIMIT 5;
-- +----------------------+--------+----------------------+---------+--------------------+-----
-- |         kind         |   id   |         tags         |  refs   |        lat         |    l
-- | enum('node', 'way'…  | int64  | map(varchar, varch…  | int64[] |       double       |   do
-- +----------------------+--------+----------------------+---------+--------------------+-----
-- | node                 | 122351 | {bicycle=yes, butt…  |         |         53.5492951 |   9.
-- | node                 | 122397 | {crossing=no, high…  |         | 53.520990100000006 | 10.0
-- | node                 | 122493 | {TMC:cid_58:tabcd_…  |         | 53.129614600000004 |  8.1
-- | node                 | 123566 | {highway=traffic_s…  |         | 54.617268200000005 |  8.9
-- | node                 | 125801 | {TMC:cid_58:tabcd_…  |         | 53.070685000000005 |  8.7
-- +----------------------+--------+----------------------+---------+--------------------+-----
```

`ST_ReadSHP` *DuckDB function ST_ReadSHP*

Description

Read a Shapefile without relying on the GDAL library.

Usage

```
ST_ReadSHP(col0, encoding)
```

Arguments

`col0`, `encoding` `VARCHAR`

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_ReducePrecision` *DuckDB function ST_ReducePrecision*

Description

Returns the geometry with all vertices reduced to the given precision.

Usage

```
ST_ReducePrecision(geom, precision)
```

Arguments

`geom`	`GEOMETRY`
`precision`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_RemoveRepeatedPoints`
DuckDB function ST_RemoveRepeatedPoints

Description

- `ST_RemoveRepeatedPoints(line = LINESTRING_2D)`, `ST_RemoveRepeatedPoints(line = LINESTRING_2D, tolerance = DOUBLE)`: Remove repeated points from a LINESTRING.
- `ST_RemoveRepeatedPoints(geom = GEOMETRY)`, `ST_RemoveRepeatedPoints(geom = GEOMETRY, tolerance = DOUBLE)`: Returns the geometry with repeated points removed.

Usage

```
ST_RemoveRepeatedPoints(line, tolerance)
```

Arguments

`line`	`LINESTRING_2D`
`tolerance`	`DOUBLE`

Value

`LINESTRING_2D | GEOMETRY`

Overloads

- `ST_RemoveRepeatedPoints(line = LINESTRING_2D)`
- `ST_RemoveRepeatedPoints(line = LINESTRING_2D, tolerance = DOUBLE)`
- `ST_RemoveRepeatedPoints(geom = GEOMETRY)`
- `ST_RemoveRepeatedPoints(geom = GEOMETRY, tolerance = DOUBLE)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Reverse` *DuckDB function ST_Reverse*

Description

Returns the geometry with the order of its vertices reversed.

Usage

```
ST_Reverse(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Rotate` *DuckDB function ST_Rotate*

Description

Alias of ST_RotateZ.

Usage

```
ST_Rotate(geom, radians)
```

Arguments

`geom`, `radians`	Unspecified.

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_RotateX` *DuckDB function ST_RotateX*

Description

Rotates a geometry around the X axis. This is a shorthand macro for calling ST_Affine.

Usage

```
ST_RotateX(geom, radians)
```

Arguments

`geom`, `radians` Unspecified.

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Rotate a 3D point 90 degrees ( /2 radians) around the X-axis
SELECT ST_RotateX(ST_GeomFromText('POINT Z(0 1 0)'), pi()/2);
-- POINT Z (0 0 1)
```

`ST_RotateY` *DuckDB function ST_RotateY*

Description

Rotates a geometry around the Y axis. This is a shorthand macro for calling ST_Affine.

Usage

```
ST_RotateY(geom, radians)
```

Arguments

`geom`, `radians` Unspecified.

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Rotate a 3D point 90 degrees ( /2 radians) around the Y-axis
SELECT ST_RotateY(ST_GeomFromText('POINT Z(1 0 0)'), pi()/2);
-- POINT Z (0 0 -1)
```

`ST_RotateZ` *DuckDB function ST_RotateZ*

Description

Rotates a geometry around the Z axis. This is a shorthand macro for calling ST_Affine.

Usage

```
ST_RotateZ(geom, radians)
```

Arguments

`geom`, `radians` Unspecified.

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Rotate a point 90 degrees ( /2 radians) around the Z-axis
SELECT ST_RotateZ(ST_Point(1, 0), pi()/2);
-- POINT (0 1)
```

`ST_Scale` *DuckDB function ST_Scale*

Description

DuckDB function `ST_Scale()`.

Usage

```
ST_Scale(geom, xs, ys, zs)
```

Arguments

`geom`, `xs`, `ys`, `zs` Unspecified.

Value

Unspecified.

Overloads

- `ST_Scale(geom, xs, ys, zs)`
- `ST_Scale(geom, xs, ys)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`st_setcrs` *DuckDB function st_setcrs*

Description

Sets the Coordinate Reference System (CRS) identifier of the geometry.

Usage

```
st_setcrs(geom, crs)
```

Arguments

`geom`	`GEOMETRY`
`crs`	`VARCHAR`

Value

`GEOMETRY`

`ST_ShortestLine` *DuckDB function ST_ShortestLine*

Description

Returns the shortest line between two geometries.

Usage

```
ST_ShortestLine(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Simplify` *DuckDB function ST_Simplify*

Description

Returns a simplified version of the geometry.

Usage

```
ST_Simplify(geom, tolerance)
```

Arguments

`geom`	`GEOMETRY`
`tolerance`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_SimplifyPreserveTopology`
DuckDB function ST_SimplifyPreserveTopology

Description

Returns a simplified version of the geometry that preserves topology.

Usage

```
ST_SimplifyPreserveTopology(geom, tolerance)
```

Arguments

`geom`	`GEOMETRY`
`tolerance`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Snap` *DuckDB function ST_Snap*

Description

Snaps the vertices and segments of a geometry to another geometry's vertices within the given tolerance.

Usage

```
ST_Snap(geom, snap_to, tolerance)
```

Arguments

`geom`	`GEOMETRY`
`snap_to`	`GEOMETRY`
`tolerance`	`DOUBLE`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_StartPoint` *DuckDB function ST_StartPoint*

Description

Returns the start point of a LINESTRING.

Usage

```
ST_StartPoint(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY | POINT_2D`

Overloads

- `ST_StartPoint(geom = GEOMETRY)`
- `ST_StartPoint(line = LINESTRING_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_SymDifference` *DuckDB function ST_SymDifference*

Description

Returns the symmetric difference of two geometries.

Usage

```
ST_SymDifference(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_TileEnvelope` *DuckDB function ST_TileEnvelope*

Description

The `ST_TileEnvelope` scalar function generates tile envelope rectangular polygons from specified zoom level and tile indices.

This is used in MVT generation to select the features corresponding to the tile extent. The envelope is in the Web Mercator coordinate reference system (EPSG:3857). The tile pyramid starts at zoom level 0, corresponding to a single tile for the world. Each zoom level doubles the number of tiles in each direction, such that zoom level 1 is 2 tiles wide by 2 tiles high, zoom level 2 is 4 tiles wide by 4 tiles high, and so on. Tile indices start at `[x=0, y=0]` at the top left, and increase down and right. For example, at zoom level 2, the top right tile is `[x=3, y=0]`, the bottom left tile is `[x=0, y=3]`, and the bottom right is `[x=3, y=3]`.

```
SELECT ST_TileEnvelope(2, 3, 1);
```.

```
[x=0, y=0]: R:x=0,%20y=0
[x=3, y=0]: R:x=3,%20y=0
[x=0, y=3]: R:x=0,%20y=3
[x=3, y=3]: R:x=3,%20y=3
```

Usage

```
ST_TileEnvelope(tile_zoom, tile_x, tile_y)
```

Arguments

`tile_zoom`, `tile_x`, `tile_y`
INTEGER

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_TileEnvelope(2, 3, 1);
-- +-----------------------------------------------------------------------------------------------
-- |                                          st_tileenvelope(2, 3, 1)
-- |                                                  geometry
-- +-----------------------------------------------------------------------------------------------
-- | POLYGON ((1.00188E+07 0, 1.00188E+07 1.00188E+07, 2.00375E+07 1.00188E+07, 2.00375E+07 0,
-- +-----------------------------------------------------------------------------------------------
```

ST_Touches *DuckDB function ST_Touches*

Description

Returns true if the geometries touch.

Usage

```
ST_Touches(geom1, geom2)
```

Arguments

geom1, geom2	GEOMETRY

Value

BOOLEAN

Provided by

The `spatial` extension (`LOAD spatial;`).

ST_Transform *DuckDB function ST_Transform*

Description

Transforms a geometry between two coordinate systems

The source and target coordinate systems can be specified using any format that the PROJ library supports.

The third optional `always_xy` parameter can be used to force the input and output geometries to be interpreted as having a [easting, northing] coordinate axis order regardless of what the source and target coordinate system definition says. This is particularly useful when transforming to/from the WGS84/EPSG:4326 coordinate system (what most people think of when they hear ”longitude”/”latitude” or ”GPS coordinates”), which is defined as

having a [latitude, longitude] axis order even though [longitude, latitude] is commonly used in practice (e.g. in GeoJSON). More details available in the PROJ documentation.

DuckDB spatial vendors its own static copy of the PROJ database of coordinate systems, so if you have your own installation of PROJ on your system the available coordinate systems may differ to what's available in other GIS software.

Usage

```
ST_Transform(box, source_crs, target_crs, always_xy)
```

Arguments

`box`	`BOX_2D`
`source_crs`	`VARCHAR`
`target_crs`	`VARCHAR`
`always_xy`	`BOOLEAN`

Value

`BOX_2D | POINT_2D | GEOMETRY`

Overloads

- `ST_Transform(box = BOX_2D, source_crs = VARCHAR, target_crs = VARCHAR)`
- `ST_Transform(box = BOX_2D, source_crs = VARCHAR, target_crs = VARCHAR, always_xy = BOOLEAN)`
- `ST_Transform(point = POINT_2D, source_crs = VARCHAR, target_crs = VARCHAR)`
- `ST_Transform(point = POINT_2D, source_crs = VARCHAR, target_crs = VARCHAR, always_xy = BOOLEAN)`
- `ST_Transform(geom = GEOMETRY, source_crs = VARCHAR, target_crs = VARCHAR)`
- `ST_Transform(geom = GEOMETRY, source_crs = VARCHAR, target_crs = VARCHAR, always_xy = BOOLEAN)`
- `ST_Transform(geom = GEOMETRY, target_crs = VARCHAR)`
- `ST_Transform(geom = GEOMETRY, target_crs = VARCHAR, always_xy = BOOLEAN)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Transform a geometry from EPSG:4326 to EPSG:3857 (WGS84 to WebMercator)
-- Note that since WGS84 is defined as having a [latitude, longitude] axis order
-- we follow the standard and provide the input geometry using that axis order,
-- but the output will be [easting, northing] because that is what's defined by
-- WebMercator.
```

```
SELECT
    ST_Transform(
        st_point(52.373123, 4.892360),
        'EPSG:4326',
        'EPSG:3857'
    );
-- POINT (544615.0239773799 6867874.103539125)
--
-- -- Alternatively, let's say we got our input point from e.g. a GeoJSON file,
-- -- which uses WGS84 but with [longitude, latitude] axis order. We can use the
-- -- `always_xy` parameter to force the input geometry to be interpreted as having
-- -- a [northing, easting] axis order instead, even though the source coordinate
-- -- reference system definition (WGS84) says otherwise.
--
-- SELECT
--     ST_Transform(
--         -- note the axis order is reversed here
--         st_point(4.892360, 52.373123),
--         'EPSG:4326',
--         'EPSG:3857',
--         always_xy := true
--     );
-- POINT (544615.0239773799 6867874.103539125)
--
-- -- Transform a geometry from OSG36 British National Grid EPSG:27700 to EPSG:4326 WGS84
-- -- Standard transform is often fine for the first few decimal places before being wrong
-- -- which could result in an error starting at about 10m and possibly much more
-- SELECT ST_Transform(bng, 'EPSG:27700', 'EPSG:4326', xy := true) AS without_grid_file
-- FROM (SELECT ST_GeomFromText('POINT( 170370.718 11572.405 )') AS bng);
-- POINT (-5.202992651563592 49.96007490162923)
--
-- -- By using an official NTv2 grid file, we can reduce the error down around the 9th decimal
-- -- which in theory is below a millimetre, and in practise unlikely that your coordinates are
-- -- British National Grid "NTv2 format files" download available here:
-- -- https://www.ordnancesurvey.co.uk/products/os-net/for-developers
-- SELECT ST_Transform(bng
--     , '+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=airy
--     , 'EPSG:4326', xy := true) AS with_grid_file
-- FROM (SELECT ST_GeomFromText('POINT( 170370.718 11572.405 )') AS bng) t;
-- POINT (-5.203046090608746 49.96006137018598)
```

`ST_Translate` *DuckDB function ST_Translate*

Description

DuckDB function `ST_Translate()`.

Usage

```
ST_Translate(geom, dx, dy, dz)
```

Arguments

`geom`, `dx`, `dy`, `dz` Unspecified.

Value

Unspecified.

Overloads

- `ST_Translate(geom, dx, dy, dz)`
- `ST_Translate(geom, dx, dy)`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_TransScale` *DuckDB function ST_TransScale*

Description

Translates and then scales a geometry in X and Y direction. This is a shorthand macro for calling ST_Affine.

Usage

```
ST_TransScale(geom, dx, dy, xs, ys)
```

Arguments

`geom`, `dx`, `dy`, `xs`, `ys`
Unspecified.

Value

Unspecified.

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
-- Translate by (1, 2) then scale by (2, 3)
SELECT ST_TransScale(ST_Point(1, 1), 1, 2, 2, 3);
-- POINT (4 9)
```

`ST_Union` *DuckDB function ST_Union*

Description

Returns the union of two geometries.

Usage

```
ST_Union(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Union_Agg` *DuckDB function ST_Union_Agg*

Description

Computes the union of a set of input geometries.

Usage

```
ST_Union_Agg(col0)
```

Arguments

`col0`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_VoronoiDiagram` *DuckDB function ST_VoronoiDiagram*

Description

Returns the Voronoi diagram of the supplied MultiPoint geometry.

Usage

```
ST_VoronoiDiagram(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`GEOMETRY`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_Within` *DuckDB function ST_Within*

Description

Returns true if the first geometry is within the second.

Usage

```
ST_Within(geom1, geom2)
```

Arguments

`geom1`	`POINT_2D | GEOMETRY`
`geom2`	`POLYGON_2D | GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_WithinProperly` *DuckDB function ST_WithinProperly*

Description

Returns true if the first geometry \"properly\" is contained by the second geometry
This function functions the same as `ST_ContainsProperly`, but the arguments are swapped.

Usage

```
ST_WithinProperly(geom1, geom2)
```

Arguments

`geom1`, `geom2`	`GEOMETRY`

Value

`BOOLEAN`

Provided by

The `spatial` extension (`LOAD spatial;`).

`ST_X` *DuckDB function ST_X*

Description

Returns the X coordinate of a point geometry.

Usage

```
ST_X(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_X(geom = GEOMETRY)`
- `ST_X(point = POINT_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_X(ST_Point(1, 2))
```

`ST_XMax` *DuckDB function ST_XMax*

Description

Returns the maximum X coordinate of a geometry.

Usage

```
ST_XMax(box)
```

Arguments

`box`	`BOX_2D | BOX_2DF`

Value

`DOUBLE | FLOAT`

Overloads

- `ST_XMax(geom = GEOMETRY)`
- `ST_XMax(point = POINT_2D)`
- `ST_XMax(line = LINESTRING_2D)`
- `ST_XMax(polygon = POLYGON_2D)`
- `ST_XMax(box = BOX_2D)`
- `ST_XMax(box = BOX_2DF)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_XMax(ST_Point(1, 2))
```

`ST_XMin` *DuckDB function ST_XMin*

Description

Returns the minimum X coordinate of a geometry.

Usage

```
ST_XMin(box)
```

Arguments

box	BOX_2D | BOX_2DF

Value

`DOUBLE | FLOAT`

Overloads

- `ST_XMin(geom = GEOMETRY)`
- `ST_XMin(point = POINT_2D)`
- `ST_XMin(line = LINESTRING_2D)`
- `ST_XMin(polygon = POLYGON_2D)`
- `ST_XMin(box = BOX_2D)`
- `ST_XMin(box = BOX_2DF)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_XMin(ST_Point(1, 2))
```

`ST_Y` *DuckDB function ST_Y*

Description

Returns the Y coordinate of a point geometry.

Usage

```
ST_Y(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Overloads

- `ST_Y(geom = GEOMETRY)`
- `ST_Y(point = POINT_2D)`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Y(ST_Point(1, 2))
```

`ST_YMax` *DuckDB function ST_YMax*

Description

Returns the maximum Y coordinate of a geometry.

Usage

```
ST_YMax(box)
```

Arguments

`box`	`BOX_2D | BOX_2DF`

Value

DOUBLE | FLOAT

Overloads

- ST_YMax(geom = GEOMETRY)
- ST_YMax(point = POINT_2D)
- ST_YMax(line = LINESTRING_2D)
- ST_YMax(polygon = POLYGON_2D)
- ST_YMax(box = BOX_2D)
- ST_YMax(box = BOX_2DF)

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
SELECT ST_YMax(ST_Point(1, 2))
```

ST_YMin — *DuckDB function ST_YMin*

Description

Returns the minimum Y coordinate of a geometry.

Usage

```
ST_YMin(box)
```

Arguments

box	BOX_2D | BOX_2DF

Value

DOUBLE | FLOAT

Overloads

- ST_YMin(geom = GEOMETRY)
- ST_YMin(point = POINT_2D)
- ST_YMin(line = LINESTRING_2D)
- ST_YMin(polygon = POLYGON_2D)
- ST_YMin(box = BOX_2D)
- ST_YMin(box = BOX_2DF)

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_YMin(ST_Point(1, 2))
```

`ST_Z` *DuckDB function ST_Z*

Description

Returns the Z coordinate of a point geometry.

Usage

```
ST_Z(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_Z(ST_Point(1, 2, 3))
```

`ST_ZMax` *DuckDB function ST_ZMax*

Description

Returns the maximum Z coordinate of a geometry.

Usage

```
ST_ZMax(geom)
```

Arguments

geom	GEOMETRY

Value

DOUBLE

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
SELECT ST_ZMax(ST_Point(1, 2, 3))
```

ST_ZMFlag *DuckDB function ST_ZMFlag*

Description

Returns a flag indicating the presence of Z and M values in the input geometry. 0 = No Z or M values 1 = M values only 2 = Z values only 3 = Z and M values.

Usage

```
ST_ZMFlag(geom)
```

Arguments

geom	GEOMETRY

Value

UTINYINT

Provided by

The spatial extension (LOAD spatial;).

SQL examples

```
-- ZMFlag for a 2D geometry
SELECT ST_ZMFlag(ST_GeomFromText('POINT(1 1)'));
-- 0
--
-- -- ZMFlag for a 3DZ geometry
-- SELECT ST_ZMFlag(ST_GeomFromText('POINT Z(1 1 1)'));
-- 2
--
```

```
-- -- ZMFlag for a 3DM geometry
-- SELECT ST_ZMFlag(ST_GeomFromText('POINT M(1 1 1)'));
-- 1
--
-- -- ZMFlag for a 4D geometry
-- SELECT ST_ZMFlag(ST_GeomFromText('POINT ZM(1 1 1 1)'));
-- 3
```

`ST_ZMin` *DuckDB function ST_ZMin*

Description

Returns the minimum Z coordinate of a geometry.

Usage

```
ST_ZMin(geom)
```

Arguments

`geom`	`GEOMETRY`

Value

`DOUBLE`

Provided by

The `spatial` extension (`LOAD spatial;`).

SQL examples

```
SELECT ST_ZMin(ST_Point(1, 2, 3))
```

`start_ui` *DuckDB function start_ui*

Description

DuckDB function `start_ui()`.

Usage

```
start_ui()
```

Value

Unspecified.

Provided by

The `ui` extension (`LOAD ui;`).

`start_ui_server` *DuckDB function start_ui_server*

Description

DuckDB function `start_ui_server()`.

Usage

```
start_ui_server()
```

Value

Unspecified.

Provided by

The `ui` extension (`LOAD ui;`).

`starts_with` *DuckDB function starts_with*

Description

Returns `true` if `string` begins with `search_string`.

Usage

```
starts_with(string, search_string)
```

Arguments

`string`, `search_string`
: `VARCHAR`

Value

`BOOLEAN`

SQL examples

```
starts_with('abc', 'a')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

stats *DuckDB function stats*

Description

Returns a string with statistics about the expression. Expression can be a column, constant, or SQL expression.

Usage

```
stats(expression)
```

Arguments

expression	ANY

Value

VARCHAR

SQL examples

```
stats(5)
```

`stddev_pop` *DuckDB function stddev_pop*

Description

Returns the population standard deviation.

Usage

```
stddev_pop(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
sqrt(var_pop(x))
```

`stddev_samp` *DuckDB function stddev_samp*

Description

Returns the sample standard deviation.

Usage

```
stddev_samp(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
sqrt(var_samp(x))
```

`stem` *DuckDB function stem*

Description

DuckDB function `stem()`.

Usage

```
stem(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `fts` extension (`LOAD fts;`).

`stop_ui_server` *DuckDB function stop_ui_server*

Description

DuckDB function `stop_ui_server()`.

Usage

```
stop_ui_server()
```

Value

Unspecified.

Provided by

The `ui` extension (`LOAD ui;`).

`storage_info` *DuckDB function storage_info*

Description

DuckDB function `storage_info()`.

Usage

```
storage_info(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`strftime` *DuckDB function strftime*

Description

Converts a `date` to a string according to the format string.

Usage

```
strftime(data, format)
```

Arguments

`data`	`DATE | TIMESTAMP | TIMESTAMP_NS | VARCHAR | TIMESTAMP WITH TIME ZONE`
`format`	`VARCHAR | DATE | TIMESTAMP | TIMESTAMP_NS | TIMESTAMP WITH TIME ZONE`

Value

`VARCHAR`

SQL examples

```
strftime(date '1992-01-01', '%a, %-d %B %Y')
```

`string_agg` *DuckDB function string_agg*

Description

Concatenates the column string values with an optional separator.

Usage

```
string_agg(str, arg)
```

Arguments

`str`	`ANY`
`arg`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `string_agg(str = ANY)`
- `string_agg(str = ANY, arg = VARCHAR)`

SQL examples

```
string_agg(A, '-')
```

`string_split` *DuckDB function string_split*

Description

Splits the `string` along the `separator`.

Usage

```
string_split(string, separator)
```

Arguments

`string`, `separator`	`VARCHAR`

Value

VARCHAR[]

SQL examples

```
string_split('hello-world', '-')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

---

string_split_regex *DuckDB function string_split_regex*

Description

Splits the string along the regex. A set of optional regex options can be set.

Usage

```
string_split_regex(string, regex, options)
```

Arguments

string, regex, options	VARCHAR

Value

VARCHAR[]

Overloads

- string_split_regex(string = VARCHAR, regex = VARCHAR)
- string_split_regex(string = VARCHAR, regex = VARCHAR, options = VARCHAR)

SQL examples

```
string_split_regex('hello world; 42', ';? ')
```

See Also

Other regex: `regexp_escape()`, `regexp_extract()`, `regexp_extract_all()`, `regexp_full_match()`, `regexp_matches()`, `regexp_replace()`

`strip_accents` *DuckDB function strip_accents*

Description

Strips accents from `string`.

Usage

```
strip_accents(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
strip_accents('mühleisen')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`strlen` *DuckDB function strlen*

Description

Number of bytes in `string`.

Usage

```
strlen(string)
```

Arguments

`string`	`VARCHAR`

Value

`BIGINT`

SQL examples

```
strlen(' ')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`strptime` *DuckDB function strptime*

Description

- `strptime(text = VARCHAR, format = VARCHAR)`: Converts the `string` text to timestamp according to the format string. Throws an error on failure. To return `NULL` on failure, use try_strptime.
- `strptime(text = VARCHAR, format.list = `VARCHAR[]`)`: Converts the `string` text to timestamp applying the format strings in the list until one succeeds. Throws an error on failure. To return `NULL` on failure, use try_strptime.

Usage

```
strptime(text, format)
```

Arguments

`text`, `format` VARCHAR

Value

`TIMESTAMP`

Overloads

- `strptime(text = VARCHAR, format = VARCHAR)`
- `strptime(text = VARCHAR, format.list = `VARCHAR[]`)`

SQL examples

```
strptime('Wed, 1 January 1992 - 08:38:40 PM', '%a, %-d %B %Y - %I:%M:%S %p')
strptime('4/15/2023 10:56:00', ['%d/%m/%Y %H:%M:%S', '%m/%d/%Y %H:%M:%S'])
```

`struct_concat` *DuckDB function struct_concat*

Description

Merge the multiple STRUCTs into a single STRUCT.

Usage

```
struct_concat()
```

Value

`STRUCT`

SQL examples

```
struct_concat(struct_pack(i := 4), struct_pack(s := 'string'))
```

`struct_contains` *DuckDB function struct_contains*

Description

Check if an unnamed STRUCT contains the value.

Usage

```
struct_contains(struct, entry)
```

Arguments

`struct`	`STRUCT`
`entry`	`ANY`

Value

`BOOLEAN`

SQL examples

```
struct_contains(ROW(3, 3, 0), 3)
```

`struct_extract` *DuckDB function struct_extract*

Description

Extract the named entry from the STRUCT.

Usage

```
struct_extract(struct, entry)
```

Arguments

`struct`	`STRUCT`
`entry`	`VARCHAR | BIGINT`

Value

`ANY`

SQL examples

```
struct_extract({'i': 3, 'v2': 3, 'v3': 0}, 'i')
```

`struct_extract_at` *DuckDB function struct_extract_at*

Description

Extract the entry from the STRUCT by position (starts at 1!).

Usage

```
struct_extract_at(struct, entry)
```

Arguments

`struct`	`STRUCT`
`entry`	`BIGINT`

Value

`ANY`

SQL examples

```
struct_extract_at({'i': 3, 'v2': 3, 'v3': 0}, 2)
```

`struct_insert` *DuckDB function struct_insert*

Description

Adds field(s)/value(s) to an existing STRUCT with the argument values. The entry name(s) will be the bound variable name(s).

Usage

```
struct_insert()
```

Value

`STRUCT`

SQL examples

```
struct_insert({'a': 1}, b := 2)
```

`struct_keys` *DuckDB function struct_keys*

Description

Returns the field names of a STRUCT as a list.

Usage

```
struct_keys(struct)
```

Arguments

`struct`	`ANY`

Value

`VARCHAR[]`

SQL examples

```
struct_keys({'a': 1, 'b': 2})
```

`struct_pack` *DuckDB function struct_pack*

Description

Create a STRUCT containing the argument values. The entry name will be the bound variable name.

Usage

```
struct_pack()
```

Value

`STRUCT`

SQL examples

```
struct_pack(i := 4, s := 'string')
```

`struct_position` *DuckDB function struct_position*

Description

Get the position of the entry in an unnamed STRUCT, starting at 1.

Usage

```
struct_position(struct, entry)
```

Arguments

`struct`	`STRUCT`
`entry`	`ANY`

Value

`INTEGER`

SQL examples

```
struct_position(ROW(3, 3, 0), 3)
```

`struct_update` *DuckDB function struct_update*

Description

Changes field(s)/value(s) to an existing STRUCT with the argument values. The entry name(s) will be the bound variable name(s).

Usage

```
struct_update()
```

Value

`STRUCT`

SQL examples

```
struct_update({'a': 1}, a := 2)
```

`struct_values` *DuckDB function struct_values*

Description

Returns the field values of a STRUCT as an UnnamedStruct.

Usage

```
struct_values(struct)
```

Arguments

`struct`	`STRUCT`

Value

`STRUCT`

SQL examples

```
struct_values({'a': 1, 'b': 'world'})
```

`substring` *DuckDB function substring*

Description

Extracts substring starting from character `start` up to the end of the string. If optional argument `length` is set, extracts a substring of `length` characters instead. Note that a `start` value of `1` refers to the first character of the `string`.

Usage

```
substring(string, start, length)
```

Arguments

`string`	`VARCHAR`
`start`	`BIGINT`
`length`	`BIGINT`

Value

`VARCHAR`

Overloads

- `substring(string = VARCHAR, start = BIGINT, length = BIGINT)`
- `substring(string = VARCHAR, start = BIGINT)`

SQL examples

```
substring('Hello', 2)
substring('Hello', 2, 2)
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`substring_grapheme` *DuckDB function substring_grapheme*

Description

Extracts substring starting from grapheme clusters `start` up to the end of the string. If optional argument `length` is set, extracts a substring of `length` grapheme clusters instead. Note that a `start` value of `1` refers to the `first` character of the `string`.

Usage

```
substring_grapheme(string, start, length)
```

Arguments

`string`	`VARCHAR`
`start`	`BIGINT`
`length`	`BIGINT`

Value

`VARCHAR`

Overloads

- `substring_grapheme(string = VARCHAR, start = BIGINT, length = BIGINT)`
- `substring_grapheme(string = VARCHAR, start = BIGINT)`

SQL examples

```
substring_grapheme('    ', 3)
substring_grapheme('    ', 3, 2)
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`subtract` *DuckDB function subtract*

Description

DuckDB function `subtract()`.

Usage

```
subtract(col0, col1)
```

Arguments

`col0`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTI`
`col1`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIGNUM | DATE | TIMESTAMP | INTERVAL`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLI`

Overloads

- subtract(col0 = TINYINT)
- subtract(col0 = TINYINT, col1 = TINYINT)
- subtract(col0 = SMALLINT)
- subtract(col0 = SMALLINT, col1 = SMALLINT)
- subtract(col0 = INTEGER)
- subtract(col0 = INTEGER, col1 = INTEGER)
- subtract(col0 = BIGINT)
- subtract(col0 = BIGINT, col1 = BIGINT)
- subtract(col0 = HUGEINT)
- subtract(col0 = HUGEINT, col1 = HUGEINT)
- subtract(col0 = FLOAT)
- subtract(col0 = FLOAT, col1 = FLOAT)
- subtract(col0 = DOUBLE)
- subtract(col0 = DOUBLE, col1 = DOUBLE)
- subtract(col0 = DECIMAL)
- subtract(col0 = DECIMAL, col1 = DECIMAL)
- subtract(col0 = UTINYINT)
- subtract(col0 = UTINYINT, col1 = UTINYINT)
- subtract(col0 = USMALLINT)
- subtract(col0 = USMALLINT, col1 = USMALLINT)
- subtract(col0 = UINTEGER)
- subtract(col0 = UINTEGER, col1 = UINTEGER)
- subtract(col0 = UBIGINT)
- subtract(col0 = UBIGINT, col1 = UBIGINT)
- subtract(col0 = UHUGEINT)
- subtract(col0 = UHUGEINT, col1 = UHUGEINT)
- subtract(col0 = BIGNUM)
- subtract(col0 = BIGNUM, col1 = BIGNUM)
- subtract(col0 = DATE, col1 = DATE)
- subtract(col0 = DATE, col1 = INTEGER)
- subtract(col0 = TIMESTAMP, col1 = TIMESTAMP)
- subtract(col0 = INTERVAL, col1 = INTERVAL)
- subtract(col0 = DATE, col1 = INTERVAL)
- subtract(col0 = TIME, col1 = INTERVAL)
- subtract(col0 = TIMESTAMP, col1 = INTERVAL)
- subtract(col0 = `TIME WITH TIME ZONE`, col1 = INTERVAL)
- subtract(col0 = INTERVAL)

`suffix` *DuckDB function suffix*

Description

Returns `true` if `string` ends with `search_string`.

Usage

```
suffix(string, search_string)
```

Arguments

`string`, `search_string`	`VARCHAR`

Value

`BOOLEAN`

SQL examples

```
suffix('abc', 'bc')
```

See Also

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`sum` *DuckDB function sum*

Description

Calculates the sum value for all tuples in arg.

Usage

```
sum(arg)
```

Arguments

arg	DECIMAL | BOOLEAN | SMALLINT | INTEGER | BIGINT | HUGEINT | DOUBLE | BIGNUM

Value

DECIMAL | HUGEINT | DOUBLE | BIGNUM

SQL examples

```
sum(A)
```

---

sum_no_overflow *DuckDB function sum_no_overflow*

Description

Internal only. Calculates the sum value for all tuples in arg without overflow checks.

Usage

```
sum_no_overflow(arg)
```

Arguments

arg	INTEGER | BIGINT | DECIMAL

Value

HUGEINT | DECIMAL

SQL examples

```
sum_no_overflow(A)
```

`summary` *DuckDB function summary*

Description

DuckDB function `summary()`.

Usage

```
summary(col0)
```

Arguments

`col0`	`TABLE`

Value

Unspecified.

`switch` *DuckDB function switch*

Description

Creates a switch statement similar to CASE WHEN/THEN.

Usage

```
switch(key, map)
```

Arguments

`key`	`K | MAP(K, V)`
`map`	`MAP(K, V) | V`

Value

`V`

Overloads

- `switch(key = K, map = \`MAP(K, V)\`)`
- `switch(key = K, map = \`MAP(K, V)\`, value = V)`
- `switch(key = \`MAP(K, V)\`, map = V)`
- `switch(key = \`MAP(K, V)\`)`

SQL examples

```
switch(x, map({1 : 1}, default)
```

`table_info` *DuckDB function table_info*

Description

DuckDB function `table_info()`.

Usage

```
table_info(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

`tan` *DuckDB function tan*

Description

Computes the tan of x.

Usage

```
tan(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
tan(90)
```

`tanh` *DuckDB function tanh*

Description

Computes the hyperbolic tan of x.

Usage

```
tanh(x)
```

Arguments

`x`	`DOUBLE`

Value

`DOUBLE`

SQL examples

```
tanh(1)
```

`test_all_types` *DuckDB function test_all_types*

Description

DuckDB function `test_all_types()`.

Usage

```
test_all_types(use_large_bignum, use_large_enum)
```

Arguments

`use_large_bignum`, `use_large_enum`	`BOOLEAN`

Value

Unspecified.

`test_vector_types` *DuckDB function test_vector_types*

Description

DuckDB function `test_vector_types()`.

Usage

```
test_vector_types(col0, all_flat)
```

Arguments

`col0`	`ANY`
`all_flat`	`BOOLEAN`

Value

Unspecified.

`text` *DuckDB function text*

Description

DuckDB function `text()`.

Usage

```
text(col0, col1)
```

Arguments

`col0`	`DOUBLE`
`col1`	`VARCHAR`

Value

`VARCHAR`

Provided by

The `excel` extension (`LOAD excel;`).

`time_bucket` *DuckDB function time_bucket*

Description

Truncate TIMESTAMPTZ by the specified interval bucket_width. Buckets are aligned relative to origin TIMESTAMPTZ. The origin defaults to 2000-01-03 00:00:00+00 for buckets that do not include a month or year interval, and to 2000-01-01 00:00:00+00 for month and year buckets.

Usage

```
time_bucket(bucket_width, timestamp, origin)
```

Arguments

bucket_width	INTERVAL
timestamp	DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE
origin	DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE | VARCHAR

Value

```
DATE | TIMESTAMP | TIMESTAMP WITH TIME ZONE
```

Overloads

- `time_bucket(bucket_width = INTERVAL, timestamp = DATE)`
- `time_bucket(bucket_width = INTERVAL, timestamp = DATE, origin = DATE)`
- `time_bucket(bucket_width = INTERVAL, timestamp = DATE, origin = INTERVAL)`
- `time_bucket(bucket_width = INTERVAL, timestamp = TIMESTAMP)`
- `time_bucket(bucket_width = INTERVAL, timestamp = TIMESTAMP, origin = INTERVAL)`
- `time_bucket(bucket_width = INTERVAL, timestamp = TIMESTAMP, origin = TIMESTAMP)`
- `time_bucket(bucket_width = INTERVAL, timestamp = `TIMESTAMP WITH TIME ZONE`)`
- `time_bucket(bucket_width = INTERVAL, timestamp = `TIMESTAMP WITH TIME ZONE`, origin = INTERVAL)`
- `time_bucket(bucket_width = INTERVAL, timestamp = `TIMESTAMP WITH TIME ZONE`, origin = `TIMESTAMP WITH TIME ZONE`)`
- `time_bucket(bucket_width = INTERVAL, timestamp = `TIMESTAMP WITH TIME ZONE`, origin = VARCHAR)`

SQL examples

```
time_bucket(INTERVAL '2 weeks', TIMESTAMP '1992-04-20 15:26:00-07', TIMESTAMP '1992-04-01 00:0
```

`timetz_byte_comparable`
DuckDB function timetz_byte_comparable

Description

Converts a TIME WITH TIME ZONE to an integer sort key.

Usage

```
timetz_byte_comparable(time_tz)
```

Arguments

`time_tz`	`TIME WITH TIME ZONE`

Value

`UBIGINT`

SQL examples

```
timetz_byte_comparable('18:18:16.21-07:00'::TIMETZ)
```

`timezone` *DuckDB function timezone*

Description

Extract the timezone component from a date or timestamp.

Usage

```
timezone(ts, col1)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE | VARCHAR`
`col1`	`TIME WITH TIME ZONE | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT | TIME WITH TIME ZONE | TIMESTAMP WITH TIME ZONE | TIMESTAMP`

Overloads

- `timezone(ts = DATE)`
- `timezone(ts = INTERVAL)`
- `timezone(ts = INTERVAL, col1 = `TIME WITH TIME ZONE`)`
- `timezone(ts = TIMESTAMP)`
- `timezone(ts = `TIMESTAMP WITH TIME ZONE`)`
- `timezone(ts = VARCHAR, col1 = TIMESTAMP)`
- `timezone(ts = VARCHAR, col1 = `TIMESTAMP WITH TIME ZONE`)`
- `timezone(ts = VARCHAR, col1 = `TIME WITH TIME ZONE`)`

SQL examples

```
timezone(timestamp '2021-08-03 11:59:44.123456')
```

`timezone_hour` *DuckDB function timezone_hour*

Description

Extract the timezone_hour component from a date or timestamp.

Usage

```
timezone_hour(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
timezone_hour(timestamp '2021-08-03 11:59:44.123456')
```

`timezone_minute` *DuckDB function timezone_minute*

Description

Extract the timezone_minute component from a date or timestamp.

Usage

```
timezone_minute(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
timezone_minute(timestamp '2021-08-03 11:59:44.123456')
```

`to_base` *DuckDB function to_base*

Description

Converts `number` to a string in the given base `radix`, optionally padding with leading zeros to `min_length`.

Usage

```
to_base(number, radix, min_length)
```

Arguments

`number`	`BIGINT`
`radix`	`INTEGER`
`min_length`	`INTEGER`

Value

`VARCHAR`

Overloads

- `to_base(number = BIGINT, radix = INTEGER)`
- `to_base(number = BIGINT, radix = INTEGER, min_length = INTEGER)`

SQL examples

```
to_base(42, 16, 5)
```

See Also

Other numeric: `bin()`, `bit_length()`, `formatReadableDecimalSize()`, `format_bytes()`, `greatest()`, `hex()`, `least()`, `len()`, `parse_formatted_bytes()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`to_base64` *DuckDB function to_base64*

Description

Converts a `blob` to a base64 encoded string.

Usage

```
to_base64(blob)
```

Arguments

`blob`	`BLOB`

Value

`VARCHAR`

SQL examples

```
to_base64('A'::BLOB)
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `unbin()`, `unhex()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `translate()`, `trim()`, `unbin()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`to_centuries` *DuckDB function to_centuries*

Description

Construct a century interval.

Usage

```
to_centuries(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_centuries(5)
```

`to_days` *DuckDB function to_days*

Description

Construct a day interval.

Usage

```
to_days(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_days(5)
```

`to_decades` *DuckDB function to_decades*

Description

Construct a decade interval.

Usage

```
to_decades(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_decades(5)
```

`to_hours` *DuckDB function to_hours*

Description

Construct a hour interval.

Usage

```
to_hours(integer)
```

Arguments

`integer`	`BIGINT`

Value

`INTERVAL`

SQL examples

```
to_hours(5)
```

`to_json` *DuckDB function to_json*

Description

DuckDB function `to_json()`.

Usage

```
to_json()
```

Value

`JSON`

Provided by

The `json` extension (`LOAD json;`).

`to_microseconds` *DuckDB function to_microseconds*

Description

Construct a microsecond interval.

Usage

```
to_microseconds(integer)
```

Arguments

`integer`	`BIGINT`

Value

`INTERVAL`

SQL examples

```
to_microseconds(5)
```

`to_millennia` *DuckDB function to_millennia*

Description

Construct a millenium interval.

Usage

```
to_millennia(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_millennia(1)
```

`to_milliseconds` *DuckDB function to_milliseconds*

Description

Construct a millisecond interval.

Usage

```
to_milliseconds(double)
```

Arguments

`double`	`DOUBLE`

Value

`INTERVAL`

SQL examples

```
to_milliseconds(5.5)
```

`to_minutes` *DuckDB function to_minutes*

Description

Construct a minute interval.

Usage

```
to_minutes(integer)
```

Arguments

`integer`	`BIGINT`

Value

`INTERVAL`

SQL examples

```
to_minutes(5)
```

`to_months` *DuckDB function to_months*

Description

Construct a month interval.

Usage

```
to_months(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_months(5)
```

`to_quarters` *DuckDB function to_quarters*

Description

Construct a quarter interval.

Usage

```
to_quarters(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_quarters(5)
```

`to_seconds` *DuckDB function to_seconds*

Description

Construct a second interval.

Usage

```
to_seconds(double)
```

Arguments

`double`	`DOUBLE`

Value

`INTERVAL`

SQL examples

```
to_seconds(5.5)
```

`to_timestamp` *DuckDB function to_timestamp*

Description

Converts secs since epoch to a timestamp with time zone.

Usage

```
to_timestamp(sec)
```

Arguments

`sec`	`DOUBLE`

Value

`TIMESTAMP WITH TIME ZONE`

SQL examples

```
to_timestamp(1284352323.5)
```

`to_weeks` *DuckDB function to_weeks*

Description

Construct a week interval.

Usage

```
to_weeks(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_weeks(5)
```

`to_years` *DuckDB function to_years*

Description

Construct a year interval.

Usage

```
to_years(integer)
```

Arguments

`integer`	`INTEGER | BIGINT`

Value

`INTERVAL`

SQL examples

```
to_years(5)
```

`today` *DuckDB function today*

Description

DuckDB function `today()`.

Usage

```
today()
```

Value

`DATE`

Provided by

The `icu` extension (`LOAD icu;`).

`tpcds` *DuckDB function tpcds*

Description

DuckDB function `tpcds()`.

Usage

```
tpcds(col0)
```

Arguments

`col0`	`BIGINT`

Value

Unspecified.

Provided by

The `tpcds` extension (`LOAD tpcds;`).

`tpcds_answers` *DuckDB function tpcds_answers*

Description

DuckDB function `tpcds_answers()`.

Usage

```
tpcds_answers()
```

Value

Unspecified.

Provided by

The `tpcds` extension (`LOAD tpcds;`).

`tpcds_queries` *DuckDB function tpcds_queries*

Description

DuckDB function `tpcds_queries()`.

Usage

```
tpcds_queries()
```

Value

Unspecified.

Provided by

The `tpcds` extension (`LOAD tpcds;`).

`tpch` *DuckDB function tpch*

Description

DuckDB function `tpch()`.

Usage

```
tpch(col0)
```

Arguments

`col0`	`BIGINT`

Value

Unspecified.

Provided by

The `tpch` extension (`LOAD tpch;`).

`tpch_answers` *DuckDB function tpch_answers*

Description

DuckDB function `tpch_answers()`.

Usage

```
tpch_answers()
```

Value

Unspecified.

Provided by

The `tpch` extension (`LOAD tpch;`).

`tpch_queries` *DuckDB function tpch_queries*

Description

DuckDB function `tpch_queries()`.

Usage

```
tpch_queries()
```

Value

Unspecified.

Provided by

The `tpch` extension (`LOAD tpch;`).

`translate` *DuckDB function translate*

Description

Replaces each character in `string` that matches a character in the `from` set with the corresponding character in the `to` set. If `from` is longer than `to`, occurrences of the extra characters in `from` are deleted.

Usage

```
translate(string, from, to)
```

Arguments

`string`, `from`, `to`
VARCHAR

Value

VARCHAR

SQL examples

```
translate('12345', '143', 'ax')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), trim(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`trim` *DuckDB function trim*

Description

Removes any occurrences of any of the `characters` from either side of the `string`. `characters` defaults to `space`.

Usage

```
trim(string, characters)
```

Arguments

`string, characters`	`VARCHAR`

Value

`VARCHAR`

Overloads

- `trim(string = VARCHAR)`
- `trim(string = VARCHAR, characters = VARCHAR)`

SQL examples

```
trim('    test  ')
trim('>>>>test<<', '><')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), unbin(), unhex(), unicode(), upper(), url_decode(), url_encode()

`trunc` *DuckDB function trunc*

Description

Truncates the number.

Usage

```
trunc(x, col1)
```

Arguments

x	TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT
col1	INTEGER

Value

TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | FLOAT | DOUBLE | DECIMAL | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT

Overloads

- trunc(x = TINYINT)
- trunc(x = TINYINT, col1 = INTEGER)
- trunc(x = SMALLINT)
- trunc(x = SMALLINT, col1 = INTEGER)
- trunc(x = INTEGER)
- trunc(x = INTEGER, col1 = INTEGER)
- trunc(x = BIGINT)
- trunc(x = BIGINT, col1 = INTEGER)
- trunc(x = HUGEINT)

- `trunc(x = HUGEINT, col1 = INTEGER)`
- `trunc(x = FLOAT)`
- `trunc(x = FLOAT, col1 = INTEGER)`
- `trunc(x = DOUBLE)`
- `trunc(x = DOUBLE, col1 = INTEGER)`
- `trunc(x = DECIMAL)`
- `trunc(x = DECIMAL, col1 = INTEGER)`
- `trunc(x = UTINYINT)`
- `trunc(x = UTINYINT, col1 = INTEGER)`
- `trunc(x = USMALLINT)`
- `trunc(x = USMALLINT, col1 = INTEGER)`
- `trunc(x = UINTEGER)`
- `trunc(x = UINTEGER, col1 = INTEGER)`
- `trunc(x = UBIGINT)`
- `trunc(x = UBIGINT, col1 = INTEGER)`
- `trunc(x = UHUGEINT)`
- `trunc(x = UHUGEINT, col1 = INTEGER)`

SQL examples

```
trunc(17.4)
```

`truncate_duckdb_logs` *DuckDB function truncate_duckdb_logs*

Description

DuckDB function `truncate_duckdb_logs()`.

Usage

```
truncate_duckdb_logs()
```

Value

Unspecified.

`try_strptime` *DuckDB function try_strptime*

Description

Converts the `string` text to timestamp according to the format string. Returns `NULL` on failure.

Usage

```
try_strptime(text, format)
```

Arguments

`text`	`VARCHAR`
`format`	`VARCHAR | VARCHAR[]`

Value

```
TIMESTAMP
```

SQL examples

```
try_strptime('Wed, 1 January 1992 - 08:38:40 PM', '%a, %-d %B %Y - %I:%M:%S %p')
```

`txid_current` *DuckDB function txid_current*

Description

Returns the current transaction's ID (a BIGINT). It will assign a new one if the current transaction does not have one already.

Usage

```
txid_current()
```

Value

```
UBIGINT
```

SQL examples

```
txid_current()
```

`typeof` *DuckDB function typeof*

Description

Returns the name of the data type of the result of the expression.

Usage

```
typeof(expression)
```

Arguments

`expression`	`ANY`

Value

`VARCHAR`

SQL examples

```
typeof('abc')
```

`ui_is_started` *DuckDB function ui_is_started*

Description

DuckDB function `ui_is_started()`.

Usage

```
ui_is_started()
```

Value

Unspecified.

Provided by

The `ui` extension (`LOAD ui;`).

`unbin` *DuckDB function unbin*

Description

Converts a `value` from binary representation to a blob.

Usage

```
unbin(value)
```

Arguments

`value`	`VARCHAR`

Value

`BLOB`

SQL examples

```
unbin('0110')
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unhex()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unhex()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

`unhex` *DuckDB function unhex*

Description

Converts a `value` from hexadecimal representation to a blob.

Usage

```
unhex(value)
```

Arguments

`value`	`VARCHAR`

Value

`BLOB`

SQL examples

```
unhex('2A')
```

See Also

Other blob: `decode()`, `encode()`, `from_base64()`, `hex()`, `md5()`, `md5_number()`, `octet_length()`, `or--or`, `repeat()`, `sha1()`, `sha256()`, `to_base64()`, `unbin()`

Other string: `array_extract()`, `ascii()`, `bar()`, `bin()`, `bit_length()`, `chr()`, `concat()`, `concat_ws()`, `contains()`, `format()`, `formatReadableDecimalSize()`, `format_bytes()`, `from_base64()`, `greatest()`, `hash()`, `hex()`, `ilike_escape()`, `instr()`, `least()`, `left()`, `left_grapheme()`, `len()`, `length_grapheme()`, `like_escape()`, `list_slice()`, `lower()`, `lpad()`, `ltrim()`, `md5()`, `md5_number()`, `nfc_normalize()`, `not_ilike_escape()`, `not_like_escape()`, `or--or`, `parse_dirname()`, `parse_dirpath()`, `parse_filename()`, `parse_formatted_bytes()`, `parse_path()`, `prefix()`, `printf()`, `repeat()`, `replace()`, `reverse()`, `right()`, `right_grapheme()`, `rpad()`, `rtrim()`, `sha1()`, `sha256()`, `starts_with()`, `string_split()`, `strip_accents()`, `strlen()`, `substring()`, `substring_grapheme()`, `suffix()`, `to_base()`, `to_base64()`, `translate()`, `trim()`, `unbin()`, `unicode()`, `upper()`, `url_decode()`, `url_encode()`

unicode *DuckDB function unicode*

Description

Returns an INTEGER representing the unicode codepoint of the first character in the string.

Usage

```
unicode(string)
```

Arguments

string	VARCHAR

Value

INTEGER

SQL examples

```
[unicode('âbcd'), unicode('â'), unicode(''), unicode(NULL)]
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), upper(), url_decode(), url_encode()

union_extract *DuckDB function union_extract*

Description

Extract the value with the named tags from the union. NULL if the tag is not currently selected.

Usage

```
union_extract(union, tag)
```

Arguments

`union`	`UNION`
`tag`	`VARCHAR`

Value

`ANY`

SQL examples

```
union_extract(s, 'k')
```

`union_tag` *DuckDB function union_tag*

Description

Retrieve the currently selected tag of the union as an ENUM.

Usage

```
union_tag(union)
```

Arguments

`union`	`UNION`

Value

`ANY`

SQL examples

```
union_tag(union_value(k := 'foo'))
```

`union_value` *DuckDB function union_value*

Description

Create a single member UNION containing the argument value. The tag of the value will be the bound variable name.

Usage

```
union_value()
```

Value

`UNION`

SQL examples

```
union_value(k := 'hello')
```

`unity_catalog_checkpoint_table`
DuckDB function unity_catalog_checkpoint_table

Description

DuckDB function `unity_catalog_checkpoint_table()`.

Usage

```
unity_catalog_checkpoint_table(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `unity_catalog` extension (`LOAD unity_catalog;`).

`unity_catalog_force_checkpoint_table`
DuckDB function unity_catalog_force_checkpoint_table

Description

DuckDB function `unity_catalog_force_checkpoint_table()`.

Usage

```
unity_catalog_force_checkpoint_table(col0)
```

Arguments

`col0`	`VARCHAR`

Value

Unspecified.

Provided by

The `unity_catalog` extension (`LOAD unity_catalog;`).

`unnest` *DuckDB function unnest*

Description

DuckDB function `unnest()`.

Usage

```
unnest(col0)
```

Arguments

`col0`	`ANY`

Value

Unspecified.

`unpivot_list` *DuckDB function unpivot_list*

Description

Identical to list_value, but generated as part of unpivot for better error messages.

Usage

```
unpivot_list()
```

Value

`LIST`

SQL examples

```
unpivot_list(4, 5, 6)
```

See Also

Other list: `array_extract()`, `array_length()`, `concat()`, `contains()`, `flatten()`, `generate_series()`, `len()`, `list_aggregate()`, `list_concat()`, `list_contains()`, `list_cosine_distance()`, `list_cosine_similarity()`, `list_distance()`, `list_distinct()`, `list_extract()`, `list_filter()`, `list_grade_up()`, `list_has_all()`, `list_has_any()`, `list_inner_product()`, `list_intersect()`, `list_negative_inner_product()`, `list_position()`, `list_reduce()`, `list_resize()`, `list_reverse_sort()`, `list_select()`, `list_slice()`, `list_sort()`, `list_transform()`, `list_unique()`, `list_value()`, `list_where()`, `list_zip()`, `or--or`, `range()`, `repeat()`

`upper` *DuckDB function upper*

Description

Converts `string` to upper case.

Usage

```
upper(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
upper('Hello')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), url_decode(), url_encode()

---

url_decode *DuckDB function url_decode*

Description

Decodes a URL from a representation using Percent-Encoding.

Usage

```
url_decode(string)
```

Arguments

string	VARCHAR

Value

VARCHAR

SQL examples

```
url_decode('https%3A%2F%2Fduckdb.org%2Fwhy_duckdb%23portable')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(),

rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_encode()

`url_encode` *DuckDB function url_encode*

Description

Encodes a URL to a representation using Percent-Encoding.

Usage

```
url_encode(string)
```

Arguments

`string`	`VARCHAR`

Value

`VARCHAR`

SQL examples

```
url_encode('this string has/ special+ characters>')
```

See Also

Other string: array_extract(), ascii(), bar(), bin(), bit_length(), chr(), concat(), concat_ws(), contains(), format(), formatReadableDecimalSize(), format_bytes(), from_base64(), greatest(), hash(), hex(), ilike_escape(), instr(), least(), left(), left_grapheme(), len(), length_grapheme(), like_escape(), list_slice(), lower(), lpad(), ltrim(), md5(), md5_number(), nfc_normalize(), not_ilike_escape(), not_like_escape(), or--or, parse_dirname(), parse_dirpath(), parse_filename(), parse_formatted_bytes(), parse_path(), prefix(), printf(), repeat(), replace(), reverse(), right(), right_grapheme(), rpad(), rtrim(), sha1(), sha256(), starts_with(), string_split(), strip_accents(), strlen(), substring(), substring_grapheme(), suffix(), to_base(), to_base64(), translate(), trim(), unbin(), unhex(), unicode(), upper(), url_decode()

`user` *DuckDB function user*

Description

DuckDB macro `user()`.

Usage

```
user()
```

Value

Unspecified.

`user_agent` *DuckDB function user_agent*

Description

DuckDB function `user_agent()`.

Usage

```
user_agent()
```

Value

Unspecified.

`uuid` *DuckDB function uuid*

Description

Returns a random UUID v4 similar to this: eeccb8c5-9943-b2bb-bb5e-222f4e14b687.

Usage

```
uuid()
```

Value

`UUID`

SQL examples

```
uuid()
```

`uuid_extract_timestamp`
DuckDB function uuid_extract_timestamp

Description

Extract the timestamp for the given UUID v7.

Usage

```
uuid_extract_timestamp(uuid)
```

Arguments

```
uuid           UUID
```

Value

```
TIMESTAMP WITH TIME ZONE
```

SQL examples

```
uuid_extract_timestamp('019482e4-1441-7aad-8127-eec99573b0a0')
```

`uuid_extract_version` *DuckDB function uuid_extract_version*

Description

Extract a version for the given UUID.

Usage

```
uuid_extract_version(uuid)
```

Arguments

```
uuid           UUID
```

Value

```
UINTEGER
```

SQL examples

```
uuid_extract_version('019482e4-1441-7aad-8127-eec99573b0a0')
```

`uuidv4` *DuckDB function uuidv4*

Description

Returns a random UUIDv4 similar to this: eeccb8c5-9943-b2bb-bb5e-222f4e14b687.

Usage

```
uuidv4()
```

Value

`UUID`

SQL examples

```
uuidv4()
```

`uuidv7` *DuckDB function uuidv7*

Description

Returns a random UUID v7 similar to this: 019482e4-1441-7aad-8127-eec99573b0a0.

Usage

```
uuidv7()
```

Value

`UUID`

SQL examples

```
uuidv7()
```

`var_pop` *DuckDB function var_pop*

Description

Returns the population variance.

Usage

```
var_pop(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

`var_samp` *DuckDB function var_samp*

Description

Returns the sample variance of all input values.

Usage

```
var_samp(x)
```

Arguments

x	DOUBLE

Value

DOUBLE

SQL examples

```
(SUM(x^2) - SUM(x)^2 / COUNT(x)) / (COUNT(x) - 1)
```

`variant_bytes_to_variant`
DuckDB function variant_bytes_to_variant

Description

DuckDB function `variant_bytes_to_variant()`.

Usage

```
variant_bytes_to_variant(col0)
```

Arguments

`col0`	`BLOB`

Value

`VARIANT`

`variant_extract` *DuckDB function variant_extract*

Description

- `variant_extract(input_variant = VARIANT, field = VARCHAR)`: Returns the `field` from the `input_variant` if it's an OBJECT.
- `variant_extract(input_variant = VARIANT, index = UINTEGER)`: Returns the entry at `index` from the `input_variant` if it's an ARRAY.

Usage

```
variant_extract(input_variant, field)
```

Arguments

`input_variant`	`VARIANT`
`field`	`VARCHAR`

Value

`VARIANT`

Overloads

- `variant_extract(input_variant = VARIANT, field = VARCHAR)`
- `variant_extract(input_variant = VARIANT, index = UINTEGER)`

SQL examples

```
variant_extract({'a': 42, 'b': [1,2,3])::VARIANT, 'b')
variant_extract([1,2,3])::VARIANT, 0)
```

See Also

Other variant: `variant_normalize()`, `variant_typeof()`

`variant_normalize` *DuckDB function variant_normalize*

Description

Normalizes the `input_variant` to a canonical representation.

Usage

```
variant_normalize(input_variant)
```

Arguments

```
input_variant  VARIANT
```

Value

```
VARIANT
```

SQL examples

```
variant_normalize({'b': [1,2,3], 'a': 42})::VARIANT)
```

See Also

Other variant: `variant_extract()`, `variant_typeof()`

variant_to_parquet_variant

DuckDB function variant_to_parquet_variant

Description

DuckDB function `variant_to_parquet_variant()`.

Usage

```
variant_to_parquet_variant(col0)
```

Arguments

col0	VARIANT

Value

ANY

variant_typeof

DuckDB function variant_typeof

Description

Returns the internal type of the `input_variant`.

Usage

```
variant_typeof(input_variant)
```

Arguments

input_variant	VARIANT

Value

VARCHAR

SQL examples

```
variant_typeof({'a': 42, 'b': [1,2,3]}::VARIANT)
```

See Also

Other variant: variant_extract(), variant_normalize()

vector_type *DuckDB function vector_type*

Description

Returns the VectorType of a given column.

Usage

```
vector_type(col)
```

Arguments

col	ANY

Value

VARCHAR

SQL examples

```
vector_type(col)
```

verify_external *DuckDB function verify_external*

Description

DuckDB function `verify_external()`.

Usage

```
verify_external()
```

Value

Unspecified.

verify_fetch_row *DuckDB function verify_fetch_row*

Description

DuckDB function `verify_fetch_row()`.

Usage

```
verify_fetch_row()
```

Value

Unspecified.

verify_parallelism *DuckDB function verify_parallelism*

Description

DuckDB function `verify_parallelism()`.

Usage

```
verify_parallelism()
```

Value

Unspecified.

verify_serializer *DuckDB function verify_serializer*

Description

DuckDB function `verify_serializer()`.

Usage

```
verify_serializer()
```

Value

Unspecified.

`version` *DuckDB function version*

Description

Returns the currently active version of DuckDB in this format: v0.3.2 .

Usage

```
version()
```

Value

`VARCHAR`

SQL examples

```
version()
```

`vortex_scan` *DuckDB function vortex_scan*

Description

DuckDB function `vortex_scan()`.

Usage

```
vortex_scan(col0)
```

Arguments

`col0`	`VARCHAR | VARCHAR[]`

Value

Unspecified.

Provided by

The `vortex` extension (`LOAD vortex;`).

`vss_join` *DuckDB function vss_join*

Description

DuckDB macro `vss_join()`.

Usage

```
vss_join(left_table, right_table, left_col, right_col, k, metric)
```

Arguments

`left_table`, `right_table`, `left_col`, `right_col`, `k`, `metric`
: Unspecified.

Value

Unspecified.

Provided by

The `vss` extension (`LOAD vss;`).

`vss_match` *DuckDB function vss_match*

Description

DuckDB macro `vss_match()`.

Usage

```
vss_match(right_table, left_col, right_col, k, metric)
```

Arguments

`right_table`, `left_col`, `right_col`, `k`, `metric`
: Unspecified.

Value

Unspecified.

Provided by

The `vss` extension (`LOAD vss;`).

wavg — *DuckDB function wavg*

Description

DuckDB macro `wavg()`.

Usage

```
wavg(value, weight)
```

Arguments

`value`, `weight`	Unspecified.

Value

Unspecified.

week — *DuckDB function week*

Description

Extract the week component from a date or timestamp.

Usage

```
week(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
week(timestamp '2021-08-03 11:59:44.123456')
```

`weekday` *DuckDB function weekday*

Description

Extract the weekday component from a date or timestamp.

Usage

```
weekday(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
weekday(timestamp '2021-08-03 11:59:44.123456')
```

`weekofyear` *DuckDB function weekofyear*

Description

Extract the weekofyear component from a date or timestamp.

Usage

```
weekofyear(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
weekofyear(timestamp '2021-08-03 11:59:44.123456')
```

`weighted_avg` *DuckDB function weighted_avg*

Description

DuckDB macro `weighted_avg()`.

Usage

```
weighted_avg(value, weight)
```

Arguments

`value`, `weight`	Unspecified.

Value

Unspecified.

`which_secret` *DuckDB function which_secret*

Description

DuckDB function `which_secret()`.

Usage

```
which_secret(col0, col1)
```

Arguments

`col0`, `col1`	`VARCHAR`

Value

Unspecified.

`whoami` *DuckDB function whoami*

Description

DuckDB macro `whoami()`.

Usage

```
whoami()
```

Value

Unspecified.

Provided by

The `quack` extension (`LOAD quack;`).

`write_blob` *DuckDB function write_blob*

Description

DuckDB function `write_blob()`.

Usage

```
write_blob(col0, col1)
```

Arguments

`col0`	`VARCHAR`
`col1`	`BLOB`

Value

`BOOLEAN`

Provided by

The `delta` extension (`LOAD delta;`).

`write_log` *DuckDB function write_log*

Description

Writes to the logger.

Usage

```
write_log(string)
```

Arguments

`string`	`VARCHAR`

Value

`ANY`

SQL examples

```
write_log('Hello')
```

`xor` *DuckDB function xor*

Description

Bitwise XOR.

Usage

```
xor(left, right)
```

Arguments

`left, right`	`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

Value

`TINYINT | SMALLINT | INTEGER | BIGINT | HUGEINT | UTINYINT | USMALLINT | UINTEGER | UBIGINT | UHUGEINT | BIT`

SQL examples

```
xor(17, 5)
```

`year` *DuckDB function year*

Description

Extract the year component from a date or timestamp.

Usage

```
year(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
year(timestamp '2021-08-03 11:59:44.123456')
```

`yearweek` *DuckDB function yearweek*

Description

Extract the yearweek component from a date or timestamp.

Usage

```
yearweek(ts)
```

Arguments

`ts`	`DATE | INTERVAL | TIMESTAMP | TIMESTAMP WITH TIME ZONE`

Value

`BIGINT`

SQL examples

```
yearweek(timestamp '2021-08-03 11:59:44.123456')
```

Index

!__postfix *(factorial)*, 160
!~~ *(not-~~)*, 381
!~~* *(not-~~*)*, 381
* **aggregate**
  greatest, 180
  least, 305
* **array**
  array_cosine_distance, 51
  array_cosine_similarity, 52
  array_cross_product, 53
  array_distance, 53
  array_inner_product, 55
  array_negative_inner_product, 57
  array_value, 62
* **bitstring**
  octet_length, 386
* **blob**
  decode, 108
  encode, 151
  from_base64, 170
  hex, 189
  md5, 365
  md5_number, 366
  octet_length, 386
  or--or, 387
  repeat, 469
  sha1, 486
  sha256, 487
  to_base64, 640
  unbin, 658
  unhex, 659
* **date**
  greatest, 180
  least, 305
* **geometry**
  st_astext, 507
  st_aswkb, 509
  st_crs, 524
  st_geomfromwkb, 546
* **lambda**
  list_filter, 323
  list_reduce, 335
  list_transform, 344
* **list**
  array_extract, 54
  array_length, 56
  concat, 82
  contains, 84
  flatten, 164
  generate_series, 173
  len, 308
  list_aggregate, 312
  list_concat, 317
  list_contains, 317
  list_cosine_distance, 318
  list_cosine_similarity, 319
  list_distance, 320
  list_distinct, 321
  list_extract, 322
  list_filter, 323
  list_grade_up, 324
  list_has_all, 325
  list_has_any, 326
  list_inner_product, 327
  list_intersect, 328
  list_negative_inner_product, 332
  list_position, 333
  list_reduce, 335
  list_resize, 336
  list_reverse_sort, 337
  list_select, 338
  list_slice, 340
  list_sort, 341
  list_transform, 344
  list_unique, 344
  list_value, 345
  list_where, 347
  list_zip, 347

or--or, 387
range, 434
repeat, 469
unpivot_list, 664
* **map**
contains, 84
* **numeric**
bin, 68
bit_length, 70
format_bytes, 167
formatReadableDecimalSize, 169
greatest, 180
hex, 189
least, 305
len, 308
parse_formatted_bytes, 396
to_base, 639
* **regex**
regexp_escape, 458
regexp_extract, 458
regexp_extract_all, 460
regexp_full_match, 461
regexp_matches, 461
regexp_replace, 462
string_split_regex, 618
* **string**
array_extract, 54
ascii, 64
bar, 67
bin, 68
bit_length, 70
chr, 80
concat, 82
concat_ws, 83
contains, 84
format, 166
format_bytes, 167
formatReadableDecimalSize, 169
from_base64, 170
greatest, 180
hash, 189
hex, 189
ilike_escape, 272
instr, 275
least, 305
left, 306
left_grapheme, 307
len, 308
length_grapheme, 309
like_escape, 311
list_slice, 340
lower, 351
lpad, 352
ltrim, 353
md5, 365
md5_number, 366
nfc_normalize, 380
not_ilike_escape, 382
not_like_escape, 383
or--or, 387
parse_dirname, 392
parse_dirpath, 393
parse_filename, 395
parse_formatted_bytes, 396
parse_path, 397
prefix, 422
printf, 423
repeat, 469
replace, 471
reverse, 474
right, 474
right_grapheme, 475
rpad, 479
rtrim, 481
sha1, 486
sha256, 487
starts_with, 612
string_split, 617
strip_accents, 619
strlen, 620
substring, 626
substring_grapheme, 627
suffix, 630
to_base, 639
to_base64, 640
translate, 652
trim, 653
unbin, 658
unhex, 659
unicode, 660
upper, 664
url_decode, 665
url_encode, 666
* **struct**
array_extract, 54
* **text_similarity**

damerau_levenshtein, 100
hamming, 182
jaccard, 279
jaro_similarity, 280
jaro_winkler_similarity, 281
levenshtein, 310
* **timestamp**
greatest, 180
least, 305
* **variant**
variant_extract, 671
variant_normalize, 672
variant_typeof, 673
* *(multiply)*, 374
** *(pow)*, 417
+ *(add)*, 31
- *(subtract)*, 628
->> *(arrow->>)*, 62
/, 25
// *(divide)*, 120
<-> *(list_distance)*, 320
<=> *(list_cosine_distance)*, 318
<<, 28
<<=, 28
>>, 29
>>=, 29
% *(mod)*, 372
&, 25
&& *(list_has_any)*, 326
^ *(pow)*, 417
~, 27
~~, 27
~~*, 26
~~~, 26

abs, 30
acos, 30
acosh, 31
add, 31
add_parquet_key, 33
age, 33
aggregate *(list_aggregate)*, 312
ago, 34
alias, 34
all_profiling_output, 35
any_value, 35
apply *(list_transform)*, 344
approx_count_distinct, 36
approx_quantile, 36
approx_top_k, 37
arbitrary *(first)*, 163
arg_max, 37
arg_max_null, 40
arg_max_nulls_last, 41
arg_min, 44
arg_min_null, 47
arg_min_nulls_last, 47
argmax *(arg_max)*, 37
argmin *(arg_min)*, 44
array_agg, 50
array_aggr *(list_aggregate)*, 312
array_aggregate *(list_aggregate)*, 312
array_append, 51
array_apply *(list_transform)*, 344
array_cat *(list_concat)*, 317
array_concat *(list_concat)*, 317
array_contains *(list_contains)*, 317
array_cosine_distance, 51
array_cosine_distance(), *52–54*, *56*, *57*, *62*
array_cosine_similarity, 52
array_cosine_similarity(), *52–54*, *56*, *57*, *62*
array_cross_product, 53
array_cross_product(), *52*, *54*, *56*, *57*, *62*
array_distance, 53
array_distance(), *52*, *53*, *56*, *57*, *62*
array_distinct *(list_distinct)*, 321
array_dot_product *(array_inner_product)*, 55
array_extract, 54
array_extract(), *56*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *165*, *167*, *169*, *170*, *174*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340–342*, *344–348*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *387*, *388*, *393–397*, *423*, *424*, *435*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *664–666*
array_filter *(list_filter)*, 323
array_grade_up *(list_grade_up)*, 324
array_has *(list_contains)*, 317
array_has_all *(list_has_all)*, 325
~~~

array_has_any *(list_has_any)*, 326
array_indexof *(list_position)*, 333
array_inner_product, 55
array_inner_product(), *52–54*, *57*, *62*
array_intersect *(list_intersect)*, 328
array_length, 56
array_length(), *55*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
array_negative_dot_product *(array_negative_inner_product)*, 57
array_negative_inner_product, 57
array_negative_inner_product(), *52–54*, *56*, *62*
array_pop_back, 57
array_pop_front, 58
array_position *(list_position)*, 333
array_prepend, 58
array_push_back, 59
array_push_front, 59
array_reduce *(list_reduce)*, 335
array_resize *(list_resize)*, 336
array_reverse, 60
array_reverse_sort *(list_reverse_sort)*, 337
array_select *(list_select)*, 338
array_slice *(list_slice)*, 340
array_sort *(list_sort)*, 341
array_to_json, 60
array_to_string, 61
array_to_string_comma_default, 61
array_transform *(list_transform)*, 344
array_unique *(list_unique)*, 344
array_value, 62
array_value(), *52–54*, *56*, *57*
array_where *(list_where)*, 347
array_zip *(list_zip)*, 347
arrow->>, 62
arrow_scan, 63
arrow_scan_dumb, 63
ascii, 64
ascii(), *55*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
asin, 64
asinh, 65
atan, 65
atan2, 66
atanh, 66
avg, 67

bar, 67
bar(), *55*, *64*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
base64 *(to_base64)*, 640
bin, 68
bin(), *55*, *64*, *68*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
bit_and, 69
bit_count, 70
bit_length, 70
bit_length(), *55*, *64*, *68*, *69*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
bit_or, 71
bit_position, 72
bit_xor, 72
bitstring, 73
bitstring_agg, 73
bool_and, 74
bool_or, 75
broadcast, 75

can_cast_implicitly, 76
cardinality, 76
cast_to_type, 77
cbrt, 77
ceil, 78
ceiling *(ceil)*, 78
century, 78
char_length *(len)*, 308
character_length *(len)*, 308
check_peg_parser, 79
checkpoint, 79
chr, 80
chr(), *55*, *64*, *68*, *69*, *71*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
col_description, 80
collations, 81
combine, 81
concat, 82
concat(), *55*, *56*, *64*, *68*, *69*, *71*, *80*, *83*, *85*, *165*, *167*, *169*, *170*, *174*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340–342*, *344–348*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *387*, *388*, *393–397*, *423*, *424*, *435*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *664–666*
concat_ws, 83
concat_ws(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
constant_or_null, 83
contains, 84
contains(), *55*, *56*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *165*, *167*, *169*, *170*, *174*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340–342*, *344–348*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *387*, *388*, *393–397*, *423*, *424*, *435*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *664–666*
copy_database, 85
copy_dir, 86
corr, 86
cos, 87
cosh, 87
cot, 88
count, 88
count_if, 89
count_star, 89
countif *(count_if)*, 89
covar_pop, 90
covar_samp, 90
create_fts_index, 91
create_sort_key, 91
cume_dist, 92
current_catalog, 92
current_connection_id, 93
current_database, 93
current_date, 94
current_localtime, 94
current_localtimestamp, 95
current_query, 95
current_query_id, 96
current_role, 96
current_schema, 97
current_schemas, 97
current_setting, 98
current_transaction_id, 98
current_user, 99
currval, 99

damerau_levenshtein, 100
damerau_levenshtein(), *182*, *280*, *281*, *310*
database_list, 100
database_size, 101
date_add, 101
date_diff, 102
date_part, 102
date_sub, 103

date_trunc, 103
datediff *(date_diff)*, 102
datepart *(date_part)*, 102
datesub *(date_sub)*, 103
datetrunc *(date_trunc)*, 103
day, 104
dayname, 104
dayofmonth, 105
dayofweek, 105
dayofyear, 106
days_in_month, 106
dbgen, 107
dd, 107
decade, 108
decode, 108
decode(), *152*, *170*, *190*, *366*, *367*, *386*, *387*, *470*, *486*, *487*, *641*, *658*, *659*
degrees, 109
delta_domain_metadata, 110
delta_filter_pushdown_log, 110
delta_filter_pushdown_log_tpcds, 111
delta_get_transaction_version, 111
delta_list_files, 112
delta_scan, 112
delta_set_transaction_version, 114
dense_rank, 114
disable_checkpoint_on_shutdown, 115
disable_logging, 115
disable_object_cache, 115
disable_optimizer, 116
disable_peg_parser, 116
disable_print_progress_bar, 117
disable_profile, 117
disable_profiling, 117
disable_progress_bar, 118
disable_verification, 118
disable_verify_external, 118
disable_verify_fetch_row, 119
disable_verify_parallelism, 119
disable_verify_serializer, 119
divide, 120
drop_fts_index, 120
dsdgen, 121
duckdb_approx_database_count, 121
duckdb_columns, 122
duckdb_connection_count, 122
duckdb_constraints, 122
duckdb_coordinate_systems, 123
duckdb_databases, 123
duckdb_dependencies, 123
duckdb_extensions, 124
duckdb_external_file_cache, 124
duckdb_functions, 124
duckdb_indexes, 125
duckdb_keywords, 125
duckdb_log_contexts, 125
duckdb_logs, 126
duckdb_logs_parsed, 126
duckdb_memory, 127
duckdb_optimizers, 127
duckdb_prepared_statements, 127
duckdb_profiling_settings, 128
DuckDB_PROJ_Compiled_Version, 128
DuckDB_Proj_Version, 129
duckdb_schemas, 129
duckdb_secret_types, 130
duckdb_secrets, 130
duckdb_sequences, 130
duckdb_settings, 131
duckdb_table_sample, 131
duckdb_tables, 131
duckdb_temporary_files, 132
duckdb_types, 132
duckdb_variables, 132
duckdb_views, 133
ducklake_add_data_files, 133
ducklake_cleanup_old_files, 134
ducklake_commit, 135
ducklake_current_snapshot, 135
ducklake_delete_orphaned_files, 136
ducklake_expire_snapshots, 137
ducklake_flush_inlined_data, 137
ducklake_last_committed_snapshot, 138
ducklake_list_files, 138
ducklake_merge_adjacent_files, 139
ducklake_options, 140
ducklake_rewrite_data_files, 140
ducklake_scan, 141
ducklake_set_commit_message, 142
ducklake_set_option, 143
ducklake_settings, 143
ducklake_snapshots, 144
ducklake_table_changes, 144
ducklake_table_deletions, 145
ducklake_table_info, 146
ducklake_table_insertions, 146

editdist3 *(levenshtein)*, 310
element_at *(map_extract)*, 362
enable_checkpoint_on_shutdown, 147
enable_logging, 147
enable_object_cache, 148
enable_optimizer, 148
enable_peg_parser, 149
enable_print_progress_bar, 149
enable_profile, 150
enable_profiling, 150
enable_progress_bar, 151
enable_verification, 151
encode, 151
encode(), *109*, *170*, *190*, *366*, *367*, *386*, *387*, *470*, *486*, *487*, *641*, *658*, *659*
ends_with *(suffix)*, 630
entropy, 152
enum_code, 152
enum_first, 153
enum_last, 153
enum_range, 154
enum_range_boundary, 154
epoch, 155
epoch_ms, 155
epoch_ns, 156
epoch_us, 156
equi_width_bins, 157
era, 157
error, 158
even, 158
excel_text, 159
exp, 159
extension_versions, 160

factorial, 160
family, 161
favg, 161
fdiv, 162
fill, 162
filter *(list_filter)*, 323
finalize, 163
first, 163
first_value, 164
flatten, 164
flatten(), *55*, *56*, *82*, *85*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
floor, 165
fmod, 165
force_checkpoint, 166
format, 166
format(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
format_bytes, 167
format_bytes(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
format_pg_type, 168
format_type, 168
formatReadableDecimalSize, 169
formatReadableDecimalSize(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
formatReadableSize *(format_bytes)*, 167
from_base64, 170
from_base64(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167*, *169*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
from_binary *(unbin)*, 658
from_hex *(unhex)*, 659
from_json, 171

from_json_strict, 171
fsum *(kahan_sum)*, 298
functions, 172

gamma, 172
gcd *(greatest_common_divisor)*, 181
gen_random_uuid *(uuid)*, 667
generate_series, 173
generate_series(), *55*, *56*, *82*, *85*, *165*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
generate_subscripts, 174
geomean, 175
geometric_mean, 175
get_bit, 176
get_block_size, 176
get_current_time, 177
get_current_timestamp, 177
get_delta_test_expression, 178
get_type, 178
get_ui_url, 179
getvariable, 179
glob, 180
grade_up *(list_grade_up)*, 324
greatest, 180
greatest(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
greatest_common_divisor, 181
group_concat *(string_agg)*, 617

hamming, 182
hamming(), *100*, *280*, *281*, *310*
has_any_column_privilege, 182
has_column_privilege, 183
has_database_privilege, 183
has_foreign_data_wrapper_privilege, 184
has_function_privilege, 185
has_language_privilege, 185
has_schema_privilege, 186
has_sequence_privilege, 186
has_server_privilege, 187
has_table_privilege, 187
has_tablespace_privilege, 188
hash, 189
hash(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
hex, 189
hex(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167*, *169*, *170*, *181*, *189*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
histogram, 191
histogram_exact, 191
histogram_values, 192
hnsw_compact_index, 192
hnsw_index_scan, 193
host, 193
hour, 194
html_escape, 194
html_unescape, 195

iceberg_bucket, 195
iceberg_column_stats, 196
iceberg_load_table_response, 197
iceberg_metadata, 197
iceberg_partition_stats, 198
iceberg_scan, 199
iceberg_schema_properties, 201
iceberg_snapshots, 201
iceberg_table_properties, 202
iceberg_to_ducklake, 202
iceberg_truncate, 203
iceberg_verify_equality_deletes, 203
icu_calendar_names, 204
icu_collate_af, 204
icu_collate_am, 205
icu_collate_ar, 205
icu_collate_ar_sa, 206
icu_collate_as, 206
icu_collate_az, 207

icu_collate_be, 207
icu_collate_bg, 208
icu_collate_bn, 208
icu_collate_bo, 209
icu_collate_br, 209
icu_collate_bs, 210
icu_collate_ca, 210
icu_collate_ceb, 211
icu_collate_chr, 211
icu_collate_cs, 212
icu_collate_cy, 212
icu_collate_da, 213
icu_collate_de, 213
icu_collate_de_at, 214
icu_collate_dsb, 214
icu_collate_dz, 215
icu_collate_ee, 215
icu_collate_el, 216
icu_collate_en, 216
icu_collate_en_us, 217
icu_collate_eo, 217
icu_collate_es, 218
icu_collate_et, 218
icu_collate_fa, 219
icu_collate_fa_af, 219
icu_collate_ff, 220
icu_collate_fi, 220
icu_collate_fil, 221
icu_collate_fo, 221
icu_collate_fr, 222
icu_collate_fr_ca, 222
icu_collate_fy, 223
icu_collate_ga, 223
icu_collate_gl, 224
icu_collate_gu, 224
icu_collate_ha, 225
icu_collate_haw, 225
icu_collate_he, 226
icu_collate_he_il, 226
icu_collate_hi, 227
icu_collate_hr, 227
icu_collate_hsb, 228
icu_collate_hu, 228
icu_collate_hy, 229
icu_collate_id, 229
icu_collate_id_id, 230
icu_collate_ig, 230
icu_collate_is, 231
icu_collate_it, 231
icu_collate_ja, 232
icu_collate_ka, 232
icu_collate_kk, 233
icu_collate_kl, 233
icu_collate_km, 234
icu_collate_kn, 234
icu_collate_ko, 235
icu_collate_kok, 235
icu_collate_ku, 236
icu_collate_ky, 236
icu_collate_lb, 237
icu_collate_lij, 237
icu_collate_lkt, 238
icu_collate_ln, 238
icu_collate_lo, 239
icu_collate_lt, 239
icu_collate_lv, 240
icu_collate_mk, 240
icu_collate_ml, 241
icu_collate_mn, 241
icu_collate_mr, 242
icu_collate_ms, 242
icu_collate_mt, 243
icu_collate_my, 243
icu_collate_nb, 244
icu_collate_nb_no, 244
icu_collate_ne, 245
icu_collate_nl, 245
icu_collate_nn, 246
icu_collate_noaccent, 246
icu_collate_nso, 247
icu_collate_om, 247
icu_collate_or, 248
icu_collate_pa, 248
icu_collate_pa_in, 249
icu_collate_pl, 249
icu_collate_ps, 250
icu_collate_pt, 250
icu_collate_ro, 251
icu_collate_ru, 251
icu_collate_sa, 252
icu_collate_se, 252
icu_collate_si, 253
icu_collate_sk, 253
icu_collate_sl, 254
icu_collate_smn, 254
icu_collate_sq, 255

icu_collate_sr, 255
icu_collate_sr_ba, 256
icu_collate_sr_me, 256
icu_collate_sr_rs, 257
icu_collate_st, 257
icu_collate_sv, 258
icu_collate_sw, 258
icu_collate_ta, 259
icu_collate_te, 259
icu_collate_th, 260
icu_collate_tk, 260
icu_collate_tn, 261
icu_collate_to, 261
icu_collate_tr, 262
icu_collate_ug, 262
icu_collate_uk, 263
icu_collate_ur, 263
icu_collate_uz, 264
icu_collate_vi, 264
icu_collate_wae, 265
icu_collate_wo, 265
icu_collate_xh, 266
icu_collate_yi, 266
icu_collate_yo, 267
icu_collate_yue, 267
icu_collate_yue_cn, 268
icu_collate_zh, 268
icu_collate_zh_cn, 269
icu_collate_zh_hk, 269
icu_collate_zh_mo, 270
icu_collate_zh_sg, 270
icu_collate_zh_tw, 271
icu_collate_zu, 271
icu_sort_key, 272
ilike_escape, 272
ilike_escape(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
import_database, 273
in_search_path, 273
inet_client_addr, 274
inet_client_port, 274
inet_server_addr, 275
inet_server_port, 275
instr, 275
instr(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
is_histogram_other_bin, 276
isfinite, 277
isinf, 277
isnan, 278
isodow, 278
isoyear, 279

jaccard, 279
jaccard(), *100*, *182*, *280*, *281*, *310*
jaro_similarity, 280
jaro_similarity(), *100*, *182*, *280*, *281*, *310*
jaro_winkler_similarity, 281
jaro_winkler_similarity(), *100*, *182*, *280*, *310*
json, 282
json_array, 282
json_array_length, 283
json_contains, 283
json_deserialize_sql, 284
json_each, 284
json_execute_serialized_sql, 285
json_exists, 286
json_extract, 286
json_extract_path, 287
json_extract_path_text, 287
json_extract_string, 288
json_group_array, 288
json_group_object, 289
json_group_structure, 289
json_keys, 290
json_merge_patch, 290
json_object, 291
json_pretty, 291
json_quote, 292
json_serialize_plan, 292
json_serialize_sql, 293
json_structure, 294
json_transform, 294

json_transform_strict, 295
json_tree, 295
json_type, 296
json_valid, 297
json_value, 297
julian, 298

kahan_sum, 298
kurtosis, 299
kurtosis_pop, 299

lag, 300
lance_fts, 300
lance_hybrid_search, 301
lance_vector_search, 302
last, 303
last_day, 303
last_value, 304
lcase *(lower)*, 351
lcm *(least_common_multiple)*, 306
lead, 304
least, 305
least(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
least_common_multiple, 306
left, 306
left(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
left_grapheme, 307
left_grapheme(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
len, 308
len(), *55*, *56*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *165*, *167*, *169*, *170*, *174*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340–342*, *344–348*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *387*, *388*, *393–397*, *423*, *424*, *435*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *664–666*
length *(len)*, 308
length_grapheme, 309
length_grapheme(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
levenshtein, 310
levenshtein(), *100*, *182*, *280*, *281*
lgamma, 310
like_escape, 311
like_escape(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
list *(array_agg)*, 50
list_aggr *(list_aggregate)*, 312
list_aggregate, 312
list_aggregate(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_any_value, 312
list_append, 313
list_apply *(list_transform)*, 344
list_approx_count_distinct, 313

list_avg, 314
list_bit_and, 314
list_bit_or, 315
list_bit_xor, 315
list_bool_and, 316
list_bool_or, 316
list_cat *(list_concat)*, 317
list_concat, 317
list_concat(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *318–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_contains, 317
list_contains(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_cosine_distance, 318
list_cosine_distance(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_cosine_similarity, 319
list_cosine_similarity(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317*, *318*, *320–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_count, 319
list_distance, 320
list_distance(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–319*, *321–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_distinct, 321
list_distinct(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–320*, *322*, *323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_dot_product *(list_inner_product)*, 327
list_element *(list_extract)*, 322
list_entropy, 321
list_extract, 322
list_extract(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–321*, *323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_filter, 323
list_filter(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–322*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_first, 324
list_grade_up, 324
list_grade_up(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_has *(list_contains)*, 317
list_has_all, 325
list_has_all(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_has_any, 326
list_has_any(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325*, *327*, *328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_histogram, 326
list_indexof *(list_position)*, 333
list_inner_product, 327
list_inner_product(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325*, *326*, *328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_intersect, 328
list_intersect(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–327*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *387*, *435*, *470*, *664*
list_kurtosis, 328
list_kurtosis_pop, 329
list_last, 329
list_mad, 330
list_max, 330
list_median, 331
list_min, 331
list_mode, 332
list_negative_dot_product *(list_negative_inner_product)*, 332
list_negative_inner_product, 332
list_negative_inner_product(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*,

317–323, 325–328, 333, 335, 336, 338, 340, 342, 344–348, 387, 435, 470, 664
list_pack (list_value), 345
list_position, 333
list_position(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344–348, 387, 435, 470, 664
list_prepend, 334
list_product, 334
list_reduce, 335
list_reduce(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 336, 338, 340, 342, 344–348, 387, 435, 470, 664
list_resize, 336
list_resize(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 338, 340, 342, 344–348, 387, 435, 470, 664
list_reverse, 337
list_reverse_sort, 337
list_reverse_sort(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344–348, 388, 435, 470, 664
list_select, 338
list_select(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344–348, 388, 435, 470, 664
list_sem, 339
list_skewness, 339
list_slice, 340
list_slice(), 55, 56, 64, 68, 69, 71, 80, 82, 83, 85, 165, 167, 169, 170, 174, 181, 189, 190, 273, 276, 305, 307–309, 311, 312, 317–323, 325–328, 333, 335, 336, 338, 342, 344–348, 352, 353, 366, 367, 380, 382, 383, 388, 393–397, 423, 424, 435, 470, 471, 474–476, 480, 481, 486, 487, 613, 618–620, 627, 628, 630, 640, 641, 653, 654, 658–660, 664–666
list_sort, 341
list_sort(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 344–348, 388, 435, 470, 664
list_stddev_pop, 342
list_stddev_samp, 342
list_string_agg, 343
list_sum, 343
list_transform, 344
list_transform(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 345–348, 388, 435, 470, 664
list_unique, 344
list_unique(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344, 346–348, 388, 435, 470, 664
list_value, 345
list_value(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344, 345, 347, 348, 388, 435, 470, 664
list_var_pop, 346
list_var_samp, 346
list_where, 347
list_where(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344–346, 348, 388, 435, 470, 664
list_zip, 347
list_zip(), 55, 56, 82, 85, 165, 174, 308, 312, 317–323, 325–328, 333, 335, 336, 338, 340, 342, 344–347, 388, 435, 470, 664
listagg (string_agg), 617
ln, 348
load_aws_credentials, 349
log, 349
log10, 350
log2, 351
lower, 351
lower(), 55, 64, 68, 69, 71, 80, 82, 83, 85, 167, 169, 170, 181, 189, 190, 273, 276, 305, 307, 309, 311, 341, 352, 353, 366, 367, 380, 382, 383, 388, 393–397, 423, 424, 470, 471, 474–476, 480, 481, 486, 487, 613, 618–620, 627, 628, 630, 640, 641,

653, *654*, *658–660*, *665*, *666*
lpad, 352
lpad(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
ltrim, 353
ltrim(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*

mad, 354
make_date, 354
make_time, 355
make_timestamp, 356
make_timestamp_ms, 356
make_timestamp_ns, 357
make_timestamptz, 357
make_type, 358
map, 359
map_concat, 359
map_contains, 360
map_contains_entry, 360
map_contains_value, 361
map_entries, 361
map_extract, 362
map_extract_value, 362
map_from_entries, 363
map_keys, 363
map_to_pg_oid, 364
map_values, 364
max, 365
max_by *(arg_max)*, 37
md5, 365
md5(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
md5_number, 366
md5_number(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
md5_number_lower, 367
md5_number_upper, 368
mean *(avg)*, 67
median, 368
metadata_info, 369
microsecond, 369
millennium, 370
millisecond, 370
min, 371
min_by *(arg_min)*, 44
minute, 371
mismatches *(hamming)*, 182
mod, 372
mode, 372
month, 373
monthname, 373
multiply, 374
murmur3_32, 374
mysql_clear_cache, 375
mysql_debug_execution_plan, 375
mysql_execute, 376
mysql_explain_federated, 376
mysql_query, 377

nanosecond, 377
netmask, 378
network, 378
nextafter, 379
nextval, 379
nfc_normalize, 380
nfc_normalize(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*,

613, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
normalized_interval, 380
not-~~, 381
not-~~*, 381
not_ilike_escape, 382
not_ilike_escape(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
not_like_escape, 383
not_like_escape(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
now *(get_current_timestamp)*, 177
nth_value, 384
ntile, 384
nullif, 385

obj_description, 385
octet_length, 386
octet_length(), *109*, *152*, *170*, *190*, *366*, *367*, *387*, *470*, *486*, *487*, *641*, *658*, *659*
or-, 386
or--or, 387
ord *(unicode)*, 660

parquet_bloom_probe, 388
parquet_file_metadata, 388
parquet_full_metadata, 389
parquet_kv_metadata, 389
parquet_metadata, 390
parquet_scan, 390
parquet_schema, 391
parse_delta_filter_logline, 392
parse_dirname, 392
parse_dirname(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *394–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
parse_dirpath, 393
parse_dirpath(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393*, *395–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
parse_duckdb_log_message, 394
parse_filename, 395
parse_filename(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393*, *394*, *396*, *397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
parse_formatted_bytes, 396
parse_formatted_bytes(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–395*, *397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
parse_path, 397
parse_path(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–396*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
percent_rank, 398
pg_clear_cache, 398

pg_collation_is_visible, 399
pg_conf_load_time, 399
pg_conversion_is_visible, 400
pg_function_is_visible, 400
pg_get_constraintdef, 401
pg_get_expr, 401
pg_get_viewdef, 402
pg_has_role, 402
pg_is_other_temp_schema, 403
pg_my_temp_schema, 403
pg_opclass_is_visible, 404
pg_operator_is_visible, 404
pg_opfamily_is_visible, 405
pg_postmaster_start_time, 405
pg_size_pretty, 406
pg_sleep, 406
pg_table_is_visible, 407
pg_timezone_names, 407
pg_ts_config_is_visible, 408
pg_ts_dict_is_visible, 408
pg_ts_parser_is_visible, 409
pg_ts_template_is_visible, 409
pg_type_is_visible, 410
pg_typeof, 410
pi, 411
platform, 411
position *(instr)*, 275
postgres_attach, 412
postgres_configure_pool, 412
postgres_execute, 414
postgres_hstore_get, 414
postgres_hstore_to_json, 415
postgres_query, 415
postgres_scan, 416
postgres_scan_pushdown, 416
pow, 417
power *(pow)*, 417
pragma_collations, 417
pragma_database_size, 418
pragma_hnsw_index_info, 418
pragma_metadata_info, 419
pragma_platform, 419
pragma_rtree_index_info, 420
pragma_show, 420
pragma_storage_info, 421
pragma_table_info, 421
pragma_user_agent, 422
pragma_version, 422
prefix, 422
prefix(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
printf, 423
printf(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
product, 424

quack_active_connections, 424
quack_check_token, 425
quack_clear_cache, 425
quack_identify, 426
quack_nop_authorization, 426
quack_query, 427
quack_query_by_name, 427
quack_serve, 428
quack_server_list, 428
quack_stop, 429
quack_uri_parser, 429
quantile *(quantile_disc)*, 430
quantile_cont, 430
quantile_disc, 430
quarter, 431
query, 432
query_table, 432

r_dataframe_scan, 433
radians, 433
random, 434
range, 434
range(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *388*, *470*, *664*
rank, 436
rank_dense, 436
read_avro, 436

read_blob, 437
read_csv, 438
read_csv_auto, 440
read_duckdb, 443
read_json, 444
read_json_auto, 446
read_json_objects, 447
read_json_objects_auto, 449
read_ndjson, 450
read_ndjson_auto, 451
read_ndjson_objects, 453
read_parquet, 454
read_postgres_binary, 455
read_text, 456
read_vortex, 456
read_xlsx, 457
reduce *(list_reduce)*, 335
regexp_escape, 458
regexp_escape(), *459–463*, *619*
regexp_extract, 458
regexp_extract(), *458*, *460–463*, *619*
regexp_extract_all, 460
regexp_extract_all(), *458*, *459*, *461–463*, *619*
regexp_full_match, 461
regexp_full_match(), *458–460*, *462*, *463*, *619*
regexp_matches, 461
regexp_matches(), *458–461*, *463*, *619*
regexp_replace, 462
regexp_replace(), *458–462*, *619*
regexp_split_to_array *(string_split_regex)*, 618
regexp_split_to_table, 463
regr_avgx, 463
regr_avgy, 464
regr_count, 464
regr_intercept, 465
regr_r2, 465
regr_slope, 466
regr_sxx, 466
regr_sxy, 467
regr_syy, 467
remap_struct, 468
remove_iceberg_schema_properties, 468
remove_iceberg_table_properties, 469
repeat, 469
repeat(), *55*, *56*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *165*, *167*, *169*, *170*, *174*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340–342*, *344–348*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *435*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *664–666*
repeat_row, 470
replace, 471
replace(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
replace_type, 472
reservoir_quantile, 472
reverse, 474
reverse(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *475*, *476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
right, 474
right(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474*, *476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
right_grapheme, 475
right_grapheme(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167*, *169*, *170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474*, *475*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*,

640, *641*, *653*, *654*, *658–660*, *665*, *666*
round, 476
round_even, 477
roundbankers, 477
row, 478
row_number, 478
row_to_json, 479
rpad, 479
rpad(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
rtree_index_dump, 480
rtree_index_scan, 480
rtrim, 481
rtrim(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*

second, 482
sem, 482
seq_scan, 483
session_user, 483
set_bit, 483
set_iceberg_schema_properties, 484
set_iceberg_table_properties, 485
setseed, 485
sha1, 486
sha1(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
sha256, 487
sha256(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
shapefile_meta, 488
shobj_description, 488
show, 489
show_databases, 489
show_tables, 489
show_tables_expanded, 490
sign, 490
signbit, 491
sin, 491
sinh, 492
skewness, 492
sleep_ms, 493
sniff_csv, 493
split *(string_split)*, 617
split_part, 496
sql_auto_complete, 496
sqlite_attach, 497
sqlite_query, 498
sqlite_scan, 498
sqrt, 499
ST_Affine, 499
ST_Area, 501
ST_Area_Spheroid, 502
st_asbinary *(st_aswkb)*, 509
ST_AsGeoJSON, 502
ST_AsHEXWKB, 503
ST_AsMVT, 504
ST_AsMVTGeom, 506
ST_AsSVG, 507
st_astext, 507
st_astext(), *509*, *524*, *546*
st_aswkb, 509
st_aswkb(), *508*, *524*, *546*
st_aswkt *(st_astext)*, 507
ST_Azimuth, 509
ST_Boundary, 510
ST_Buffer, 510
ST_BuildArea, 511
ST_Centroid, 512
ST_ClosestPoint, 513
ST_Collect, 513
ST_CollectionExtract, 514

ST_ConcaveHull, 515
ST_Contains, 516
ST_ContainsProperly, 517
ST_ConvexHull, 517
ST_CoverageClean, 518
ST_CoverageInvalidEdges, 519
ST_CoverageInvalidEdges_Agg, 519
ST_CoverageSimplify, 520
ST_CoverageSimplify_Agg, 521
ST_CoverageUnion, 522
ST_CoverageUnion_Agg, 522
ST_CoveredBy, 523
ST_Covers, 523
ST_Crosses, 524
st_crs, 524
st_crs(), *508*, *509*, *546*
ST_Difference, 525
ST_Dimension, 525
ST_Disjoint, 526
ST_Distance, 526
ST_Distance_GEOS, 527
ST_Distance_Sphere, 528
ST_Distance_Spheroid, 528
ST_Drivers, 529
ST_Dump, 530
ST_DWithin, 531
ST_DWithin_GEOS, 531
ST_DWithin_Spheroid, 532
ST_EndPoint, 533
ST_Envelope, 533
ST_Envelope_Agg, 534
ST_Equals, 534
ST_Expand, 535
ST_Extent, 536
ST_Extent_Agg, 536
ST_Extent_Approx, 537
ST_ExteriorRing, 537
ST_FlipCoordinates, 538
ST_Force2D, 539
ST_Force3DM, 539
ST_Force3DZ, 540
ST_Force4D, 541
ST_GeneratePoints, 542
ST_GeometryType, 543
ST_GeomFromGeoJSON, 544
ST_GeomFromHEXEWKB, 544
ST_GeomFromHEXWKB, 545
ST_GeomFromText, 545
st_geomfromwkb, 546
st_geomfromwkb(), *508*, *509*, *524*
ST_HasM, 547
ST_HasZ, 548
ST_Hilbert, 549
ST_InteriorRingN, 549
ST_InterpolatePoint, 550
ST_Intersection, 551
ST_Intersection_Agg, 551
ST_Intersects, 552
st_intersects_extent *(list_has_any)*, 326
ST_IsClosed, 552
ST_IsEmpty, 553
ST_IsRing, 553
ST_IsSimple, 554
ST_IsValid, 554
ST_Length, 555
ST_Length_Spheroid, 556
ST_LineInterpolatePoint, 556
ST_LineInterpolatePoints, 557
ST_LineLocatePoint, 558
ST_LineMerge, 558
ST_LineString2DFromWKB, 559
ST_LineSubstring, 559
ST_LocateAlong, 560
ST_LocateBetween, 561
ST_M, 562
ST_MakeBox2D, 562
ST_MakeEnvelope, 563
ST_MakeLine, 563
ST_MakePoint, 564
ST_MakePolygon, 565
ST_MakeValid, 566
ST_MaximumInscribedCircle, 566
ST_MemUnion_Agg, 567
ST_MinimumRotatedRectangle, 568
ST_MMax, 568
ST_MMin, 569
ST_Multi, 569
ST_NGeometries, 570
ST_NInteriorRings, 571
ST_Node, 571
ST_Normalize, 572
ST_NPoints, 572
ST_NumGeometries, 573
ST_NumInteriorRings, 574
ST_NumPoints, 574

ST_Overlaps, 575
ST_Perimeter, 576
ST_Perimeter_Spheroid, 576
ST_Point, 577
ST_Point2D, 578
ST_Point2DFromWKB, 578
ST_Point3D, 579
ST_Point4D, 579
ST_PointN, 580
ST_PointOnSurface, 580
ST_Points, 581
ST_Polygon2DFromWKB, 581
ST_Polygonize, 582
ST_QuadKey, 583
ST_Read, 584
ST_Read_Meta, 586
ST_ReadOSM, 586
ST_ReadSHP, 588
ST_ReducePrecision, 588
ST_RemoveRepeatedPoints, 589
ST_Reverse, 590
ST_Rotate, 590
ST_RotateX, 591
ST_RotateY, 591
ST_RotateZ, 592
ST_Scale, 593
st_setcrs, 593
ST_ShortestLine, 594
ST_Simplify, 594
ST_SimplifyPreserveTopology, 595
ST_Snap, 595
ST_StartPoint, 596
ST_SymDifference, 596
ST_TileEnvelope, 597
ST_Touches, 598
ST_Transform, 598
ST_Translate, 600
ST_TransScale, 601
ST_Union, 602
ST_Union_Agg, 602
ST_VoronoiDiagram, 603
ST_Within, 603
ST_WithinProperly, 604
ST_X, 604
ST_XMax, 605
ST_XMin, 606
ST_Y, 607
ST_YMax, 607
ST_YMin, 608
ST_Z, 609
ST_ZMax, 609
ST_ZMFlag, 610
ST_ZMin, 611
start_ui, 611
start_ui_server, 612
starts_with, 612
starts_with(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
stats, 613
stddev *(stddev_samp)*, 614
stddev_pop, 614
stddev_samp, 614
stem, 615
stop_ui_server, 615
storage_info, 616
str_split *(string_split)*, 617
str_split_regex *(string_split_regex)*, 618
strftime, 616
string_agg, 617
string_split, 617
string_split(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *619*, *620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
string_split_regex, 618
string_split_regex(), *458–463*
string_to_array *(string_split)*, 617
strip_accents, 619
strip_accents(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618*, *620*, *627*, *628*, *630*, *640*,

641, *653*, *654*, *658–660*, *665*, *666*
strlen, 620
strlen(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618*, *619*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
strpos *(instr)*, 275
strptime, 620
struct_concat, 621
struct_contains, 622
struct_extract, 622
struct_extract_at, 623
struct_has *(struct_contains)*, 622
struct_indexof *(struct_position)*, 625
struct_insert, 623
struct_keys, 624
struct_pack, 624
struct_position, 625
struct_update, 625
struct_values, 626
substr *(substring)*, 626
substring, 626
substring(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
substring_grapheme, 627
substring_grapheme(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
subtract, 628
suffix, 630
suffix(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
sum, 630
sum_no_overflow, 631
sumkahan *(kahan_sum)*, 298
summary, 632
switch, 632

table_info, 633
tan, 633
tanh, 634
test_all_types, 634
test_vector_types, 635
text, 635
time_bucket, 636
timetz_byte_comparable, 637
timezone, 637
timezone_hour, 638
timezone_minute, 639
to_base, 639
to_base(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307–309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *641*, *653*, *654*, *658–660*, *665*, *666*
to_base64, 640
to_base64(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *653*, *654*, *658–660*, *665*, *666*
to_binary *(bin)*, 68
to_centuries, 641
to_days, 642
to_decades, 642
to_hex *(hex)*, 189
to_hours, 643
to_json, 643
to_microseconds, 644

to_millennia, 644
to_milliseconds, 645
to_minutes, 645
to_months, 646
to_quarters, 646
to_seconds, 647
to_timestamp, 647
to_weeks, 648
to_years, 648
today, 649
tpcds, 649
tpcds_answers, 650
tpcds_queries, 650
tpch, 651
tpch_answers, 651
tpch_queries, 652
transaction_timestamp *(get_current_timestamp)*, 177
translate, 652
translate(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *654*, *658–660*, *665*, *666*
trim, 653
trim(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *658–660*, *665*, *666*
trunc, 654
truncate_duckdb_logs, 655
try_strptime, 656
txid_current, 656
typeof, 657

ucase *(upper)*, 664
ui_is_started, 657
unbin, 658
unbin(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *659*, *660*, *665*, *666*
unhex, 659
unhex(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *109*, *152*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *386–388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658*, *660*, *665*, *666*
unicode, 660
unicode(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658*, *659*, *665*, *666*
union_extract, 660
union_tag, 661
union_value, 662
unity_catalog_checkpoint_table, 662
unity_catalog_force_checkpoint_table, 663
unnest, 663
unpivot_list, 664
unpivot_list(), *55*, *56*, *82*, *85*, *165*, *174*, *308*, *312*, *317–323*, *325–328*, *333*, *335*, *336*, *338*, *340*, *342*, *344–348*, *388*, *435*, *470*
upper, 664
upper(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *666*
url_decode, 665
url_decode(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*,

388, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
url_encode, 666
url_encode(), *55*, *64*, *68*, *69*, *71*, *80*, *82*, *83*, *85*, *167–170*, *181*, *189*, *190*, *273*, *276*, *305*, *307*, *309*, *311*, *341*, *352*, *353*, *366*, *367*, *380*, *382*, *383*, *388*, *393–397*, *423*, *424*, *470*, *471*, *474–476*, *480*, *481*, *486*, *487*, *613*, *618–620*, *627*, *628*, *630*, *640*, *641*, *653*, *654*, *658–660*, *665*, *666*
user, 667
user_agent, 667
uuid, 667
uuid_extract_timestamp, 668
uuid_extract_version, 668
uuidv4, 669
uuidv7, 669

var_pop, 670
var_samp, 670
variance *(var_samp)*, 670
variant_bytes_to_variant, 671
variant_extract, 671
variant_extract(), *672*, *673*
variant_normalize, 672
variant_normalize(), *672*, *673*
variant_to_parquet_variant, 673
variant_typeof, 673
variant_typeof(), *672*
vector_type, 674
verify_external, 674
verify_fetch_row, 675
verify_parallelism, 675
verify_serializer, 675
version, 676
vortex_scan, 676
vss_join, 677
vss_match, 677

wavg, 678
week, 678
weekday, 679
weekofyear, 679
weighted_avg, 680
which_secret, 680
whoami, 681
write_blob, 681
write_log, 682

xor, 682

year, 683
yearweek, 683