Package: sedonafns (via r-universe)

October 29, 2026

Title Apache SedonaDB Function Definitions and Documentation

Version 0.5.0.9000

Description Provides function definitions and documentation for use in 'SedonaDB'.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

Suggests sedonadb, testthat (>= 3.0.0)

Config/build/bootstrap TRUE

Config/Needs/build glue, here, rlang, roxygen2, yaml

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

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

Date/Publication 2026-10-29 13:48:59 UTC

RemoteUrl https://github.com/apache/sedona-db

RemoteRef HEAD

RemoteSha 9073652827f823791787ca1146e28d396098dc99

RemoteSubdir r/sedonafns

Contents

rs_as_geo_tiff 6
rs_as_raster 7
rs_band_is_no_data 8
rs_band_no_data_value 9
rs_band_path 9
rs_band_pixel_type 10
rs_band_to_dim 10
rs_clip 11

rs_contains . . . 12
rs_convex_hull . . . 13
rs_crs . . . 13
rs_dim_names . . . 14
rs_dim_size . . . 15
rs_dim_to_band . . . 16
rs_envelope . . . 16
rs_example . . . 17
rs_from_gdal_raster . . . 17
rs_from_path . . . 18
rs_geo_reference . . . 18
rs_geo_transform . . . 19
rs_height . . . 20
rs_intersects . . . 20
rs_is_empty . . . 21
rs_make_empty_raster . . . 22
rs_meta_data . . . 23
rs_min_convex_hull . . . 23
rs_num_bands . . . 24
rs_num_dimensions . . . 25
rs_pixel_as_centroid . . . 25
rs_pixel_as_point . . . 26
rs_pixel_as_polygon . . . 27
rs_polygonize . . . 27
rs_raster_to_world_coord . . . 28
rs_raster_to_world_coord_x . . . 29
rs_raster_to_world_coord_y . . . 29
rs_replace_band_no_data_value . . . 30
rs_reproject_match . . . 31
rs_resample . . . 32
rs_rotation . . . 33
rs_scale_x . . . 33
rs_scale_y . . . 34
rs_set_band_no_data_value . . . 35
rs_set_crs . . . 35
rs_set_geo_reference . . . 36
rs_set_srid . . . 37
rs_set_value . . . 37
rs_shape . . . 38
rs_skew_x . . . 39
rs_skew_y . . . 39
rs_slice . . . 40
rs_slice_range . . . 41
rs_srid . . . 42
rs_stack . . . 42
rs_stack_aggr . . . 43
rs_summary_stats . . . 44
rs_summary_stats_all . . . 45

rs_tile . . . 45
rs_upper_left_x . . . 46
rs_upper_left_y . . . 47
rs_value . . . 48
rs_values . . . 48
rs_width . . . 49
rs_within . . . 50
rs_world_to_raster_coord . . . 51
rs_world_to_raster_coord_x . . . 51
rs_world_to_raster_coord_y . . . 52
rs_zonal_stats . . . 53
rs_zonal_stats_all . . . 54
sd_affine . . . 55
sd_analyze_agg . . . 56
sd_area . . . 56
sd_as_binary . . . 57
sd_as_ewkb . . . 58
sd_as_geo_json . . . 58
sd_as_text . . . 59
sd_azimuth . . . 59
sd_boundary . . . 60
sd_buffer . . . 60
sd_build_area . . . 61
sd_cell_id_from_point . . . 62
sd_centroid . . . 62
sd_closest_point . . . 63
sd_collect_agg . . . 63
sd_concave_hull . . . 64
sd_contains . . . 64
sd_convex_hull . . . 65
sd_convex_hull_agg . . . 66
sd_covered_by . . . 66
sd_covering_cell_ids . . . 67
sd_covers . . . 68
sd_crosses . . . 68
sd_crs . . . 69
sd_d_within . . . 69
sd_delaunay_triangles . . . 70
sd_difference . . . 71
sd_dimension . . . 71
sd_disjoint . . . 72
sd_distance . . . 72
sd_dump . . . 73
sd_end_point . . . 73
sd_envelope . . . 74
sd_envelope_agg . . . 74
sd_equals . . . 75
sd_exterior_ring . . . 76

sd_flip_coordinates . . . 76
sd_force2d . . . 77
sd_force3d . . . 77
sd_force3dm . . . 78
sd_force4d . . . 78
sd_geo_hash . . . 79
sd_geog_from_wkb . . . 80
sd_geog_from_wkt . . . 80
sd_geog_point . . . 81
sd_geom_coll_from_text . . . 81
sd_geom_from_ewkb . . . 82
sd_geom_from_ewkt . . . 82
sd_geom_from_wkb . . . 83
sd_geom_from_wkt . . . 83
sd_geometries . . . 84
sd_geometry_n . . . 85
sd_geometry_type . . . 85
sd_has_m . . . 86
sd_has_z . . . 86
sd_hausdorff_distance . . . 87
sd_interior_ring_n . . . 88
sd_intersection . . . 88
sd_intersection_agg . . . 89
sd_intersects . . . 89
sd_is_closed . . . 90
sd_is_collection . . . 90
sd_is_empty . . . 91
sd_is_polygon_ccw . . . 91
sd_is_polygon_cw . . . 92
sd_is_ring . . . 92
sd_is_simple . . . 93
sd_is_valid . . . 94
sd_is_valid_reason . . . 94
sd_knn . . . 95
sd_length . . . 96
sd_line_from_text . . . 96
sd_line_interpolate_point . . . 97
sd_line_locate_point . . . 97
sd_line_merge . . . 98
sd_line_string_from_wkb . . . 99
sd_line_substring . . . 99
sd_longest_line . . . 100
sd_m . . . 101
sd_m_line_from_text . . . 101
sd_m_max . . . 102
sd_m_min . . . 102
sd_m_point_from_text . . . 103
sd_m_poly_from_text . . . 103

sd_make_envelope . . . 104
sd_make_line . . . 105
sd_make_valid . . . 105
sd_max_distance . . . 106
sd_minimum_clearance . . . 107
sd_minimum_clearance_line . . . 107
sd_missing_arg . . . 108
sd_n_points . . . 108
sd_n_rings . . . 109
sd_normalize . . . 109
sd_num_geometries . . . 110
sd_num_interior_rings . . . 110
sd_num_points . . . 111
sd_overlaps . . . 111
sd_perimeter . . . 112
sd_point . . . 112
sd_point_from_text . . . 113
sd_point_from_wkb . . . 114
sd_point_m . . . 114
sd_point_n . . . 115
sd_point_on_surface . . . 115
sd_point_z . . . 116
sd_point_zm . . . 117
sd_points . . . 117
sd_polygon_from_text . . . 118
sd_polygonize . . . 119
sd_polygonize_agg . . . 119
sd_reduce_precision . . . 120
sd_relate . . . 120
sd_reverse . . . 121
sd_rotate . . . 122
sd_rotate_x . . . 122
sd_rotate_y . . . 123
sd_scale . . . 123
sd_segmentize . . . 124
sd_set_crs . . . 125
sd_set_srid . . . 125
sd_shortest_line . . . 126
sd_simplify . . . 126
sd_simplify_preserve_topology . . . 127
sd_snap . . . 128
sd_srid . . . 128
sd_start_point . . . 129
sd_sym_difference . . . 129
sd_tessellate_geog . . . 130
sd_tessellate_geom . . . 131
sd_to_geography . . . 131
sd_to_geometry . . . 132

sd_touches . . . 133
sd_transform . . . 133
sd_translate . . . 134
sd_unary_union . . . 135
sd_union . . . 135
sd_union_agg . . . 136
sd_within . . . 136
sd_x . . . 137
sd_x_max . . . 137
sd_x_min . . . 138
sd_y . . . 138
sd_y_max . . . 139
sd_y_min . . . 139
sd_z . . . 140
sd_z_max . . . 141
sd_z_min . . . 141
sd_zmflag . . . 142

Index **143**

`rs_as_geo_tiff` *Encodes a raster as a GeoTIFF and returns the bytes as binary, with optional compression and tiling.*

Description

`RS_AsGeoTiff` encodes a raster as a GeoTIFF (via GDAL) and returns the file bytes as a binary value — the inverse of reading a GeoTIFF into a raster. The result is `NULL` for a `NULL` raster. A `NULL` option argument means "not specified": e.g. a `NULL` compression writes an uncompressed GeoTIFF rather than propagating `NULL` to the result. The two-argument form `RS_AsGeoTiff(raster, tile_size)` writes a tiled GeoTIFF with square `tile_size` blocks. Compression is selected by name; a JPEG quality factor and explicit tile dimensions can be supplied via the longer overloads.

Usage

```
rs_as_geo_tiff(...)
```

Arguments

`...`	Supported combinations: • rast (raster) • rast (raster), tile_size (integer) • rast (raster), compression (string), quality (double) • rast (raster), compression (string), quality (double), tile_size (integer) • rast (raster), compression (string), quality (double), tile_width (integer), tile_height (integer)

Value

(binary)

See Also

SedonaDB SQL documentation for RS_ASGEOTIFF

`rs_as_raster` *Rasterizes a geometry using the grid of a reference raster.*

Description

Creates a single-band in-db raster by rasterizing `geometry` onto the grid defined by `reference_raster`. The output pixel type is controlled by `pixel_type`. Optional arguments control whether pixels touched by the geometry are burned, the value burned into covered pixels, the nodata fill value, and whether the output extent is cropped to the geometry extent or kept equal to the full reference raster extent. When both `geometry` and `reference_raster` have a CRS and they differ, `geometry` is reprojected into the reference raster's CRS before rasterization. An error is returned if exactly one of them has a CRS.

Usage

```
rs_as_raster(
  geometry = sd_missing_arg(),
  reference_raster = sd_missing_arg(),
  pixel_type = sd_missing_arg(),
  all_touched = sd_missing_arg(),
  burn_value = sd_missing_arg(),
  nodata_value = sd_missing_arg(),
  use_geometry_extent = sd_missing_arg()
)
```

Arguments

`geometry`	(geometry): Input geometry
`reference_raster`	(raster): Input raster
`pixel_type`	(string): Input string
`all_touched`	(boolean): Input boolean
`burn_value`	(double): Input double
`nodata_value`	(double): Input double
`use_geometry_extent`	(boolean): Input boolean

Value

(raster)

See Also

SedonaDB SQL documentation for RS_ASRASTER

`rs_band_is_no_data` *Returns true if every pixel of the band holds the band's nodata value. Returns false if any pixel holds data, or if the band has no nodata value.*

Description

A pixel counts as nodata when it equals the band's nodata value; a NaN pixel matches a NaN nodata value. A band without a nodata value is never all nodata, even if every pixel is 0. Only 2-D rasters are supported.

Usage

```
rs_band_is_no_data(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`band`	(integer): Band index (1-based). Defaults to 1 if not specified.

Value

(boolean)

See Also

SedonaDB SQL documentation for RS_BANDISNODATA

`rs_band_no_data_value` *Returns the nodata value of the specified band as a double. Returns null if the band has no nodata value defined.*

Description

Returns the nodata value of the specified band as a double. Returns null if the band has no nodata value defined.

Usage

```
rs_band_no_data_value(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`band`	(int): Band index (1-based). Defaults to 1 if not specified.

Value

(float64)

See Also

SedonaDB SQL documentation for RS_BANDNODATAVALUE

`rs_band_path` *Retrieves the file path of an out-of-database (out-db) raster band, returning the external raster file location referenced by the raster.*

Description

Primarily used with out-db rasters, where only raster path and geo-referencing metadata are stored in the database.

Usage

```
rs_band_path(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(utf8)

See Also

SedonaDB SQL documentation for RS_BANDPATH

`rs_band_pixel_type` *Returns the pixel data type of the specified band as a string.*

Description

Returns the pixel data type of the specified band as a string.

Usage

```
rs_band_pixel_type(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`band`	(int): Band index (1-based). Defaults to 1 if not specified.

Value

(utf8)

See Also

SedonaDB SQL documentation for RS_BANDPIXELTYPE

`rs_band_to_dim` *Collapses all bands into a single band with a new dimension.*

Description

Collapses all bands in a raster into a single band by introducing a new dimension. The new dimension is prepended to the existing dimensions, with size equal to the number of bands. Band data is concatenated in band order. All bands must have identical dimension names, shapes, data types, and nodata values. If any of these differ, an error is returned — collapsing bands with different nodata sentinels would silently lose information about which sentinel applies where. The new dimension name must not already exist on the input bands. For example, `RS_BandToDim(raster_with_y_x, 'y')` is rejected because the output would carry duplicate `y` dimensions and the round-trip through RS_DimToBand cannot recover the original layout. This is the inverse of RS_DimToBand. A round-trip `RS_BandToDim(RS_DimToBand(raster, 'dim'), 'dim')` recovers the original data layout. This is useful for converting GeoTIFF-style multi-band rasters into a single N-dimensional chunk for export to formats like Zarr.

Usage

```
rs_band_to_dim(rast = sd_missing_arg(), dim_name = sd_missing_arg())
```

Arguments

rast	(raster): Input raster
dim_name	(utf8): Name for the new dimension (e.g., 'time', 'band').

Value

(raster)

See Also

SedonaDB SQL documentation for RS_BANDTODIM

---

rs_clip

Clips a raster to a geometry, setting pixels outside the geometry to nodata and (by default) cropping to the geometry's bounding box.

Description

`RS_Clip` clips a raster to a geometry, in the spirit of PostGIS `ST_Clip`. Pixels of the selected band that fall outside the geometry are set to the band's nodata value (or `no_data_value` if given); pixels inside are kept. When `crop` is true (the default) the output is cropped to the geometry's bounding box intersected with the raster extent, snapped to the pixel grid; unselected pixels within that window remain as nodata padding. Otherwise the original extent is preserved. The geometry is reprojected into the raster's CRS when both carry one. If exactly one side has a CRS, the call errors; if neither does, the geometry is used as-is. A pixel is selected only when its center falls inside the geometry. A geometry smaller than a pixel, or a thin sliver, can therefore select no pixels even though it overlaps the raster; set `all_touched` to true to keep every pixel the geometry touches instead. The clip is a 2-D `(y, x)` operation. For an N-D raster (bands with extra dimensions such as `time`), the same mask and crop window are broadcast across every non-spatial plane, so the non-spatial dimensions are preserved and only the `(y, x)` extent changes.

Usage

```
rs_clip(
  rast = sd_missing_arg(),
  band = sd_missing_arg(),
  geom = sd_missing_arg(),
  all_touched = sd_missing_arg(),
  no_data_value = sd_missing_arg(),
  crop = sd_missing_arg(),
  lenient = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`band`	(integer): 1-based index of the band to clip, or 0 to clip every band. A NULL band yields a NULL result.
`geom`	(geometry): Clip geometry. Reprojected into the raster's CRS when both carry one; it is an error for exactly one side to have a CRS.
`all_touched`	(boolean): If true, keep every pixel the geometry touches; otherwise keep only pixels whose center falls inside it. Defaults to false.
`no_data_value`	(double): Value written to pixels outside the geometry. Defaults to the band's own nodata value, or the band data type's minimum if it has none. It is an error if the value does not fit the band's data type.
`crop`	(boolean): Crop the output to the geometry's bounding box. Defaults to true.
`lenient`	(boolean): If true, a geometry that does not intersect the raster yields NULL; if false, it raises an error. Defaults to true.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_CLIP

`rs_contains` *Returns true if the first argument's extent contains the second.*

Description

Returns `true` if the convex hull of the first argument completely contains the second argument. Both rasters and geometries are accepted in either argument position. When two rasters are provided, their convex hulls are compared. If the arguments have different CRSes, the geometry is transformed into the raster's CRS before evaluating the predicate. For two rasters, the second raster's extent is transformed into the first raster's CRS. If the preferred transformation fails, the extent of both sides are transformed to WGS 84 as a fallback. If either argument has no CRS set, the comparison is performed directly without CRS transformation.

Usage

```
rs_contains(...)
```

Arguments

`...`	Supported combinations: • rast (raster), geom (geometry) • geom (geometry), rast (raster) • rast_a (raster), rast_b (raster)

Value

(boolean)

See Also

SedonaDB SQL documentation for RS_CONTAINS

---

rs_convex_hull *Returns the convex hull geometry of a raster.*

Description

Returns the convex hull geometry of a raster.

Usage

```
rs_convex_hull(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_CONVEXHULL

---

rs_crs *Returns the CRS string for a raster.*

Description

Returns the CRS string for a raster.

Usage

```
rs_crs(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(string)

See Also

SedonaDB SQL documentation for RS_CRS

`rs_dim_names` *Returns the ordered list of dimension names for a raster band.*

Description

Returns the ordered list of dimension names for a raster band. Standard 2D rasters have dimensions `["y", "x"]`. N-dimensional rasters may include additional dimensions such as `time`, `pressure`, `wavelength`, etc. The dimension names correspond to the entries in RS_Shape — the i-th name matches the i-th size. When the band index is omitted, all bands must agree on their dimension names; if they disagree, an error is returned prompting the user to specify a band index.

Usage

```
rs_dim_names(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`band`	(int): Band index (1-based). A null or out-of-range index returns null.

Value

(list)

See Also

SedonaDB SQL documentation for RS_DIMNAMES

`rs_dim_size` *Returns the size of a named dimension in a raster band.*

Description

Returns the size of a named dimension in a raster. For example, `RS_DimSize(raster, 'x')` returns the width and `RS_DimSize(raster, 'time')` returns the number of time steps. This is equivalent to looking up a specific entry in RS_Shape by name rather than by position. ### Heterogeneous rasters Bands within a raster can have different dimensionality. When the band index is omitted, `RS_DimSize` considers only the bands that carry the named dimension ("pass-through"): - If at least one band carries the dimension and they all agree on its size, that size is returned. - If the bands that carry the dimension disagree on its size, an error is returned prompting the user to specify a band index. - If **no** band carries the dimension, an error is returned. This usually signals a typo'd dimension name. When the band index is supplied explicitly, returns the size of the dimension in that band, or `NULL` if the band does not carry it.

Usage

```
rs_dim_size(
  rast = sd_missing_arg(),
  dim_name = sd_missing_arg(),
  band = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`dim_name`	(utf8): Name of the dimension to query.
`band`	(int): Band index (1-based). A null or out-of-range index returns null.

Value

(int64)

See Also

SedonaDB SQL documentation for RS_DIMSIZE

rs_dim_to_band — *Promotes a within-band dimension into separate bands.*

Description

Promotes a named non-spatial dimension into separate bands. Each index along the dimension becomes its own band with that dimension removed. This bridges the gap between the dimension model (where all indices live in one band) and the band model (where each index is a separate band accessible by number). For example, a raster with 1 band of shape `[wavelength=200, y=256, x=256]` becomes a raster with 200 bands, each of shape `[y=256, x=256]`. Standard band math functions can then operate on individual wavelength bands by index. Bands that do not contain the named dimension are passed through unchanged. If **no** band carries the named dimension, the function errors — that condition usually signals a typo'd dimension name. The spatial dimensions (`x_dim` and `y_dim`) cannot be promoted. The inverse operation is RS_BandToDim.

Usage

```
rs_dim_to_band(rast = sd_missing_arg(), dim_name = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`dim_name`	(utf8): Name of the dimension to promote (e.g., 'wavelength', 'time').

Value

(raster)

See Also

SedonaDB SQL documentation for RS_DIMTOBAND

rs_envelope — *Returns the envelope (bounding box) of a raster as a geometry.*

Description

Returns the envelope (bounding box) of a raster as a geometry.

Usage

```
rs_envelope(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_ENVELOPE

---

rs_example — *Creates a simple example raster for testing and demos.*

Description

Creates a simple example raster for testing and demos.

Usage

```
rs_example()
```

Value

(raster)

See Also

SedonaDB SQL documentation for RS_EXAMPLE

---

rs_from_gdal_raster — *Decodes a GDAL-readable raster (e.g. a GeoTIFF) from binary content into an in-database raster.*

Description

`RS_FromGDALRaster` parses raster file bytes with GDAL and returns an in-database raster with all band data materialised inline — the inverse of `RS_AsGeoTiff`, and the binary counterpart of `RS_FromPath` (which references a file on disk as an out-db raster instead). A `NULL` input yields a `NULL` raster. It is typically applied to a binary column holding encoded rasters — for example rasters produced by `RS_AsGeoTiff`, which round-trips back through `RS_FromGDALRaster`: `sql SELECT RS_Width(RS_FromGDALRas`

Usage

```
rs_from_gdal_raster(content = sd_missing_arg())
```

Arguments

`content`	(binary): Raster file bytes in any format GDAL can read (GeoTIFF, PNG, ...). A NULL input yields a NULL raster.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_FROMGDALRASTER

`rs_from_path` *Creates an out-of-database raster from a raster file path.*

Description

Loads raster metadata from the file at `path` and returns a raster whose bands reference the source file as out-db bands. This is useful when you want to work with rasters stored on disk without copying their pixel data into the raster value itself.

Usage

```
rs_from_path(path = sd_missing_arg())
```

Arguments

`path`	(string): Input string

Value

(raster)

See Also

SedonaDB SQL documentation for RS_FROMPATH

`rs_geo_reference` *Returns the georeference metadata of raster as a string in GDAL or ESRI format as commonly seen in a world file. Default is GDAL if not specified. Both formats output six lines: scalex, skewy, skewx, scaley, upperleftx, upperlefty. In GDAL format the upper-left coordinates refer to the corner of the upper-left pixel, while in ESRI format they are shifted to the center of the upper-left pixel.*

Description

Returns the georeference metadata of raster as a string in GDAL or ESRI format as commonly seen in a world file. Default is GDAL if not specified. Both formats output six lines: scalex, skewy, skewx, scaley, upperleftx, upperlefty. In GDAL format the upper-left coordinates refer to the corner of the upper-left pixel, while in ESRI format they are shifted to the center of the upper-left pixel.

Usage

```
rs_geo_reference(rast = sd_missing_arg(), format = sd_missing_arg())
```

Arguments

rast	(raster): Input raster
format	(utf8): Output format: 'GDAL' (default; alias 'PIXEL') reports the upper-left corner of the upper-left pixel; 'ESRI' (alias 'NODE') shifts the coordinates to the center of the upper-left pixel. The center shift maps through the full affine transform, so it is exact for skewed (rotated) rasters too.

Value

(utf8)

See Also

SedonaDB SQL documentation for RS_GEOREFERENCE

---

rs_geo_transform

Returns the raster's geotransform decomposed into a struct of six doubles, matching Sedona Spark's RS_GeoTransform: magnitudeI and magnitudeJ (pixel sizes along the transformed i and j axes), thetaI (rotation of the raster in radians, positive clockwise), thetaIJ (angle from the transformed i axis to the transformed j axis in radians, positive counter-clockwise), and offsetX and offsetY (the upper-left corner of the upper-left pixel). For the raw affine coefficients instead of this decomposition, use the individual accessors (RS_ScaleX, RS_SkewX, RS_UpperLeftX, ...) or RS_GeoReference.

Description

Returns the raster's geotransform decomposed into a struct of six doubles, matching Sedona Spark's RS_GeoTransform: magnitudeI and magnitudeJ (pixel sizes along the transformed i and j axes), thetaI (rotation of the raster in radians, positive clockwise), thetaIJ (angle from the transformed i axis to the transformed j axis in radians, positive counter-clockwise), and offsetX and offsetY (the upper-left corner of the upper-left pixel). For the raw affine coefficients instead of this decomposition, use the individual accessors (RS_ScaleX, RS_SkewX, RS_UpperLeftX, ...) or RS_GeoReference.

Usage

```
rs_geo_transform(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(struct)

See Also

SedonaDB SQL documentation for RS_GEOTRANSFORM

---

rs_height *Returns the height of a raster in pixels.*

Description

Returns the height of a raster in pixels.

Usage

```
rs_height(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(bigint)

See Also

SedonaDB SQL documentation for RS_HEIGHT

---

rs_intersects *Returns true if the extents of the two arguments intersect.*

Description

Returns `true` if the convex hull of the first argument intersects the second argument. Both rasters and geometries are accepted in either argument position. When two rasters are provided, their convex hulls are compared. If the arguments have different CRSes, the geometry is transformed into the raster's CRS before evaluating the predicate. For two rasters, the second raster's extent is transformed into the first raster's CRS. If the preferred transformation fails, the extent of both sides are transformed to WGS 84 as a fallback. If either argument has no CRS set, the comparison is performed directly without CRS transformation. Across CRSes a reprojected raster footprint is densified with about 10 points per edge before it is reprojected, so it follows the curve of each edge instead of chording straight through the four reprojected corners. Same-CRS comparisons are exact.

Usage

```
rs_intersects(...)
```

Arguments

`...`	Supported combinations: • rast (raster), geom (geometry) • geom (geometry), rast (raster) • rast_a (raster), rast_b (raster)

Value

(boolean)

See Also

SedonaDB SQL documentation for RS_INTERSECTS

`rs_is_empty` *Returns true when the raster has zero visible volume — some dimension has length 0, so it holds no cells. This is visible-volume emptiness, not byte-emptiness: a lazy raster whose pixels are resolved on demand carries no bytes but is not empty.*

Description

Returns true when the raster has zero visible volume — some dimension has length 0, so it holds no cells. This is visible-volume emptiness, not byte-emptiness: a lazy raster whose pixels are resolved on demand carries no bytes but is not empty.

Usage

```
rs_is_empty(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(boolean)

See Also

SedonaDB SQL documentation for RS_ISEMPTY

`rs_make_empty_raster` *Creates an in-database raster from a grid definition: a pixel count, a placement (an extent geometry, or an upper-left corner with pixel size or a full affine transform), and a number of zero-filled bands. Use it to define a target grid for RS_AsRaster or as a template for other rasters.*

Description

`RS_MakeEmptyRaster` builds a raster from nothing but a grid definition. Every band holds `width * height` zeros of `band_data_type` (default `float64`) and has no nodata value. The grid can be placed three ways: - **From an extent geometry.** The raster covers the envelope of `extent` with `width` x `height` north-up pixels: the upper-left corner is the envelope's (xmin, ymax), the pixel size is the envelope size divided by the pixel count, and the raster's CRS is the geometry's. This is the direct way to describe a grid as "columns, rows, bounding box and CRS"; pair it with `ST_MakeEnvelope`, `ST_Envelope` or `RS_Envelope`. The envelope must span a positive width and height. - **From an upper-left corner and a square cell size.** No skew and no CRS (attach one with `RS_SetSRID` or `RS_SetCRS`). - **From a full affine transform and SRID.** The six geotransform parameters and an EPSG code, `0` meaning no CRS. The last two forms take the same arguments as Sedona Spark's `RS_MakeEmptyRaster`; the extent form is a SedonaDB extension. Where Sedona Spark silently falls back to `float64` for an unrecognised `band_data_type`, SedonaDB raises an error. `num_bands` may be `0`, which yields a raster with a grid but no bands: a lightweight template that reference-taking functions can consume without carrying pixel data. The result is an in-database raster, so each band occupies `width * height * bytes-per-pixel` in memory (a `float64` band of 1000 x 1000 pixels is 8 MB). A single band is limited to 4 GiB. Null arguments yield a null raster.

Usage

```
rs_make_empty_raster(...)
```

Arguments

...	Supported combinations: • num_bands (integer), width (integer), height (integer), extent (geometry) • num_bands (integer), band_data_type (string), width (integer), height (integer), extent (geometry) • num_bands (integer), width (integer), height (integer), upper_left_x (double), upper_left_y (double), cell_size (double) • num_bands (integer), band_data_type (string), width (integer), height (integer), upper_left_x (double), upper_left_y (double), cell_size (double) • num_bands (integer), width (integer), height (integer), upper_left_x (double), upper_left_y (double), scale_x (double), scale_y (double), skew_x (double), skew_y (double), srid (integer) • num_bands (integer), band_data_type (string), width (integer), height (integer), upper_left_x (double), upper_left_y (double), scale_x (double), scale_y (double), skew_x (double), skew_y (double), srid (integer)

Value

(raster)

See Also

SedonaDB SQL documentation for RS_MAKEEMPTYRASTER

`rs_meta_data` *Returns raster metadata as a struct.*

Description

`RS_MetaData()` returns a struct containing: - `upperLeftX`, `upperLeftY`: origin of the raster geotransform - `gridWidth`, `gridHeight`: raster dimensions in pixels - `scaleX`, `scaleY`: pixel scale - `skewX`, `skewY`: geotransform skew terms - `srid`: raster SRID if available, otherwise `0` - `numSampleDimensions`: band count - `tileWidth`, `tileHeight`: GDAL block size derived tile dimensions For rasters with no bands, `tileWidth` and `tileHeight` are `0`.

Usage

```
rs_meta_data(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(struct)

See Also

SedonaDB SQL documentation for RS_METADATA

`rs_min_convex_hull` *Returns the footprint of the raster's data pixels: the polygon over the smallest pixel-aligned rectangle of the grid that holds every pixel not equal to its band's nodata value. Returns null if no pixel holds data.*

Description

`RS_MinConvexHull` is `RS_ConvexHull` with the nodata margin trimmed off. The rectangle is aligned to the raster's grid, so on a rotated or skewed raster the result follows the grid axes, like `RS_ConvexHull`. A band without a nodata value holds data everywhere, so it spans the whole grid. Only 2-D rasters are supported.

Usage

```
rs_min_convex_hull(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

rast	(raster): Input raster
band	(integer): Band index (1-based). When omitted, a pixel holds data if any band holds data there.

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_MINCONVEXHULL

---

rs_num_bands *Returns the number of bands in the raster.*

Description

Returns the number of bands in the raster.

Usage

```
rs_num_bands(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(uint32)

See Also

SedonaDB SQL documentation for RS_NUMBANDS

rs_num_dimensions *Returns the number of dimensions in a raster band.*

Description

Returns the number of dimensions in a raster band. A standard 2D raster has 2 dimensions (`y` and `x`). N-dimensional rasters (e.g., from Zarr or NetCDF sources) may have additional dimensions such as `time`, `pressure`, or `wavelength`. When the band index is omitted, all bands must agree on the number of dimensions; if they disagree, an error is returned prompting the user to specify a band index.

Usage

```
rs_num_dimensions(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`band`	(int): Band index (1-based). A null or out-of-range index returns null.

Value

(int32)

See Also

SedonaDB SQL documentation for RS_NUMDIMENSIONS

rs_pixel_as_centroid *Returns the centroid of the specified pixel as a Point geometry.*

Description

Returns the centroid of the specified pixel as a Point geometry.

Usage

```
rs_pixel_as_centroid(
  rast = sd_missing_arg(),
  colX = sd_missing_arg(),
  rowY = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`colX`	(integer): Input integer
`rowY`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_PIXELASCENTROID

`rs_pixel_as_point` *Returns the upper-left corner of the specified pixel as a Point geometry.*

Description

Returns the upper-left corner of the specified pixel as a Point geometry.

Usage

```
rs_pixel_as_point(
  rast = sd_missing_arg(),
  colX = sd_missing_arg(),
  rowY = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`colX`	(integer): Input integer
`rowY`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_PIXELASPOINT

`rs_pixel_as_polygon` *Returns the bounding polygon of the specified pixel.*

Description

Returns the bounding polygon of the specified pixel.

Usage

```
rs_pixel_as_polygon(
  rast = sd_missing_arg(),
  colX = sd_missing_arg(),
  rowY = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`colX`	(integer): Input integer
`rowY`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_PIXELASPOLYGON

`rs_polygonize` *Converts a raster band into polygons grouped by connected pixel value regions.*

Description

Converts the specified raster band into vector polygons, returning one item per connected region of pixels that shares the same value. Each returned list item contains: - `geom`: the polygon geometry encoded as WKB - `value`: the pixel value represented by that polygon

Usage

```
rs_polygonize(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`band`	(integer): Input integer

Value

(list)

See Also

SedonaDB SQL documentation for RS_POLYGONIZE

`rs_raster_to_world_coord`

Converts raster pixel coordinates to world coordinates as a point.

Description

Pixel coordinates are 1-based, as in PostGIS `ST_RasterToWorldCoord`: `(1, 1)` is the upper-left pixel, and the result is the world coordinate of that pixel's upper-left corner. Coordinates outside the grid, including `0`, extrapolate along the geotransform.

Usage

```
rs_raster_to_world_coord(
  rast = sd_missing_arg(),
  x = sd_missing_arg(),
  y = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`x`	(integer): Input integer
`y`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_RASTERTOWORLDCOORD

`rs_raster_to_world_coord_x`
Converts raster pixel coordinates to world X coordinate.

Description

Pixel coordinates are 1-based, as in PostGIS `ST_RasterToWorldCoord`: `(1, 1)` is the upper-left pixel, and the result is the world coordinate of that pixel's upper-left corner. Coordinates outside the grid, including `0`, extrapolate along the geotransform.

Usage

```
rs_raster_to_world_coord_x(
  rast = sd_missing_arg(),
  x = sd_missing_arg(),
  y = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`x`	(integer): Input integer
`y`	(integer): Input integer

Value

(double)

See Also

SedonaDB SQL documentation for RS_RASTERTOWORLDCOORDX

`rs_raster_to_world_coord_y`
Converts raster pixel coordinates to world Y coordinate.

Description

Pixel coordinates are 1-based, as in PostGIS `ST_RasterToWorldCoord`: `(1, 1)` is the upper-left pixel, and the result is the world coordinate of that pixel's upper-left corner. Coordinates outside the grid, including `0`, extrapolate along the geotransform.

Usage

```
rs_raster_to_world_coord_y(
  rast = sd_missing_arg(),
  x = sd_missing_arg(),
  y = sd_missing_arg()
)
```

Arguments

rast	(raster): Input raster
x	(integer): Input integer
y	(integer): Input integer

Value

(double)

See Also

SedonaDB SQL documentation for RS_RASTERTOWORLDCOORDY

---

rs_replace_band_no_data_value

Moves a band's nodata value to a new value, carrying the nodata pixels with it: every pixel that reads as nodata is rewritten to the new value, which then becomes the band's nodata, so the same pixels read as nodata before and after.

Description

RS_SetBandNoDataValue changes only the declared nodata value, so pixels that held the old value become ordinary data. RS_ReplaceBandNoDataValue also rewrites those pixels, so they keep reading as nodata. A pixel reads as nodata by the same rules sampling uses: exactly for integer bands, and numerically for float bands, so -0.0 matches a 0.0 nodata and any NaN matches a NaN nodata. Other bands are unchanged. The band must already have a nodata value; otherwise there is nothing to replace and the call is an error. A null raster, band or nodata value gives a null raster. Only 2-D rasters are supported.

Usage

```
rs_replace_band_no_data_value(
  rast = sd_missing_arg(),
  band = sd_missing_arg(),
  nodata = sd_missing_arg()
)
```

Arguments

rast	(raster): Input raster
band	(int): Band index (1-based).
nodata	(float64): The new nodata value. It must fit the band's data type exactly.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_REPLACEBANDNODATAVALUE

---

rs_reproject_match *Reprojects a raster onto a reference raster's CRS, grid, and extent.*

Description

RS_ReprojectMatch reprojects raster onto the grid of reference: the output always has the **same** CRS, extent, resolution, and dimensions as reference, in the spirit of rioxarray's reproject_match. The reference contributes only its grid — its pixel values are never read. The input's band count and order, per-band data type, and nodata are preserved. Pixel values are recomputed by GDAL's warp using the chosen algorithm (nearest neighbour by default). Cells of the reference grid that the reprojected input does not cover are filled with the input band's nodata value (or zero when a band has none). Both rasters must agree on whether they carry a CRS: reprojecting between an unknown CRS and a real one is undefined, so it is an error for exactly one side to have a CRS. When neither has one, the operation is a same-space regrid onto the reference grid. Int64 and UInt64 input rasters are **not supported**: GDAL's warp routes 64-bit integer pixels through a floating working type (a double for nearest/interpolation, a 32-bit float for mode on GDAL < 3.13), so no resampling method can represent them exactly. Cast to a supported type (for example Int32 or Float64) first. ::: callout-note This reads the entire input raster into memory. :::

Usage

```
rs_reproject_match(
  raster = sd_missing_arg(),
  reference = sd_missing_arg(),
  algorithm = sd_missing_arg()
)
```

Arguments

raster	(raster): Input raster
reference	(raster): Input raster
algorithm	(string): Resampling algorithm (case-insensitive): NearestNeighbor (the default), Bilinear, Cubic (alias Bicubic), CubicSpline, Lanczos, Average, or Mode.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_REPROJECTMATCH

`rs_resample` *Resamples a raster onto a new pixel grid — a new width/height, a new pixel size, and/or a snapped origin — in the same CRS.*

Description

`RS_Resample` changes a raster's pixel grid while keeping its band count and order and per-band nodata. Pixel values are recomputed by GDAL with the chosen `algorithm` (nearest neighbour by default). The output stays in the input's CRS — `RS_Resample` never reprojects (use `RS_ReprojectMatch` for that). `RS_Resample` has three positional overloads: - `RS_Resample(raster, width_or_scale, height_or_scale, use_scale, algorithm)` — when `use_scale` is false, `width_or_scale`/`height_or_scale` are the output pixel dimensions (a fractional value is an error) and the world extent is preserved. When `use_scale` is true they are the target pixel size (`scaleX`, `scaleY`, with `scaleY` negative for a north-up raster); the pixel size is kept exact and the extent grows by up to one pixel so it tiles into whole pixels. Cells outside the source footprint become nodata. - `RS_Resample(raster, width_or_scale, height_or_scale, grid_x, grid_y, use_scale, algorithm)` — additionally snaps the output origin to the grid anchored at `(grid_x, grid_y)`, expanding outward so the original extent stays covered. - `RS_Resample(raster, reference_raster, use_scale, algorithm)` — takes the target dimensions (or pixel size, when `use_scale`) and the origin grid from `reference_raster`, which must share the input's CRS (it is an error otherwise). `algorithm` (case-insensitive) is one of `NearestNeighbor` (the default), `Bilinear`, `Cubic` (alias `Bicubic`), `CubicSpline`, `Lanczos`, `Average`, or `Mode`. Dimensions and pixel size use the rows-then-columns intuition of the raster (`height_or_scale`/`scaleY` for rows, `width_or_scale`/`scaleX` for columns). In the extent-preserving dimension form, a skewed or rotated raster is re-gridded with its footprint preserved (both the scale and skew terms of each axis scale by the same factor). A scale change or origin snap on a skewed raster is not supported and errors. For an N-D raster (bands with extra dimensions such as `time`), the resample is a 2-D `(y, x)` operation broadcast across every non-spatial plane, so the non-spatial dimensions are preserved and only the `(y, x)` extent changes. ::: callout-note Extent-growing scale changes and origin snaps read the entire raster into memory. :::

Usage

```
rs_resample(...)
```

Arguments

`...`	Supported combinations: • raster (raster), reference_raster (raster), use_scale (boolean), algorithm (string) • raster (raster), width_or_scale (double), height_or_scale (double), use_scale (boolean), algorithm (string)

- raster (raster), width_or_scale (double), height_or_scale (double), grid_x (double), grid_y (double), use_scale (boolean), algorithm (string)

Value

(raster)

See Also

SedonaDB SQL documentation for RS_RESAMPLE

---

rs_rotation *Returns the raster rotation in radians based on skew parameters.*

Description

Returns the raster rotation in radians based on skew parameters.

Usage

```
rs_rotation(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_ROTATION

---

rs_scale_x *Returns the pixel width (scale X) of a raster.*

Description

Returns the pixel width (scale X) of a raster.

Usage

```
rs_scale_x(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_SCALEX

`rs_scale_y` *Returns the pixel height (scale Y) of a raster.*

Description

Returns the pixel height (scale Y) of a raster.

Usage

```
rs_scale_y(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_SCALEY

`rs_set_band_no_data_value` *Sets the nodata value of the specified band, packing the value into the band's native data type. A null raster or band yields a null raster; a null nodata value instead clears the addressed band's nodata. This is metadata-only: raster cell values are not modified.*

Description

Sets the nodata value of the specified band, packing the value into the band's native data type. A null raster or band yields a null raster; a null nodata value instead clears the addressed band's nodata. This is metadata-only: raster cell values are not modified.

Usage

```
rs_set_band_no_data_value(...)
```

Arguments

`...`	Supported combinations: • rast (raster), nodata (float64) • rast (raster), band (int), nodata (float64)

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SETBANDNODATAVALUE

`rs_set_crs` *Sets the Coordinate Reference System (CRS) for a raster.*

Description

Sets a CRS on a raster. This is metadata-only: raster cell values and geotransform are not transformed. See Coordinate Reference Systems for how SedonaDB represents and compares CRS.

Usage

```
rs_set_crs(rast = sd_missing_arg(), target_crs = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`target_crs`	(string): Input string

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SETCRS

`rs_set_geo_reference` *Sets the affine geotransform of a raster from a world-file georeference string of six whitespace-separated numbers in the order scalex, skewy, skewx, scaley, upperleftx, upperlefty — the same order RS_GeoReference emits. The format is 'GDAL' (default) or 'ESRI'; in GDAL the upper-left coordinates refer to the corner of the upper-left pixel, while in ESRI they refer to its center. The transform can also be given as six numeric arguments instead of a string, always as GDAL/corner coordinates. This is metadata-only: raster cell values are not resampled.*

Description

Sets the affine geotransform of a raster from a world-file georeference string of six whitespace-separated numbers in the order scalex, skewy, skewx, scaley, upperleftx, upperlefty — the same order RS_GeoReference emits. The format is 'GDAL' (default) or 'ESRI'; in GDAL the upper-left coordinates refer to the corner of the upper-left pixel, while in ESRI they refer to its center. The transform can also be given as six numeric arguments instead of a string, always as GDAL/corner coordinates. This is metadata-only: raster cell values are not resampled.

Usage

```
rs_set_geo_reference(...)
```

Arguments

`...`	Supported combinations: • rast (raster), georeference (string) • rast (raster), georeference (string), format (string) • rast (raster), upperleftx (numeric), upperlefty (numeric), scalex (numeric), scaley (numeric), skewx (numeric), skewy (numeric)

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SETGEOREFERENCE

`rs_set_srid` *Sets the SRID (spatial reference identifier) for a raster.*

Description

Sets the SRID of a raster. This is metadata-only: raster cell values and geotransform are not transformed. See Coordinate Reference Systems for how SedonaDB represents and compares CRS.

Usage

```
rs_set_srid(rast = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`srid`	(integer): Input integer

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SETSRID

`rs_set_value` *Returns the raster with one pixel of a band set to a new value.*

Description

`col_x` and `row_y` count from 1, as in PostGIS and Sedona Spark: `(1, 1)` is the upper-left pixel. The pixel is overwritten even if it holds the band's nodata value, and the band's nodata value is unchanged. The value is stored in the band's pixel type: an integer band truncates it toward zero (3.9 is stored as 3) and a float32 band rounds it to the nearest float32. A value outside an integer band's range (for a 64-bit integer band, beyond ±2^53, the largest integers a double holds exactly), NaN, a pixel outside the grid, or a band the raster does not have is an error. Only 2-D rasters are supported.

Usage

```
rs_set_value(...)
```

Arguments

...	Supported combinations: • rast (raster), col_x (integer), row_y (integer), new_value (double) • rast (raster), band (integer), col_x (integer), row_y (integer), new_value (double)

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SETVALUE

---

rs_shape *Returns the shape (size of each dimension) of a raster band.*

Description

Returns the shape of a raster band as a list of dimension sizes. The entries correspond to the dimension names returned by RS_DimNames. For a standard 2D raster this returns `[height, width]`. For an N-dimensional raster with a time dimension it might return `[12, 256, 256]` meaning 12 time steps at 256x256 spatial resolution. When the band index is omitted, all bands must agree on their shape; if they disagree, an error is returned prompting the user to specify a band index.

Usage

```
rs_shape(rast = sd_missing_arg(), band = sd_missing_arg())
```

Arguments

rast	(raster): Input raster
band	(int): Band index (1-based). A null or out-of-range index returns null.

Value

(list)

See Also

SedonaDB SQL documentation for RS_SHAPE

rs_skew_x *Returns the X skew (rotation) parameter of a raster.*

Description

Returns the X skew (rotation) parameter of a raster.

Usage

```
rs_skew_x(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_SKEWX

rs_skew_y *Returns the Y skew (rotation) parameter of a raster.*

Description

Returns the Y skew (rotation) parameter of a raster.

Usage

```
rs_skew_y(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_SKEWY

`rs_slice` *Selects a single index along a dimension, removing that dimension from the output.*

Description

Extracts a single slice along a named non-spatial dimension, removing that dimension from every band that carries it in the output raster. For example, slicing a raster with shape `[time=12, y=256, x=256]` on `'time'` at index 5 produces a 2D raster with shape `[y=256, x=256]` containing the data at time step 5. The spatial dimensions (`x_dim` and `y_dim`) cannot be sliced. ### Heterogeneous rasters Bands within a raster can have different dimensionality. Bands that do not carry the named dimension are emitted unchanged ("pass-through") rather than producing an error. This matches the convention used by `RS_DimToBand` and `xarray`'s `ds.isel({dim: i})`. For example, with a raster containing an elevation band (`[y, x]`) and a temperature band (`[time, y, x]`), `RS_Slice(raster, 'time', 5)` slices the temperature band and passes the elevation band through unchanged. If **no** band carries the named dimension, the function errors — that condition usually signals a typo'd dimension name. Returns an error if the index is out of range for any band that does carry the dimension. See also RS_SliceRange to extract a range of indices while keeping the dimension.

Usage

```
rs_slice(
  rast = sd_missing_arg(),
  dim_name = sd_missing_arg(),
  index = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`dim_name`	(utf8): Name of the dimension to slice (e.g., 'time').
`index`	(int): Zero-based index along the dimension.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SLICE

`rs_slice_range` *Narrows a dimension to a half-open range, keeping the dimension with reduced size.*

Description

Narrows a named non-spatial dimension to the half-open range `[start, end)` on every band that carries it, keeping the dimension in the output with reduced size. For example, narrowing a raster with shape `[time=12, y=256, x=256]` on `'time'` with range `[2, 7)` produces a raster with shape `[time=5, y=256, x=256]`. The spatial dimensions (`x_dim` and `y_dim`) cannot be narrowed. ### Heterogeneous rasters Bands within a raster can have different dimensionality. Bands that do not carry the named dimension are emitted unchanged ("pass-through") rather than producing an error. This matches the convention used by `RS_Slice`, `RS_DimToBand`, and `xarray`'s `ds.isel`. If **no** band carries the named dimension, the function errors — that condition usually signals a typo'd dimension name. Returns an error if `start >= end` or the range is out of bounds for any band that does carry the dimension. See also RS_Slice to extract a single index and remove the dimension entirely.

Usage

```
rs_slice_range(
  rast = sd_missing_arg(),
  dim_name = sd_missing_arg(),
  start = sd_missing_arg(),
  end = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`dim_name`	(utf8): Name of the dimension to narrow (e.g., 'time').
`start`	(int): Start index (inclusive, zero-based).
`end`	(int): End index (exclusive).

Value

(raster)

See Also

SedonaDB SQL documentation for RS_SLICERANGE

rs_srid — *Returns the SRID of a raster.*

Description

Returns the SRID of a raster.

Usage

```
rs_srid(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(integer)

See Also

SedonaDB SQL documentation for RS_SRID

rs_stack — *Returns one raster holding the bands of any number of rasters, in argument order.*

Description

Sedona Spark 1.9 calls this function RS_Union. The result holds every band of the first raster, then every band of the second, and so on. Each band keeps its own pixel type, nodata value and name, so rasters of different pixel types can be stacked. Every raster must be on the first raster's grid (the same width, height, geotransform and CRS); a raster on any other grid is an error. A band's spatial dimensions take the first raster's names, matched by role, so a lat/lon raster stacked onto a y/x raster gives y/x bands; a band with another dimension already named y or x is an error. A NULL raster gives a NULL result. No pixel is read: bands are carried over as they are, so stacking rasters that are still out-of-database does not load them.

Usage

```
rs_stack(rast_a = sd_missing_arg(), rasters = sd_missing_arg())
```

Arguments

rast_a	(raster): Input raster
rasters	(raster): One or more further rasters (any number), on the same grid as the first: the same width, height, geotransform and CRS.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_STACK

`rs_stack_aggr` *Aggregates a column of rasters into one raster holding all of their bands, ordered by an index column.*

Description

An aggregate function: the aggregate form of RS_Stack. Sedona Spark 1.9 calls it `RS_Union_Aggr`. The result holds every band of the raster with the lowest index, then every band of the next, and so on, under the georeference of the lowest-index raster. The index decides the order because an aggregate sees its rows in no particular order. Indexes must be distinct and evenly spaced: a gap usually means a raster is missing, which would silently shift every later band, so it is an error. As with RS_Stack, every raster must be on one grid (the same width, height, geotransform and CRS), each band keeps its own pixel type, nodata value and name, and a band's spatial dimensions take the lowest-index raster's names. Rows with a NULL raster or a NULL index are skipped; a group with no other rows gives NULL. No pixel is read: bands are carried over as they are, so stacking rasters that are still out-of-database does not load them.

Usage

```
rs_stack_aggr(rast = sd_missing_arg(), index = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster
`index`	(integer): The position of this row's raster in the result. Indexes must be distinct and evenly spaced, such as 1, 2, 3 or 0, 10, 20.

Value

(raster)

See Also

SedonaDB SQL documentation for RS_STACK_AGGR

`rs_summary_stats` *Returns one summary statistic of a band's pixel values as a double.*

Description

Over no pixels (for example, a band that is entirely nodata), `count` and `sum` are 0 and the other statistics are NaN; over one pixel, `stddev` is 0. A NaN pixel is left out as nodata when the band's nodata value is NaN. Otherwise it is data: it makes `sum` and `mean` NaN, and `stddev` too unless it is the only pixel, while `min` and `max` skip it unless every pixel is NaN. An unknown `stat_type`, a band the raster does not have, or a 64-bit integer pixel beyond 2^53 in magnitude (which a double cannot hold exactly) is an error. Only 2-D rasters are supported.

Usage

```
rs_summary_stats(
  rast = sd_missing_arg(),
  stat_type = sd_missing_arg(),
  band = sd_missing_arg(),
  exclude_nodata = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`stat_type`	(string): Input string
`band`	(integer): Input integer
`exclude_nodata`	(boolean): Whether to leave pixels holding the band's nodata value out of the statistic. Defaults to true.

Value

(double)

See Also

SedonaDB SQL documentation for RS_SUMMARYSTATS

`rs_summary_stats_all` *Returns every summary statistic of a band's pixel values as a struct of six doubles: count, sum, mean, stddev, min and max.*

Description

The fields are the statistics `RS_SummaryStats` returns one at a time, with the same values: `stddev` is the population standard deviation, and over no pixels (for example, a band that is entirely nodata) `count` and `sum` are 0 and the other fields are NaN. Computing them together scans the band three times, where six `RS_SummaryStats` calls would scan it more often. A band the raster does not have, or a 64-bit integer pixel beyond 2^53 in magnitude (which a double cannot hold exactly), is an error. Only 2-D rasters are supported.

Usage

```
rs_summary_stats_all(
  rast = sd_missing_arg(),
  band = sd_missing_arg(),
  exclude_nodata = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`band`	(integer): Input integer
`exclude_nodata`	(boolean): Whether to leave pixels holding the band's nodata value out of the statistics. Defaults to true.

Value

(struct)

See Also

SedonaDB SQL documentation for RS_SUMMARYSTATSALL

`rs_tile` *Returns an array of rasters resulting from the split of the input raster into a grid of fixed-size tiles, one list item per tile with its grid position.*

Description

RS_Tile cuts a raster into a grid of width x height tiles, returning a list with one item per tile that you UNNEST to get one row per tile. When no band argument is given, every band is included. band_indices selects an ordered subset of bands. Band indices are 1-based. Each returned list item is a struct with: - x: the tile's grid column (0-based), counting from the left - y: the tile's grid row (0-based), counting from the top - tile: the tile raster, with a geotransform shifted to the tile's own origin The grid is ceil(width / tile_width) columns by ceil(height / tile_height) rows. When the raster does not divide evenly, the last column and row of tiles are partial: with pad_with_no_data = false the smaller edge tile is emitted as-is, and with pad_with_no_data = true those tiles are padded to the full tile size with a nodata fill. When the band(s) have no nodata value, no_data_value supplies one. ::: callout-note RS_Tile **copies the raster**: every tile carries its own pixels, so the returned list materializes a full copy of the raster spread across the tiles and held in memory as one list value. A small tile size produces many tiles (a 1 x 1 tile size yields one tile per pixel). :::

Usage

```
rs_tile(...)
```

Arguments

...	Supported combinations: • rast (raster), width (integer), height (integer), pad_with_no_data (boolean), no_data_value (double) • rast (raster), band_indices (array), width (integer), height (integer), pad_with_no_data (boolean), no_data_value (double)

Value

(list)

See Also

SedonaDB SQL documentation for RS_TILE

---

rs_upper_left_x *Returns the upper-left X coordinate of a raster.*

Description

Returns the upper-left X coordinate of a raster.

Usage

```
rs_upper_left_x(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_UPPERLEFTX

`rs_upper_left_y` *Returns the upper-left Y coordinate of a raster.*

Description

Returns the upper-left Y coordinate of a raster.

Usage

```
rs_upper_left_y(rast = sd_missing_arg())
```

Arguments

`rast`	(raster): Input raster

Value

(double)

See Also

SedonaDB SQL documentation for RS_UPPERLEFTY

`rs_value` *Returns the value of a single raster pixel as a double, selected by a point geometry. Returns null if the location is outside the raster or the pixel holds the band's nodata value.*

Description

`RS_Value` samples one pixel of a raster. The location is given as a point geometry — the value of the pixel that contains the point is returned, with no interpolation. The band defaults to 1 when omitted, but only for a single-band raster; on a multiband raster the band must be given explicitly. The pixel can instead be addressed by its grid coordinate. Grid coordinates are 1-based, as in PostGIS `ST_Value`: `(1, 1)` is the upper-left pixel, and a column or row of `0` is outside the raster. The result is `NULL` when the point falls outside the raster, or when the sampled pixel equals the band's nodata value. Only 2-D rasters are supported.

Usage

```
rs_value(...)
```

Arguments

`...`	Supported combinations: • rast (raster), point (geometry) • rast (raster), point (geometry), band (int) • rast (raster), colX (int), rowY (int), band (int)

Value

(float64)

See Also

SedonaDB SQL documentation for RS_VALUE

`rs_values` *Returns the pixel values under each point of a MultiPoint geometry as a list of doubles, one element per point. An element is null where the point falls outside the raster or the pixel holds the band's nodata value.*

Description

RS_Values is the plural form of RS_Value: it samples one pixel per point of a MultiPoint and returns the values as a list of doubles, in the same order as the input points. A single Point is accepted as well and produces a one-element list. The band defaults to 1 when omitted, but only for a single-band raster; on a multiband raster the band is ambiguous and must be given explicitly. Each list element is NULL when its point falls outside the raster or the sampled pixel equals the band's nodata value; the whole result is NULL when the raster, geometry, or band is NULL. An empty MultiPoint yields an empty list. Only 2-D rasters are supported.

Usage

```
rs_values(
  rast = sd_missing_arg(),
  points = sd_missing_arg(),
  band = sd_missing_arg()
)
```

Arguments

rast	(raster): Input raster
points	(geometry): Input geometry
band	(int): Band index (1-based). When omitted, defaults to band 1, but only for a single-band raster; sampling an unspecified band of a multiband raster is ambiguous and raises an error.

Value

(list)

See Also

SedonaDB SQL documentation for RS_VALUES

---

rs_width *Returns the width of a raster in pixels.*

Description

Returns the width of a raster in pixels.

Usage

```
rs_width(rast = sd_missing_arg())
```

Arguments

rast	(raster): Input raster

Value

(bigint)

See Also

SedonaDB SQL documentation for RS_WIDTH

`rs_within` *Returns true if the first argument's extent is within the second.*

Description

Returns `true` if the convex hull of the first argument is completely within the second argument. This is the inverse of `RS_Contains`: `RS_Within(A, B)` is equivalent to `RS_Contains(B, A)`. Both rasters and geometries are accepted in either argument position. When two rasters are provided, their convex hulls are compared. If the arguments have different CRSes, the geometry is transformed into the raster's CRS before evaluating the predicate. For two rasters, the second raster's extent is transformed into the first raster's CRS. If the preferred transformation fails, the extent of both sides are transformed to WGS 84 as a fallback. If either argument has no CRS set, the comparison is performed directly without CRS transformation.

Usage

```
rs_within(...)
```

Arguments

`...`	Supported combinations: • rast (raster), geom (geometry) • geom (geometry), rast (raster) • rast_a (raster), rast_b (raster)

Value

(boolean)

See Also

SedonaDB SQL documentation for RS_WITHIN

rs_world_to_raster_coord

Converts world coordinates to raster coordinates as a point.

Description

The result is a 1-based pixel coordinate, as in PostGIS `ST_WorldToRasterCoord`: a world coordinate inside the upper-left pixel maps to `(1, 1)`. A world coordinate outside the raster extrapolates along the geotransform, so it can map to `0` or a negative coordinate.

Usage

```
rs_world_to_raster_coord(...)
```

Arguments

`...`	Supported combinations: • rast (raster), x (double), y (double) • rast (raster), point (geometry)

Value

(geometry)

See Also

SedonaDB SQL documentation for RS_WORLDTORASTERCOORD

rs_world_to_raster_coord_x

Converts world coordinates to raster X coordinate.

Description

The result is a 1-based pixel coordinate, as in PostGIS `ST_WorldToRasterCoord`: a world coordinate inside the upper-left pixel maps to `(1, 1)`. A world coordinate outside the raster extrapolates along the geotransform, so it can map to `0` or a negative coordinate.

Usage

```
rs_world_to_raster_coord_x(...)
```

Arguments

`...`	Supported combinations: • rast (raster), x (double), y (double) • rast (raster), point (geometry)

Value

(integer)

See Also

SedonaDB SQL documentation for RS_WORLDTORASTERCOORDX

`rs_world_to_raster_coord_y`

Converts world coordinates to raster Y coordinate.

Description

The result is a 1-based pixel coordinate, as in PostGIS `ST_WorldToRasterCoord`: a world coordinate inside the upper-left pixel maps to `(1, 1)`. A world coordinate outside the raster extrapolates along the geotransform, so it can map to `0` or a negative coordinate.

Usage

```
rs_world_to_raster_coord_y(...)
```

Arguments

`...`	Supported combinations: • rast (raster), x (double), y (double) • rast (raster), point (geometry)

Value

(integer)

See Also

SedonaDB SQL documentation for RS_WORLDTORASTERCOORDY

`rs_zonal_stats` *Computes a single summary statistic of the raster pixels covered by a region of interest geometry.*

Description

`RS_ZonalStats` returns one summary statistic of the pixels of a single band that a region of interest (ROI) geometry covers. A pixel is included when its center falls inside the roi (or, with `all_touched`, when the roi touches it at all). By default the band's nodata pixels are excluded. The statistic is one of `count`, `sum`, `mean`, `median`, `mode`, `stddev`, `variance`, `min`, or `max`. `count` is returned as a whole number; all other statistics are floating point. Variance and standard deviation are the sample (n-1) values, and `mode` breaks ties toward the larger value. When the ROI overlaps the raster but selects no pixel, `count` is 0 and every other statistic is NULL. When the ROI does not intersect the raster at all, the result is NULL under the default `lenient` behavior, or an error when `lenient` is set to false. The `band`, `all_touched`, `exclude_no_data`, and `lenient` arguments are added one at a time by the wider overloads; `all_touched` defaults to false, `exclude_no_data` to true, and `lenient` to true. The band-less overload does not default to band 1 on a multiband raster: naming the band is required there. This function operates on 2-D `(y, x)` bands; computing a statistic per non-spatial plane of an N-D band is not supported. Use `RS_ZonalStatsAll` to compute every statistic at once.

Usage

```
rs_zonal_stats(...)
```

Arguments

`...`	Supported combinations: • rast (raster), roi (geometry), stat_type (string) • rast (raster), roi (geometry), band (integer), stat_type (string) • rast (raster), roi (geometry), band (integer), stat_type (string), all_touched (boolean) • rast (raster), roi (geometry), band (integer), stat_type (string), all_touched (boolean), exclude_no_data (boolean) • rast (raster), roi (geometry), band (integer), stat_type (string), all_touched (boolean), exclude_no_data (boolean), lenient (boolean)

Value

(double)

See Also

SedonaDB SQL documentation for RS_ZONALSTATS

`rs_zonal_stats_all` *Computes every summary statistic of the raster pixels covered by a region of interest geometry and returns them as a struct.*

Description

`RS_ZonalStatsAll` returns every summary statistic of the pixels of a single band that a roi geometry covers, as a struct with fields `count`, `sum`, `mean`, `median`, `mode`, `stddev`, `variance`, `min`, and `max`. A pixel is included when its center falls inside the roi (or, with `all_touched`, when the roi touches it at all), and the band's nodata pixels are excluded by default. `count` is a whole number (a 64-bit integer); every other field is floating point. Variance and standard deviation are the sample (n-1) values, and `mode` breaks ties toward the larger value. When the roi overlaps the raster but selects no pixel, `count` is 0 and every other field is NULL. When the roi does not intersect the raster at all, the whole struct is NULL under the default `lenient` behavior, or the call raises an error when `lenient` is set to false. The overloads are the same ladder as `RS_ZonalStats` without `stat_type`. The `band`, `all_touched`, `exclude_no_data`, and `lenient` arguments are added one at a time by the wider overloads; `all_touched` defaults to false, `exclude_no_data` to true, and `lenient` to true. The band-less overload does not default to band 1 on a multiband raster: naming the band is required there. This function operates on 2-D `(y, x)` bands; computing statistics per non-spatial plane of an N-D band is not supported. Use `RS_ZonalStats` to compute a single statistic.

Usage

```
rs_zonal_stats_all(
  rast = sd_missing_arg(),
  roi = sd_missing_arg(),
  band = sd_missing_arg(),
  all_touched = sd_missing_arg(),
  exclude_no_data = sd_missing_arg(),
  lenient = sd_missing_arg()
)
```

Arguments

`rast`	(raster): Input raster
`roi`	(geometry): Region-of-interest geometry. Reprojected into the raster's CRS when both carry one; it is an error for exactly one side to have a CRS.
`band`	(integer): 1-based band index to compute over. Required for a multiband raster; a single-band raster may omit it via the band-less overload.
`all_touched`	(boolean): If true, include every pixel the roi touches; otherwise only pixels whose center falls inside it. Defaults to false.
`exclude_no_data`	(boolean): If true (the default), skip pixels equal to the band's nodata value.
`lenient`	(boolean): If true (the default), return NULL when the roi does not intersect the raster; if false, raise an error.

Value

(struct)

See Also

SedonaDB SQL documentation for RS_ZONALSTATSALL

`sd_affine` *Applies an affine transformation to a geometry.*

Description

Applies the affine transformation with the given coefficients to every coordinate, as in PostGIS: - 2D: `x' = a*x + b*y + xOff`, `y' = d*x + e*y + yOff` - 3D: `x' = a*x + b*y + c*z + xOff`, `y' = d*x + e*y + f*z + yOff`, `z' = g*x + h*y + i*z + zOff` Z and M values a 2D transformation does not use are kept.

Usage

```
sd_affine(...)
```

Arguments

...	Supported combinations: • geom (geometry), a (double), b (double), d (double), e (double), xOff (double), yOff (double) • 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)

See Also

SedonaDB SQL documentation for ST_AFFINE

sd_analyze_agg *Computes the statistics of geometries for the input geometry or geography.*

Description

`ST_Analyze_Agg()` provides a high-level summary of its input geometries. The fields of its struct return type are: - `count`: Number of input geometries - `minx`, `miny`, `maxx`, `maxy`: Minimum bounding rectangle (envelope). For geography, this represents geographical bounds such that minx may be greater than maxx where this represents more compact bounds (e.g., a small feature crossing the antimeridian). - `mean_size_in_bytes` - `mean_points_per_geometry` - `puntal_count`: Number of point or multipoint geometries - `lineal_count`: Number of line or multilinestring geometries - `polygonal_count`: Number of polygon or multipolygon geometries - `geometrycollection_count`: Number of geometrycollection geometries - `mean_envelope_width` - `mean_envelope_height` - `mean_envelope_area`

Usage

```
sd_analyze_agg(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(struct)

See Also

SedonaDB SQL documentation for ST_ANALYZE_AGG

sd_area *Returns the area of a geometry or geography.*

Description

`ST_Area()` Returns the area of a polygon or multi-polygon. For geometry, returns the area in the units of the coordinate reference system (squared); for geography, returns the area in square meters, calculated on a spherical Earth model.

Usage

```
sd_area(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_AREA

`sd_as_binary` *Converts a geometry or geography to Well-Known Binary (WKB) format.*

Description

`ST_AsBinary()` Returns the Well-Known Binary representation of a geometry or geography. This function also has the alias ST_AsWKB. `ST_AsBinary()` may preserve the underlying Arrow storage type of the input geometry. For example, when the input geometry is backed by Arrow `BinaryView` storage, the WKB output may also use `BinaryView` storage. If a downstream consumer requires simpler Arrow storage such as `Binary`, wrap the result with `sd_simplifystorage()`.

Usage

```
sd_as_binary(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(binary)

See Also

SedonaDB SQL documentation for ST_ASBINARY

sd_as_ewkb *Returns the EWKB representation of a geometry or geography.*

Description

EWKB extends WKB to include SRID information in the binary header and is useful for PostGIS compatibility. This function preserves item-level CRSes or will repeat a type-level CRS for all elements if present.

Usage

```
sd_as_ewkb(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(binary)

See Also

SedonaDB SQL documentation for ST_ASEWKB

sd_as_geo_json *Returns the GeoJSON representation of a geometry.*

Description

Returns the GeoJSON representation of a geometry.

Usage

```
sd_as_geo_json(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(string)

See Also

SedonaDB SQL documentation for ST_ASGEOJSON

sd_as_text — *Returns the Well-Known Text string representation of a geometry or geography.*

Description

Alias: `ST_AsWKT`.

Usage

```
sd_as_text(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(string)

See Also

SedonaDB SQL documentation for ST_ASTEXT

sd_azimuth — *Returns the azimuth between two points in radians, or NULL if not available.*

Description

Returns the azimuth (angle) from the first point to the second point, measured in **radians** clockwise from north. Returns NULL if either input is not a point.

Usage

```
sd_azimuth(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_AZIMUTH

`sd_boundary` *Returns the closure of the combinatorial boundary of this geometry or geography.*

Description

For a polygon, the boundary is the exterior and interior rings; for a linestring it is the endpoints.

Usage

```
sd_boundary(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_BOUNDARY

`sd_buffer` *Computes a geometry or geography that represents all points whose distance from the input is less than or equal to a specified distance.*

Description

For geography, the distance is specified in meters. The buffer is computed on the sphere, producing a result that considers polar regions and the antimeridian. Negative distances are supported for polygon inputs to shrink the polygon.

Usage

```
sd_buffer(...)
```

Arguments

...	Supported combinations: • geom (geometry), distance (float64) • geom (geometry), distance (float64), params (string) • geom (geography), distance (float64) • geom (geography), distance (float64), num_quad_segs (integer) • geom (geography), distance (float64), params (string)

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_BUFFER

---

sd_build_area *Returns a geometry that encloses the area formed by the given linework.*

Description

Treats the input geometry's edges as linework and assembles them into a polygon or multipolygon. Open rings are closed automatically before assembly. Empty input returns an empty polygon, and polygon or multipolygon input is returned unchanged. If no closed ring can be formed, returns NULL.

Usage

```
sd_build_area(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_BUILDAREA

sd_cell_id_from_point *Returns the S2 cell ID containing the given point geography at the maximum S2 level (30).*

Description

Returns the 64-bit S2 cell ID at level 30 (the finest resolution) that contains the given point geography. Returns NULL for empty point geometries. This is useful for spatial indexing and efficient spatial queries.

Usage

```
sd_cell_id_from_point(geom = sd_missing_arg())
```

Arguments

geom	(geography): Input geography

Value

(int64)

See Also

SedonaDB SQL documentation for S2_CELLIDFROMPOINT

sd_centroid *Returns the centroid of a geometry or geography.*

Description

For geography, the centroid is calculated using spherical geometry.

Usage

```
sd_centroid(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_CENTROID

sd_closest_point *Returns the 2-dimensional point on geom1 that is closest to geom2.*

Description

Returns the 2-dimensional point on geom1 that is closest to geom2.

Usage

```
sd_closest_point(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_CLOSESTPOINT

sd_collect_agg *Combines multiple geometries or geographies from a set of rows into a single collection.*

Description

An aggregate function that collects multiple geometries or geographies into a single GeometryCollection or the appropriate Multi-type if all inputs share the same geometry type. Collecting geographies returns a geography.

Usage

```
sd_collect_agg(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_COLLECT_AGG

`sd_concave_hull` *Returns a concave hull enclosing the input geometry.*

Description

Returns a concave hull enclosing the input geometry. The `pct_convex` parameter controls how "tight" the hull is: 1.0 produces the convex hull, while smaller values produce tighter, more concave shapes. With `allow_holes` set to true, the hull may have holes; otherwise (the default) it has none.

Usage

```
sd_concave_hull(
  geom = sd_missing_arg(),
  pct_convex = sd_missing_arg(),
  allow_holes = sd_missing_arg()
)
```

Arguments

`geom`	(geometry): Input geometry
`pct_convex`	(double): Input double
`allow_holes`	(boolean): Whether the hull may have holes. Defaults to false.

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_CONCAVEHULL

`sd_contains` *Returns true if geomA contains geomB.*

Description

Returns `true` if no points of B lie outside A and at least one point of B lies inside A.

Usage

```
sd_contains(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_CONTAINS

`sd_convex_hull` *Returns the Convex Hull of a geometry or geography.*

Description

Returns the smallest convex polygon that encloses all points in the input geometry.

Usage

```
sd_convex_hull(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_CONVEXHULL

sd_convex_hull_agg *An aggregate function that returns the convex hull of all geometries in a group.*

Description

An aggregate function that computes the smallest convex polygon enclosing all points across every geometry in a group.

Usage

```
sd_convex_hull_agg(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_CONVEXHULL_AGG

sd_covered_by *Returns true if geomA is covered by geomB.*

Description

Returns `true` if no point in geometry A is outside geometry B. Inverse of `ST_Covers`: `ST_CoveredBy(A, B)` is equivalent to `ST_Covers(B, A)`.

Usage

```
sd_covered_by(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_COVEREDBY

`sd_covering_cell_ids` *Returns an array of S2 cell IDs that cover the given geography.*

Description

Returns an array of 64-bit S2 cell IDs that form a covering of the given geography. A covering is a set of cells that completely contains the geography with minimal overlap. For a point, returns an array with a single cell ID. For empty geometries, returns an empty array. This is useful for spatial indexing and efficient spatial queries. The returned cells may be at different S2 levels depending on the size and shape of the geography.

Usage

```
sd_covering_cell_ids(
  geom = sd_missing_arg(),
  min_level = sd_missing_arg(),
  max_level = sd_missing_arg(),
  max_cells = sd_missing_arg()
)
```

Arguments

`geom`	(geography): Input geography
`min_level`	(int64): Input int64
`max_level`	(int64): Input int64
`max_cells`	(int64): Maximum number of cells to return. Must be greater than 0. Defaults to 8.

Value

(array)

See Also

SedonaDB SQL documentation for S2_COVERINGCELLIDS

sd_covers *Returns true if geomA covers geomB.*

Description

Returns `true` if no point in geometry B is outside geometry A. Similar to `ST_Contains` but does not require interior intersection.

Usage

```
sd_covers(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_COVERS

sd_crosses *Returns true if A crosses B.*

Description

Returns `true` if the two geometries have some (but not all) interior points in common and the dimension of the intersection is less than that of either input.

Usage

```
sd_crosses(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_CROSSES

`sd_crs` *Returns the Coordinate Reference System (CRS) metadata associated with a geometry or geography object.*

Description

Returns the CRS (Coordinate Reference System) string associated with a geometry or geography. Whether it was set as an authority code (e.g. `'EPSG:3857'`), PROJJSON, or WKT, the CRS is returned in the same form it was set. See Coordinate Reference Systems for how SedonaDB represents and compares CRS.

Usage

```
sd_crs(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(string)

See Also

SedonaDB SQL documentation for ST_CRS

`sd_d_within` *Returns true if two geometries or geographies are within a specified distance of each other.*

Description

For geography, the distance is specified in meters and represents the spherical approximation of the geodesic distance between the two geographies.

Usage

```
sd_d_within(
  geom_a = sd_missing_arg(),
  geom_b = sd_missing_arg(),
  distance = sd_missing_arg()
)
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry
distance	(double): Input double

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_DWITHIN

---

sd_delaunay_triangles *Returns a geometry collection of Delaunay triangles covering the vertices of the input geometry.*

Description

Computes a Delaunay triangulation of the vertices of the input geometry. The result is a GEOMETRYCOLLECTION of POLYGON triangles unless the flags parameter is set to request MULTILINESTRING output. An optional tolerance value can be specified to snap nearby vertices together before triangulation.

Usage

```
sd_delaunay_triangles(
  geom = sd_missing_arg(),
  tolerance = sd_missing_arg(),
  flags = sd_missing_arg()
)
```

Arguments

geom	(geometry): Input geometry
tolerance	(float64): Snap-rounding tolerance for vertices. Use 0.0 for exact computation.
flags	(integer): Use 0 for polygon triangles or 1 for edge-only MULTILINESTRING output.

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_DELAUNAYTRIANGLES

sd_difference *Computes the difference between geomA and geomB.*

Description

Returns the part of geometry A that does not intersect with geometry B.

Usage

```
sd_difference(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_DIFFERENCE

sd_dimension *Returns the dimension of the geometry or geography.*

Description

Returns the inherent dimension of the geometry: - 0 for points or multipoints

- 1 for linestring or multilinestrings - 2 for polygons or multipolygons For geometry collections, returns the largest dimension among the components.

Usage

```
sd_dimension(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_DIMENSION

sd_disjoint — *Returns true if geomA is disjoint from geomB.*

Description

Returns `true` if the two geometries do not share any points. This is the inverse of `ST_Intersects`.

Usage

```
sd_disjoint(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_DISJOINT

sd_distance — *Returns the distance between two geometries or geographies.*

Description

For geometry, returns the 2D Cartesian (planar) distance between two geometries, in the units of the coordinate reference system. For geography, returns the spherical approximation of the geodesic distance between two geographies in meters.

Usage

```
sd_distance(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_DISTANCE

---

sd_dump *Expands multi-part geometries into child parts*

Description

If the geometry is simple it returns the geometry itself, if the geometry is collection or multi it returns record for each of collection components. Nested collections are expanded recursively down to their atomic parts, with `path` giving each part's position, as in PostGIS. To list only a geometry's direct members, use ST_Geometries.

Usage

```
sd_dump(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(struct)

See Also

SedonaDB SQL documentation for ST_DUMP

---

sd_end_point *Returns last point of a linestring.*

Description

Returns last point of a linestring.

Usage

```
sd_end_point(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_ENDPOINT

`sd_envelope` *Returns the bounding box (envelope) of a geometry or geography as a new geometry.*

Description

Returns the minimum axis-aligned bounding box of a geometry. This is usually a polygon but can be a linestring for perfectly vertical or horizontal input and is a point if the input is also a point. For geography, the return value is still a rectangular polygon geometry that represents the latitude and longitude range of the input. For geographies that span the antimeridian, a multipolygon is returned (one on each side of the antimeridian). Because of slight numerical inaccuracy of geographical calculations, the bounds are expanded slightly compared to the corresponding geometry envelope.

Usage

```
sd_envelope(geom = sd_missing_arg())
```

Arguments

`geom` (geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_ENVELOPE

`sd_envelope_agg` *An aggregate function that returns the collective bounding box (envelope) of a set of geometries.*

Description

An aggregate function that computes the collective bounding box (envelope) of all geometries in a group. Geography input is also accepted; the envelope is always returned as a geometry. For geographies that cross the antimeridian, a multi geometry is returned with one polygon each for the west and east component.

Usage

```
sd_envelope_agg(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_ENVELOPE_AGG

---

sd_equals *Returns true if geomA equals geomB.*

Description

Returns `true` if the two geometries are topologically equal (i.e., they represent the same point set, regardless of vertex order).

Usage

```
sd_equals(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_EQUALS

sd_exterior_ring *Returns the exterior ring of a polygon as a linestring geometry.*

Description

Returns the exterior ring (outer boundary) of a polygon as a LINESTRING. Returns NULL for non-polygon geometry types.

Usage

```
sd_exterior_ring(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_EXTERIORRING

sd_flip_coordinates *Returns a new geometry with the X and Y coordinates of each vertex swapped.*

Description

Returns a new geometry with the X and Y coordinates of each vertex swapped.

Usage

```
sd_flip_coordinates(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_FLIPCOORDINATES

sd_force2d — *Forces a geometry or geography into a XY coordinate model.*

Description

Discards any Z and M values present (if any).

Usage

```
sd_force2d(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_FORCE2D

sd_force3d — *Forces a geometry or geography into a XYZ coordinate model with an optional Z value.*

Description

If the geometry already has Z values they are preserved; otherwise the optional z argument (default 0) is used. Aliases: ST_Force3DZ.

Usage

```
sd_force3d(geom = sd_missing_arg(), z = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry
z	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_FORCE3D

sd_force3dm — *Forces a geometry or geography into a XYM coordinate model with an optional M value.*

Description

If the geometry already has M values they are preserved; otherwise the optional `m` argument (default 0) is used.

Usage

```
sd_force3dm(geom = sd_missing_arg(), m = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`m`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_FORCE3DM

sd_force4d — *Forces a geometry or geography into a XYZM coordinate model with optional Z and M values.*

Description

If the geometry already has Z and M values they are preserved; otherwise optional `z` and `m` arguments (both default 0) are used.

Usage

```
sd_force4d(geom = sd_missing_arg(), z = sd_missing_arg(), m = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`z`	(double): Input double
`m`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_FORCE4D

`sd_geo_hash` *Returns the GeoHash string of a geometry with the given precision.*

Description

Encodes the geometry as a base-32 GeoHash string with `precision` characters. Non-point geometries are encoded using the center of their bounding box. A non-positive precision returns an empty string, and precision is capped at 20 characters. A GeoHash is defined against the WGS84 datum, so the argument must carry either a WGS84 CRS (`EPSG:4326` / `OGC:CRS84`) or no CRS at all; anything else raises an error. Use `ST_Transform()` to reproject first. This includes other geographic CRSes such as NAD83, whose units are degrees but whose datum differs from WGS84 by enough to shift the result at high precision. The same rule applies whether the CRS belongs to the column or to the individual rows; only the timing differs. A column-level CRS is known while the query is planned, so it is rejected before any row is read. A row-level CRS is not known until the row is reached, so it is rejected then. To encode only the WGS84 rows of a mixed column, filter them first — `WHERE ST_SRID(geom) = 4326` is applied before the encoding, so the other rows never reach it. Latitudes outside -90 to 90 return `NULL`. Longitudes outside -180 to 180 are wrapped back into range when the argument declares a WGS84 CRS, so a longitude of 181 encodes as -179; without a declared CRS they return `NULL` instead, matching Apache Sedona. When `precision` is omitted, the argument must be a `POINT` and is encoded at the maximum precision of 20 characters. Any other geometry type raises an error, because there is no single obvious precision for a geometry that covers an area. Geography arguments are bounded on the sphere and require the s2geography bounder to be registered on the session.

Usage

```
sd_geo_hash(geom = sd_missing_arg(), precision = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`precision`	(integer): Number of base-32 characters in the returned GeoHash.

Value

(string)

See Also

SedonaDB SQL documentation for ST_GEOHASH

sd_geog_from_wkb *Constructs a Geography from WKB Binary.*

Description

Constructs a Geography from WKB Binary.

Usage

```
sd_geog_from_wkb(wkb = sd_missing_arg())
```

Arguments

wkb	(binary): Input binary

Value

(geography)

See Also

SedonaDB SQL documentation for ST_GEOGFROMWKB

sd_geog_from_wkt *Constructs a Geography from WKT.*

Description

Alias: ST_GeogFromText.

Usage

```
sd_geog_from_wkt(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

wkt	(string): Input string
srid	(integer): Input integer

Value

(geography)

See Also

SedonaDB SQL documentation for ST_GEOGFROMWKT

sd_geog_point

Creates a geography POINT from given longitude and latitude coordinates.

Description

Creates a geography Point from longitude and latitude coordinates.

Usage

```
sd_geog_point(longitude = sd_missing_arg(), latitude = sd_missing_arg())
```

Arguments

longitude	(double): Input double
latitude	(double): Input double

Value

(geography)

See Also

SedonaDB SQL documentation for ST_GEOGPOINT

sd_geom_coll_from_text

Constructs a GeometryCollection from Well-Known Text (WKT), erroring if the input is not a GeometryCollection.

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a GeometryCollection.

Usage

```
sd_geom_coll_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

wkt	(string): Input string
srid	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_GEOMCOLLFROMTEXT

---

sd_geom_from_ewkb *Constructs a geometry from Extended Well-Known Binary (EWKB).*

Description

Parses an EWKB binary value and returns a geometry. EWKB extends the WKB format to include SRID information in the binary header and is useful for PostGIS compatibility.

Usage

```
sd_geom_from_ewkb(ewkb = sd_missing_arg())
```

Arguments

ewkb	(binary): Input binary

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_GEOMFROMEWKB

---

sd_geom_from_ewkt *Constructs a geometry from Extended Well-Known Text (EWKT).*

Description

Parses an EWKT string in the format SRID=<srid>;<wkt> and returns a geometry with the embedded SRID set. EWKT is used by PostGIS and other systems for human-readable output when an item-level SRID required.

Usage

```
sd_geom_from_ewkt(ewkt = sd_missing_arg())
```

Arguments

ewkt	(string): Input string

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_GEOMFROMEWKT

`sd_geom_from_wkb` *Constructs a Geometry from Well-Known Binary (WKB).*

Description

Parses a WKB binary value and returns a geometry. The input is validated by default. An optional SRID or CRS can be provided as a second argument.

Usage

```
sd_geom_from_wkb(wkb = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`wkb`	(binary): Input binary
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_GEOMFROMWKB

`sd_geom_from_wkt` *Constructs a Geometry from Well-Known Text (WKT).*

Description

An optional SRID or CRS can be provided as a second argument to set the spatial reference. Aliases: `ST_GeomFromText`, `ST_GeometryFromText`.

Usage

```
sd_geom_from_wkt(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`wkt`	(string): Input string
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_GEOMFROMWKT

`sd_geometries` *Returns the direct members of a geometry as a list, without recursing into nested collections.*

Description

Returns `ST_GeometryN(geom, n)` for each `n` from 1 to the number of members: the parts of a multi-geometry or the members of a geometry collection, one level deep, so a collection nested inside a collection is returned whole. A geometry that is not a collection is its own single member, even when it is empty, and an empty multi-geometry or collection returns an empty list. The members keep the input's spatial type and CRS, and NULL returns NULL. Unlike ST_Dump, which recurses into nested collections down to their atomic parts, `ST_Geometries` returns only a geometry's direct members. See also ST_GeometryN.

Usage

```
sd_geometries(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(list)

See Also

SedonaDB SQL documentation for ST_GEOMETRIES

`sd_geometry_n` *Returns the 1-based Nth geometry if the geometry is a GEOMETRYCOLLECTION, (MULTI)POINT, (MULTI)LINESTRING, MULTICURVE or (MULTI)POLYGON.*

Description

Returns NULL if the index is out of range.

Usage

```
sd_geometry_n(geom = sd_missing_arg(), n = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`n`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_GEOMETRYN

`sd_geometry_type` *Returns the type of a geometry or geography.*

Description

Returns the OGC geometry type name prefixed with `ST_`. Possible return values include: `'ST_Point'`, `'ST_LineString'`, `'ST_Polygon'`, `'ST_MultiPoint'`, `'ST_MultiLineString'`, `'ST_MultiPolygon'`, and `'ST_GeometryCollection'`.

Usage

```
sd_geometry_type(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(string)

See Also

SedonaDB SQL documentation for ST_GEOMETRYTYPE

`sd_has_m` *Returns true if the geometry has a M dimension.*

Description

Returns true if the geometry has an M dimension. A geometry collection has one when any of its members does, at any depth, an empty member included.

Usage

```
sd_has_m(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_HASM

`sd_has_z` *Returns true if the geometry has a Z dimension.*

Description

Returns true if the geometry has a Z dimension. A geometry collection has one when any of its members does, at any depth, an empty member included.

Usage

```
sd_has_z(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_HASZ

`sd_hausdorff_distance` *Returns the Hausdorff distance between two geometries.*

Description

The Hausdorff distance is a measure of the similarity between two geometric shapes. This implementation returns the greatest of all distances from a point in one geometry to the closest point in the other geometry. Returns `NULL` if either geometry is `NULL`. Returns 0 if either geometry is empty.

Usage

```
sd_hausdorff_distance(
  geom_a = sd_missing_arg(),
  geom_b = sd_missing_arg(),
  densifyFrac = sd_missing_arg()
)
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry
`densifyFrac`	(double): A fraction by which to densify each segment of the geometries before computing the distance. A value of 0.25 will densify the segment by adding vertices every 1/4 of the segment length. Densification can give a more accurate result for geometries with long segments or complex shapes.

Value

(double)

See Also

SedonaDB SQL documentation for ST_HAUSDORFFDISTANCE

sd_interior_ring_n *Returns the Nth interior ring of a polygon.*

Description

Returns the Nth interior LINESTRING ring of a POLYGON. Returns NULL if the geometry is not a polygon or n is out of range. n is 1-based, as in PostGIS: 1 returns the first interior ring.

Usage

```
sd_interior_ring_n(geom = sd_missing_arg(), n = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry
n	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_INTERIORRINGN

sd_intersection *Computes the intersection of two geometries or geographies.*

Description

Computes the intersection of two geometries or geographies.

Usage

```
sd_intersection(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_INTERSECTION

sd_intersection_agg *Returns the cumulative intersection of all geometries in the input.*

Description

Returns the cumulative intersection of all geometries in the input.

Usage

```
sd_intersection_agg(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_INTERSECTION_AGG

sd_intersects *Returns true if geomA intersects geomB.*

Description

Returns true if geomA intersects geomB.

Usage

```
sd_intersects(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_INTERSECTS

sd_is_closed — *Returns true if the LINESTRING start and end point are the same.*

Description

Returns `true` if the LineString's start and end points are the same. For MultiLineStrings, all elements must be closed.

Usage

```
sd_is_closed(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISCLOSED

sd_is_collection — *Returns true if the geometry or geography type of the input is a collection type.*

Description

Returns TRUE if the geometry type of the input is a geometry collection type. Collection types are the following: - GEOMETRYCOLLECTION - MULTIPOINT - MULTIPOLYGON - MULTILINESTRING

Usage

```
sd_is_collection(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISCOLLECTION

`sd_is_empty` *Returns true if the geometry or geography is empty.*

Description

Returns true if the geometry or geography is empty.

Usage

```
sd_is_empty(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISEMPTY

`sd_is_polygon_ccw` *Returns true if every polygon in the geometry has its exterior ring oriented counter-clockwise.*

Description

Returns `true` if every polygon in the geometry has its exterior ring oriented counter-clockwise and its interior rings clockwise. Polygons inside a `GEOMETRYCOLLECTION` are checked at any depth, and non-polygonal parts are ignored, so an input without any polygon (including empty geometries) returns `true`.

Usage

```
sd_is_polygon_ccw(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISPOLYGONCCW

`sd_is_polygon_cw` *Returns true if every polygon in the geometry has its exterior ring oriented clockwise.*

Description

Returns `true` if every polygon in the geometry has its exterior ring oriented clockwise and its interior rings counter-clockwise. Polygons inside a `GEOMETRYCOLLECTION` are checked at any depth, and non-polygonal parts are ignored, so an input without any polygon (including empty geometries) returns `true`.

Usage

```
sd_is_polygon_cw(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISPOLYGONCW

`sd_is_ring` *Returns true if a linestring is ST_IsClosed and ST_IsSimple.*

Description

Returns `true` if the linestring is both closed (`ST_IsClosed`) and simple (`ST_IsSimple`). In other words, the input forms a ring with no self-intersections.

Usage

```
sd_is_ring(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISRING

`sd_is_simple` *Tests if geometry's only self-intersections are at boundary points.*

Description

Returns `true` if the geometry has no anomalous geometric points such as self-intersections or self-tangency.

Usage

```
sd_is_simple(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISSIMPLE

sd_is_valid — *Checks whether a geometry meets the rules of a valid spatial object according to the OGC standard.*

Description

Returns `true` if the geometry is well-formed according to OGC rules. A geometry that GEOS cannot build or validate is reported as `false` rather than raising an error. Use `ST_IsValidReason` to get a diagnostic message for invalid geometries and `ST_MakeValid` to attempt to repair them.

Usage

```
sd_is_valid(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_ISVALID

sd_is_valid_reason — *Returns a text explanation describing why a geometry is invalid.*

Description

Returns a text string explaining why a geometry is invalid, or `"Valid Geometry"` if the geometry is valid. Useful for debugging geometry construction. If GEOS cannot build or validate the geometry, for example because a polygon ring is not closed or because coordinates close to the largest double value overflow the validity checks, the reason is the GEOS error message, such as `"IllegalArgumentException: Points of LinearRing do not form a closed linestring"`.

Usage

```
sd_is_valid_reason(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(string)

See Also

SedonaDB SQL documentation for ST_ISVALIDREASON

`sd_knn` *Returns true if geomA finds k nearest neighbors from geomB.*

Description

`ST_KNN` is used in a join condition to find the k nearest neighbors of one geometry from another set of geometries. The actual k-nearest neighbors logic is handled by the spatial join execution engine: the function itself is a stub that simply defines the expected signature.

Usage

```
sd_knn(
  geomA = sd_missing_arg(),
  geomB = sd_missing_arg(),
  k = sd_missing_arg(),
  use_spheroid = sd_missing_arg()
)
```

Arguments

`geomA`	(geometry): The geometry around which to search.
`geomB`	(geometry): Column containing candidate geometries.
`k`	(integer): The number of nearest neighbours to return. Defaults to 1.
`use_spheroid`	(boolean): true to use spherical distance, false for Euclidean. Defaults to false.

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_KNN

sd_length — *Returns the length of a geometry or geography.*

Description

Returns the length of geom. This function only supports LineString, MultiLineString, and GeometryCollections containing linear geometries. Use ST_Perimeter for polygons. For geometry, returns the length in the units of the coordinate reference system; for geography, returns the spherical approximation of the geodesic length in meters.

Usage

```
sd_length(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_LENGTH

sd_line_from_text — *Constructs a LineString from Well-Known Text (WKT), erroring if the input is not a LineString.*

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a LineString. Aliases: ST_LineStringFromText.

Usage

```
sd_line_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

wkt	(string): Input string
srid	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_LINEFROMTEXT

`sd_line_interpolate_point` *Returns a point interpolated along a line.*

Description

Returns a point interpolated along a line. First argument must be a LINESTRING. Second argument is a double between 0 and 1 representing fraction of total linestring length the point has to be located. For geography, the interpolation is computed using spherical interpolation between vertices.

Usage

```
sd_line_interpolate_point(geom = sd_missing_arg(), fraction = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`fraction`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_LINEINTERPOLATEPOINT

`sd_line_locate_point` *Returns the location of the closest point on a LineString as a fraction of its total length.*

Description

Returns a double between 0 and 1, representing the location of the closest point on the LineString as a fraction of its total 2D length. The first argument must be a LineString. For geometry, the fraction is computed using 2D Cartesian distance. For geography, the fraction is computed based on spherical interpolation between vertices.

Usage

```
sd_line_locate_point(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_LINELOCATEPOINT

---

sd_line_merge

Merges a collection of potentially connected line segments into the fewest possible LineStrings.

Description

With directed set to true, lines are merged only where one ends where the next starts, so lines running in opposite directions stay apart. Otherwise (the default), lines may be reversed to merge them.

Usage

```
sd_line_merge(geom = sd_missing_arg(), directed = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry
directed	(boolean): Whether to keep the direction of the lines. Defaults to false.

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_LINEMERGE

`sd_line_string_from_wkb` *Constructs a LineString from Well-Known Binary (WKB), returning NULL if the input is another geometry type.*

Description

Parses a WKB binary value and returns a LineString geometry. Returns `NULL` if the WKB represents another geometry type. An optional SRID or CRS can be provided as a second argument. Alias: `ST_LineFromWKB`.

Usage

```
sd_line_string_from_wkb(wkb = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`wkb`	(binary): Input binary
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_LINESTRINGFROMWKB

`sd_line_substring` *Returns a linestring being a substring of the input one starting and ending at the given fractions of total 2d length.*

Description

Returns a linestring being a substring of the input one starting and ending at the given fractions of total 2d length.

Usage

```
sd_line_substring(
  geom = sd_missing_arg(),
  start_fraction = sd_missing_arg(),
  end_fraction = sd_missing_arg()
)
```

Arguments

`geom`	(geometry): Input geometry
`start_fraction`	(double): The fraction from which the returned line will start (between 0 and 1)
`end_fraction`	(double): The fraction from which the returned line will end (between 0 and 1)

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_LINESUBSTRING

`sd_longest_line` *Returns the 2-dimensional longest line between the vertices of two geometries or geographies.*

Description

Returns the 2-dimensional longest line between the vertices of two geometries or geographies.

Usage

```
sd_longest_line(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_LONGESTLINE

sd_m — *Returns the M (measure) coordinate of a Point geometry.*

Description

Extracts the M (measure) coordinate from a Point geometry or geography. Returns NULL if the geometry has no M dimension or for non-point geometries.

Usage

```
sd_m(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_M

sd_m_line_from_text — *Constructs a MultiLineString from Well-Known Text (WKT), erroring if the input is not a MultiLineString.*

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a MultiLineString.

Usage

```
sd_m_line_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

wkt	(string): Input string
srid	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MLINEFROMTEXT

`sd_m_max` *Returns the maximum M (measure) value from a geometry or geography's bounding box.*

Description

Returns the maximum M (measure) value from a geometry or geography's bounding box.

Usage

```
sd_m_max(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_MMAX

`sd_m_min` *Returns the minimum M-coordinate (measure) of a geometry or geography's bounding box.*

Description

Returns the minimum M-coordinate (measure) of a geometry or geography's bounding box.

Usage

```
sd_m_min(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_MMIN

`sd_m_point_from_text` *Constructs a MultiPoint from Well-Known Text (WKT), erroring if the input is not a MultiPoint.*

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a MultiPoint.

Usage

```
sd_m_point_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`wkt`	(string): Input string
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MPOINTFROMTEXT

`sd_m_poly_from_text` *Constructs a MultiPolygon from Well-Known Text (WKT), erroring if the input is not a MultiPolygon.*

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a MultiPolygon.

Usage

```
sd_m_poly_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`wkt`	(string): Input string
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MPOLYFROMTEXT

`sd_make_envelope` *Constructs a rectangular Polygon from minimum and maximum coordinates.*

Description

Constructs a rectangular Polygon from minimum X, minimum Y, maximum X, and maximum Y coordinates. An optional SRID or CRS can be provided as a fifth argument.

Usage

```
sd_make_envelope(
  min_x = sd_missing_arg(),
  min_y = sd_missing_arg(),
  max_x = sd_missing_arg(),
  max_y = sd_missing_arg(),
  srid = sd_missing_arg()
)
```

Arguments

`min_x`	(double): Input double
`min_y`	(double): Input double
`max_x`	(double): Input double
`max_y`	(double): Input double
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MAKEENVELOPE

sd_make_line — *Creates a LineString from two or more input geometries or geographies.*

Description

Creates a `LineString` from two or more input `Point`, `MultiPoint`, or `LineString` geometries or geographies. The function connects the inputs in the order they are provided to form a single continuous line. Combining geographies returns a geography.

Usage

```
sd_make_line(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MAKELINE

sd_make_valid — *Creates a valid representation of an invalid geometry.*

Description

Collapsed geometries are either converted to empty (keepCollapsed=false) or a valid geometry of lower dimension (keepCollapsed=true). Default is keepCollapsed=false.

Usage

```
sd_make_valid(geom = sd_missing_arg(), keepCollapsed = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`keepCollapsed`	(boolean): Input boolean

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MAKEVALID

`sd_max_distance` *Returns the maximum distance between any pair of points in two geometries or geographies.*

Description

For geometry, returns the 2D Cartesian maximum distance between any pair of vertices across the two geometries, in the units of the coordinate reference system. For geography, returns the maximum distance in meters calculated using a spherical approximation.

Usage

```
sd_max_distance(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_MAXDISTANCE

sd_minimum_clearance *Returns the minimum clearance of a geometry.*

Description

The minimum clearance is a metric that quantifies a geometry's tolerance to changes in coordinate precision or vertex positions. It represents the maximum distance by which vertices can be adjusted without introducing invalidity to the geometry's structure. A larger minimum clearance value indicates greater robustness against such perturbations. For a geometry with a minimum clearance of x, the following conditions hold: - No two distinct vertices are separated by a distance less than x. - No vertex lies within a distance x from any line segment it is not an endpoint of. For geometries with no definable minimum clearance, such as single Point geometries or MultiPoint geometries where all points occupy the same location, the function returns Double.MAX_VALUE.

Usage

```
sd_minimum_clearance(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_MINIMUMCLEARANCE

sd_minimum_clearance_line *Returns a LineString representing the minimum clearance distance of the input geometry.*

Description

This function returns a two-point LineString geometry representing the minimum clearance distance of the input geometry. If the input geometry does not have a defined minimum clearance, such as for single Points or coincident MultiPoints, an empty LineString geometry is returned instead.

Usage

```
sd_minimum_clearance_line(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_MINIMUMCLEARANCELINE

---

sd_missing_arg *Missing argument sentinel*

Description

Missing argument sentinel

Usage

```
sd_missing_arg()
```

Value

An object of class 'sd_missing_arg'

---

sd_n_points *Returns the number of points of the geometry or geography.*

Description

Returns the total number of coordinate points in the geometry, counting all vertices across all components. Aliases: `ST_NumPoints`.

Usage

```
sd_n_points(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_NPOINTS

sd_n_rings — *Returns the total number of rings in a geometry or geography (both exterior and interior rings).*

Description

Returns the total number of rings (exterior and interior) in a polygon or multipolygon. For a simple polygon, this is 1 plus the number of holes. For a multipolygon, it is the sum of all rings in all polygons. Returns 0 for non-polygon geometry types.

Usage

```
sd_n_rings(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_NRINGS

sd_normalize — *Returns the geometry or geography in its canonical form.*

Description

Returns the geometry or geography in its canonical form.

Usage

```
sd_normalize(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_NORMALIZE

sd_num_geometries *Returns the number of geometries in a geometry or geography collection.*

Description

Returns the number of Geometries. If geometry is a GEOMETRYCOLLECTION (or MULTI*) return the number of geometries, for single geometries will return 1.

Usage

```
sd_num_geometries(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_NUMGEOMETRIES

sd_num_interior_rings *Returns the number of interior rings (holes) in a polygon geometry or geography.*

Description

Returns the number of interior rings (holes) in a polygon. For a polygon without holes, returns 0. For multipolygons, returns the number of interior rings in the first polygon. Returns NULL for non-polygon geometry types. Aliases: ST_NumInteriorRing.

Usage

```
sd_num_interior_rings(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_NUMINTERIORRINGS

`sd_num_points` *Returns the number of points of a linestring geometry or geography.*

Description

Returns the total number of coordinate points in a linestring geometry. Returns NULL for other geometry types. Use `ST_NPoints()` to calculate the number of vertices for all geometry types.

Usage

```
sd_num_points(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_NUMPOINTS

`sd_overlaps` *Returns true if A overlaps B.*

Description

Returns `true` if the two geometries share space and have the same dimension, but are not completely contained by each other.

Usage

```
sd_overlaps(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_OVERLAPS

---

sd_perimeter *Calculates the perimeter of a given geometry or geography.*

Description

This function calculates the perimeter of a given geometry or geography. It supports Polygon, MultiPolygon, and GeometryCollection geometries (as long as the GeometryCollection contains polygonal geometries). For other types, it returns 0. To measure lines, use ST_Length. For geography, returns the spherical approximation of the geodesic perimeter in meters.

Usage

```
sd_perimeter(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_PERIMETER

---

sd_point *Constructs a Point Geometry from X and Y.*

Description

Constructs a Point geometry from X and Y coordinates. An optional SRID or CRS can be provided as a third argument.

Usage

```
sd_point(x = sd_missing_arg(), y = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

x	(double): Input double
y	(double): Input double
srid	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINT

---

sd_point_from_text *Constructs a Point from Well-Known Text (WKT), erroring if the input is not a Point.*

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a Point.

Usage

```
sd_point_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

wkt	(string): Input string
srid	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTFROMTEXT

sd_point_from_wkb — *Constructs a Point from Well-Known Binary (WKB), returning NULL if the input is another geometry type.*

Description

Parses a WKB binary value and returns a Point geometry. Returns `NULL` if the WKB represents another geometry type. An optional SRID or CRS can be provided as a second argument.

Usage

```
sd_point_from_wkb(wkb = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`wkb`	(binary): Input binary
`srid`	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTFROMWKB

sd_point_m — *Constructs a Point with an M (measure) coordinate from X, Y, and M values.*

Description

Constructs a Point with an M (measure) coordinate from X, Y, and M values.

Usage

```
sd_point_m(x = sd_missing_arg(), y = sd_missing_arg(), m = sd_missing_arg())
```

Arguments

`x`	(double): Input double
`y`	(double): Input double
`m`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTM

`sd_point_n` *Returns the Nth point in a linestring.*

Description

Return the Nth point in a single linestring. Negative values are counted backwards from the end of the LineString such that -1 is the last point. Returns NULL if there is no linestring in the geometry.

Usage

```
sd_point_n(geom = sd_missing_arg(), n = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`n`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTN

`sd_point_on_surface` *Returns a point guaranteed to lie on the surface of a geometry or geography.*

Description

Unlike ST_Centroid which does not necessarily intersect a geometry or geography, ST_PointOnSurface makes an attempt to find an interior point close to the middle (using deterministic heuristics).

Usage

```
sd_point_on_surface(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTONSURFACE

---

sd_point_z *Constructs a Point with a Z coordinate from X, Y, and Z values.*

Description

Constructs a Point with a Z coordinate from X, Y, and Z values.

Usage

```
sd_point_z(x = sd_missing_arg(), y = sd_missing_arg(), z = sd_missing_arg())
```

Arguments

x	(double): Input double
y	(double): Input double
z	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTZ

sd_point_zm *Constructs a Point with X, Y, Z and M coordinates.*

Description

Constructs a Point with X, Y, Z and M coordinates.

Usage

```
sd_point_zm(
  x = sd_missing_arg(),
  y = sd_missing_arg(),
  z = sd_missing_arg(),
  m = sd_missing_arg()
)
```

Arguments

x	(double): Input double
y	(double): Input double
z	(double): Input double
m	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTZM

sd_points *Returns a MultiPoint geometry consisting of all the coordinates of the input geometry or geography.*

Description

Returns a MultiPoint geometry consisting of all the coordinates of the input geometry. It preserves duplicate points as well as M and Z coordinates.

Usage

```
sd_points(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POINTS

---

sd_polygon_from_text *Constructs a Polygon from Well-Known Text (WKT), erroring if the input is not a Polygon.*

Description

Parses a WKT string and returns a geometry. Raises an error if the WKT does not represent a Polygon.

Usage

```
sd_polygon_from_text(wkt = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

wkt	(string): Input string
srid	(crs): Input crs

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POLYGONFROMTEXT

`sd_polygonize` *Builds a polygonal geometry from linear components in the input geometry.*

Description

Builds a polygonal geometry from linear components in the input geometry. Handles GeometryCollection and multi-geometries by extracting linear components. Typically used with collections of complete, simple linework that forms polygon boundaries. Unlike many geometry constructors, this doesn't require explicit closure of rings—any closed rings in the input will become polygon boundaries.

Usage

```
sd_polygonize(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POLYGONIZE

`sd_polygonize_agg` *Creates polygons from a set of geometries that contain linework representing the edges of a polygon.*

Description

Creates polygons from a set of geometries that contain linework representing the edges of a polygon.

Usage

```
sd_polygonize_agg(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_POLYGONIZE_AGG

`sd_reduce_precision` *Reduces the coordinate precision of a geometry or geography to the specified grid size.*

Description

Rebuilds a new geometry after specifying all vertices to the specified size. Z and M values are preserved but not affected by the grid snapping. The implementation is similar to a unary union and is designed to output valid geometry (e.g., by avoiding creating new crossing edges when snapping vertices).

Usage

```
sd_reduce_precision(geom = sd_missing_arg(), grid_size = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`grid_size`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_REDUCEPRECISION

`sd_relate` *Returns the DE-9IM intersection matrix string for two geometries, or tests whether two geometries satisfy a given intersection matrix pattern.*

Description

When called with two geometry arguments, returns the DE-9IM (Dimensionally Extended 9-Intersection Model) intersection matrix as a 9-character string describing the spatial relationship between two geometries. When called with a third string argument, returns `true` if the two geometries satisfy the given DE-9IM intersection matrix pattern, `false` otherwise. The pattern can use wildcards (`*`) and `T` (any non-empty intersection) in addition to exact dimension values (0, 1, 2, F).

Usage

```
sd_relate(
  geom_a = sd_missing_arg(),
  geom_b = sd_missing_arg(),
  intersectionMatrixPattern = sd_missing_arg()
)
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry
`intersectionMatrixPattern`	(string): A 9-character DE-9IM pattern string. Each character can be 0-2, T, F, or * (wildcard).

Value

(string)

See Also

SedonaDB SQL documentation for ST_RELATE

`sd_reverse` *Returns the geometry or geography with vertex order reversed.*

Description

Returns the geometry or geography with vertex order reversed.

Usage

```
sd_reverse(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_REVERSE

sd_rotate — *Rotates a geometry counter-clockwise around the Z axis by an angle in radians.*

Description

Rotates a geometry counter-clockwise around the Z axis by an angle in radians.

Usage

```
sd_rotate(geom = sd_missing_arg(), rot = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry
rot	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_ROTATE

sd_rotate_x — *Rotates a geometry around the X axis by an angle in radians.*

Description

Rotates a geometry around the X axis by an angle in radians.

Usage

```
sd_rotate_x(geom = sd_missing_arg(), rot = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry
rot	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_ROTATEX

sd_rotate_y *Rotates a geometry around the Y axis by an angle in radians.*

Description

Rotates a geometry around the Y axis by an angle in radians.

Usage

```
sd_rotate_y(geom = sd_missing_arg(), rot = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry
rot	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_ROTATEY

sd_scale *Scales a geometry by multiplying ordinates with scale factors.*

Description

Scales a geometry by multiplying each coordinate by the corresponding scale factor. A 3D variant accepts scaleZ.

Usage

```
sd_scale(
  geom = sd_missing_arg(),
  scaleX = sd_missing_arg(),
  scaleY = sd_missing_arg(),
  scaleZ = sd_missing_arg()
)
```

Arguments

geom	(geometry): Input geometry
scaleX	(double): Input double
scaleY	(double): Input double
scaleZ	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SCALE

`sd_segmentize` *Densifies a geometry by adding intermediate points along segments that exceed a maximum length.*

Description

Densifies a geometry or geography by adding intermediate points along line segments that exceed the specified maximum segment length. For geometry, uses planar distance; for geography, uses the spherical approximation of the geodesic distance in meters and points are added along the great circle arc. In both cases, points are added so that each resulting segment is no longer than `max_segment_length`.

Usage

```
sd_segmentize(geom = sd_missing_arg(), max_segment_length = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`max_segment_length`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SEGMENTIZE

sd_set_crs *Sets the Coordinate Reference System (CRS) for a geometry.*

Description

Sets a CRS to a geometry/geography. This is metadata-only - no coordinates are transformed. See Coordinate Reference Systems for how SedonaDB represents and compares CRS. This is different from ST_Transform which actually transforms coordinates. `target_crs` accepts an authority code (e.g. `'EPSG:4326'`), PROJJSON, or WKT. It is stored as given and read back unchanged by `ST_CRS`.

Usage

```
sd_set_crs(geom = sd_missing_arg(), target_crs = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`target_crs`	(string): Input string

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SETCRS

sd_set_srid *Sets the SRID (spatial reference identifier) for a geometry.*

Description

Sets the SRID (spatial reference system identifier) of the geometry. This is metadata only - no coordinate transformation occurs. Use ST_Transform to actually reproject coordinates. See Coordinate Reference Systems for how SedonaDB represents and compares CRS.

Usage

```
sd_set_srid(geom = sd_missing_arg(), srid = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`srid`	(integer): Input integer

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SETSRID

`sd_shortest_line` *Returns the 2-dimensional shortest line between two geometries or geographies.*

Description

Returns the 2-dimensional shortest line between two geometries or geographies.

Usage

```
sd_shortest_line(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SHORTESTLINE

`sd_simplify` *Simplifies an input geometry or geography using the Douglas-Peucker algorithm.*

Description

Simplifies a geometry or geography using the Douglas-Peucker algorithm. The `tolerance` parameter controls the degree of simplification: higher values produce simpler geometries. For geography, the tolerance is specified in meters and a different algorithm is used. This function may produce invalid geometries; use `ST_SimplifyPreserveTopology` when validity must be maintained.

Usage

```
sd_simplify(geom = sd_missing_arg(), tolerance = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`tolerance`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SIMPLIFY

`sd_simplify_preserve_topology` *Simplifies a geometry, ensuring the result is a valid geometry with the same topology.*

Description

Simplifies a geometry, ensuring that the result is a valid geometry having the same dimension and number of components as the input, and with the components having the same topological relationship.

Usage

```
sd_simplify_preserve_topology(
  geom = sd_missing_arg(),
  tolerance = sd_missing_arg()
)
```

Arguments

`geom`	(geometry): Input geometry
`tolerance`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SIMPLIFYPRESERVETOPOLOGY

sd_snap — *Snaps input geometry to reference geometry within tolerance.*

Description

Snaps input geometry to reference geometry within tolerance. Result geometry aligns to the reference geometry where within tolerance distance.

Usage

```
sd_snap(
  geom_a = sd_missing_arg(),
  geom_b = sd_missing_arg(),
  tolerance = sd_missing_arg()
)
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry
tolerance	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SNAP

sd_srid — *Returns the SRID (spatial reference identifier) of a geometry.*

Description

Returns the SRID (spatial reference identifier) of a geometry.

Usage

```
sd_srid(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_SRID

`sd_start_point` *Returns the start point of a linestring geometry.*

Description

Returns the start point of a linestring geometry.

Usage

```
sd_start_point(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_STARTPOINT

`sd_sym_difference` *Returns the parts of geometries or geographies A and B that do not overlap.*

Description

Returns the parts of geometries or geographies A and B that do not overlap.

Usage

```
sd_sym_difference(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_SYMDIFFERENCE

`sd_tessellate_geog` *Converts a geometry to geography, densifying edges to approximate planar lines as geodesic segments.*

Description

Converts a geometry to a geography while tessellating edges using spherical (geodesic) interpolation. The tolerance parameter specifies the maximum deviation in meters between the original edge and the tessellated approximation. This is useful when converting planar geometries to geographies where edges need to follow geodesic paths on the sphere.

Usage

```
sd_tessellate_geog(geom = sd_missing_arg(), tolerance = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry
`tolerance`	(double): Input double

Value

(geography)

See Also

SedonaDB SQL documentation for ST_TESSELLATEGEOG

sd_tessellate_geom *Converts a geography to geometry, densifying edges to approximate geodesic segments as planar lines.*

Description

Converts a geography to a geometry while tessellating edges using spherical (geodesic) interpolation. The tolerance parameter specifies the maximum deviation in meters between the original spherical edge and the tessellated planar approximation. This is useful when converting geographies to geometries while preserving the shape of geodesic edges as approximated line segments. When an edge crosses the antimeridian, the returned coordinates are "wrapped" such that the returned longitude is either greater than 180 or less than -180. This helps produce valid geometry in more cases (e.g., linestring or polygon crossing the antimeridian); however, it does not produce valid geometry in the case of a polygon ring that spans the full longitude range (e.g., Antarctica).

Usage

```
sd_tessellate_geom(geom = sd_missing_arg(), tolerance = sd_missing_arg())
```

Arguments

geom	(geography): Input geography
tolerance	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_TESSELLATEGEOM

sd_to_geography *Converts a geometry to a geography by changing the edge interpretation to spherical.*

Description

Converts a geometry to a geography by changing the edge interpretation from planar to spherical. This is a metadata-only operation that does not modify the underlying coordinate data. If the input is already a geography, it is returned unchanged.

Usage

```
sd_to_geography(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geography)

See Also

SedonaDB SQL documentation for ST_TOGEOGRAPHY

---

sd_to_geometry *Converts a geography to a geometry by changing the edge interpretation to planar.*

Description

Converts a geography to a geometry by changing the edge interpretation from spherical to planar. This is a metadata-only operation that does not modify the underlying coordinate data. If the input is already a geometry, it is returned unchanged.

Usage

```
sd_to_geometry(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_TOGEOMETRY

sd_touches — *Returns true if A touches B.*

Description

Returns `true` if the two geometries have at least one boundary point in common but no interior points in common.

Usage

```
sd_touches(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_TOUCHES

sd_transform — *Transforms a geometry from one coordinate reference system to another.*

Description

Transforms the coordinate reference system of a geometry between EPSG reference systems. If provided with 2 arguments, target_crs can be specified as an EPSG code (e.g., 'EPSG:4326') or as a PROJ string. See Coordinate Reference Systems for how SedonaDB represents and compares CRS. When 3 arguments are provided, a source_crs can be used to specify the source CRS in case the input geometry doesn't have an SRID defined. The source_crs takes precedence over the geometry's SRID. Geography input is also accepted; because geography is always defined on a geographic (longitude/latitude) CRS, the target_crs must be geographic as well.

Usage

```
sd_transform(...)
```

Arguments

...	Supported combinations: • geom (geometry), target_crs (string) • geom (geometry), source_crs (string), target_crs (string) • geom (geography), target_crs (string) • geom (geography), source_crs (string), target_crs (string)

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_TRANSFORM

---

sd_translate *Returns a geometry with coordinates translated by deltaX and deltaY.*

Description

Translates (shifts) a geometry by the given deltaX and deltaY offsets. A 3D variant accepts `deltaZ`.

Usage

```
sd_translate(
  geom = sd_missing_arg(),
  deltaX = sd_missing_arg(),
  deltaY = sd_missing_arg(),
  deltaZ = sd_missing_arg()
)
```

Arguments

`geom`	(geometry): Input geometry
`deltaX`	(double): Input double
`deltaY`	(double): Input double
`deltaZ`	(double): Input double

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_TRANSLATE

sd_unary_union *Returns a single geometry which is the union of all components.*

Description

Returns a single geometry which is the union of all components of the input geometry. Useful for computing the union of a set of geometries stored in a single geometry (e.g., a GEOMETRYCOLLECTION or MULTI* geometry).

Usage

```
sd_unary_union(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_UNARYUNION

sd_union *Returns a geometry or geography that represents the point set union of two geometries or geographies.*

Description

Returns a geometry or geography that represents the point set union of two geometries or geographies.

Usage

```
sd_union(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

geom_a	(geometry): Input geometry
geom_b	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_UNION

`sd_union_agg` *Returns a geometry that represents the point set union of all geometries.*

Description

Returns a geometry that represents the point set union of all geometries.

Usage

```
sd_union_agg(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(geometry)

See Also

SedonaDB SQL documentation for ST_UNION_AGG

`sd_within` *Returns true if A is completely inside B.*

Description

Returns `true` if geometry A is completely inside geometry B. This is the inverse of `ST_Contains`: `ST_Within(A, B)` is equivalent to `ST_Contains(B, A)`.

Usage

```
sd_within(geom_a = sd_missing_arg(), geom_b = sd_missing_arg())
```

Arguments

`geom_a`	(geometry): Input geometry
`geom_b`	(geometry): Input geometry

Value

(boolean)

See Also

SedonaDB SQL documentation for ST_WITHIN

`sd_x` *Returns the X coordinate (longitude for geography) of the point, or NULL if not available.*

Description

Returns NULL for non-point geometries or NULL input.

Usage

```
sd_x(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_X

`sd_x_max` *Returns the maximum X coordinate (longitude for geography) of a geometry or geography's bounding box.*

Description

Returns the maximum X coordinate (longitude for geography) of a geometry or geography's bounding box.

Usage

```
sd_x_max(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_XMAX

`sd_x_min` *Returns the minimum X coordinate (longitude for geography) of a geometry or geography's bounding box.*

Description

Returns the minimum X coordinate (longitude for geography) of a geometry or geography's bounding box.

Usage

```
sd_x_min(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_XMIN

`sd_y` *Returns the Y coordinate (latitude for geography) of the point, or NULL if not available.*

Description

Returns NULL for non-point geometries or NULL input.

Usage

```
sd_y(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_Y

`sd_y_max` *Returns the maximum Y coordinate (latitude for geography) of a geometry or geography's bounding box.*

Description

For geography, the maximum latitude is computed considering spherical geometry (e.g., the maximum latitude of a linestring that crosses the North Pole is 90).

Usage

```
sd_y_max(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_YMAX

`sd_y_min` *Returns the minimum Y coordinate (latitude for geography) of a geometry or geography's bounding box.*

Description

For geography, the minimum latitude is computed considering spherical geometry (e.g., the minimum latitude of a linestring that crosses the South Pole is -90).

Usage

```
sd_y_min(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_YMIN

---

sd_z *Returns the Z coordinate of the point, or NULL if not available.*

Description

Returns NULL if the geometry has no Z dimension or for non-point geometries.

Usage

```
sd_z(geom = sd_missing_arg())
```

Arguments

geom	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_Z

`sd_z_max` *Returns the maximum Z coordinate of a geometry or geography's bounding box.*

Description

Returns the maximum Z coordinate of a geometry or geography's bounding box.

Usage

```
sd_z_max(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_ZMAX

`sd_z_min` *Returns the minimum Z coordinate of a geometry or geography's bounding box.*

Description

Returns the minimum Z coordinate of a geometry or geography's bounding box.

Usage

```
sd_z_min(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(double)

See Also

SedonaDB SQL documentation for ST_ZMIN

`sd_zmflag` *Returns a code indicating the dimension of the coordinates in a geometry.*

Description

Returns a code indicating the coordinate dimension: - 0 = 2D (XY) - 1 = 3D with M coordinate (XYM) - 2 = 3D with Z coordinate (XYZ) - 3 = 4D (XYZM)

Usage

```
sd_zmflag(geom = sd_missing_arg())
```

Arguments

`geom`	(geometry): Input geometry

Value

(integer)

See Also

SedonaDB SQL documentation for ST_ZMFLAG

Index

rs_as_geo_tiff, 6
rs_as_raster, 7
rs_band_is_no_data, 8
rs_band_no_data_value, 9
rs_band_path, 9
rs_band_pixel_type, 10
rs_band_to_dim, 10
rs_clip, 11
rs_contains, 12
rs_convex_hull, 13
rs_crs, 13
rs_dim_names, 14
rs_dim_size, 15
rs_dim_to_band, 16
rs_envelope, 16
rs_example, 17
rs_from_gdal_raster, 17
rs_from_path, 18
rs_geo_reference, 18
rs_geo_transform, 19
rs_height, 20
rs_intersects, 20
rs_is_empty, 21
rs_make_empty_raster, 22
rs_meta_data, 23
rs_min_convex_hull, 23
rs_num_bands, 24
rs_num_dimensions, 25
rs_pixel_as_centroid, 25
rs_pixel_as_point, 26
rs_pixel_as_polygon, 27
rs_polygonize, 27
rs_raster_to_world_coord, 28
rs_raster_to_world_coord_x, 29
rs_raster_to_world_coord_y, 29
rs_replace_band_no_data_value, 30
rs_reproject_match, 31
rs_resample, 32
rs_rotation, 33
rs_scale_x, 33
rs_scale_y, 34
rs_set_band_no_data_value, 35
rs_set_crs, 35
rs_set_geo_reference, 36
rs_set_srid, 37
rs_set_value, 37
rs_shape, 38
rs_skew_x, 39
rs_skew_y, 39
rs_slice, 40
rs_slice_range, 41
rs_srid, 42
rs_stack, 42
rs_stack_aggr, 43
rs_summary_stats, 44
rs_summary_stats_all, 45
rs_tile, 45
rs_upper_left_x, 46
rs_upper_left_y, 47
rs_value, 48
rs_values, 48
rs_width, 49
rs_within, 50
rs_world_to_raster_coord, 51
rs_world_to_raster_coord_x, 51
rs_world_to_raster_coord_y, 52
rs_zonal_stats, 53
rs_zonal_stats_all, 54

sd_affine, 55
sd_analyze_agg, 56
sd_area, 56
sd_as_binary, 57
sd_as_ewkb, 58
sd_as_geo_json, 58
sd_as_text, 59
sd_azimuth, 59
sd_boundary, 60
sd_buffer, 60

sd_build_area, 61
sd_cell_id_from_point, 62
sd_centroid, 62
sd_closest_point, 63
sd_collect_agg, 63
sd_concave_hull, 64
sd_contains, 64
sd_convex_hull, 65
sd_convex_hull_agg, 66
sd_covered_by, 66
sd_covering_cell_ids, 67
sd_covers, 68
sd_crosses, 68
sd_crs, 69
sd_d_within, 69
sd_delaunay_triangles, 70
sd_difference, 71
sd_dimension, 71
sd_disjoint, 72
sd_distance, 72
sd_dump, 73
sd_end_point, 73
sd_envelope, 74
sd_envelope_agg, 74
sd_equals, 75
sd_exterior_ring, 76
sd_flip_coordinates, 76
sd_force2d, 77
sd_force3d, 77
sd_force3dm, 78
sd_force4d, 78
sd_geo_hash, 79
sd_geog_from_wkb, 80
sd_geog_from_wkt, 80
sd_geog_point, 81
sd_geom_coll_from_text, 81
sd_geom_from_ewkb, 82
sd_geom_from_ewkt, 82
sd_geom_from_wkb, 83
sd_geom_from_wkt, 83
sd_geometries, 84
sd_geometry_n, 85
sd_geometry_type, 85
sd_has_m, 86
sd_has_z, 86
sd_hausdorff_distance, 87
sd_interior_ring_n, 88
sd_intersection, 88
sd_intersection_agg, 89
sd_intersects, 89
sd_is_closed, 90
sd_is_collection, 90
sd_is_empty, 91
sd_is_polygon_ccw, 91
sd_is_polygon_cw, 92
sd_is_ring, 92
sd_is_simple, 93
sd_is_valid, 94
sd_is_valid_reason, 94
sd_knn, 95
sd_length, 96
sd_line_from_text, 96
sd_line_interpolate_point, 97
sd_line_locate_point, 97
sd_line_merge, 98
sd_line_string_from_wkb, 99
sd_line_substring, 99
sd_longest_line, 100
sd_m, 101
sd_m_line_from_text, 101
sd_m_max, 102
sd_m_min, 102
sd_m_point_from_text, 103
sd_m_poly_from_text, 103
sd_make_envelope, 104
sd_make_line, 105
sd_make_valid, 105
sd_max_distance, 106
sd_minimum_clearance, 107
sd_minimum_clearance_line, 107
sd_missing_arg, 108
sd_n_points, 108
sd_n_rings, 109
sd_normalize, 109
sd_num_geometries, 110
sd_num_interior_rings, 110
sd_num_points, 111
sd_overlaps, 111
sd_perimeter, 112
sd_point, 112
sd_point_from_text, 113
sd_point_from_wkb, 114
sd_point_m, 114
sd_point_n, 115
sd_point_on_surface, 115
sd_point_z, 116

sd_point_zm, 117
sd_points, 117
sd_polygon_from_text, 118
sd_polygonize, 119
sd_polygonize_agg, 119
sd_reduce_precision, 120
sd_relate, 120
sd_reverse, 121
sd_rotate, 122
sd_rotate_x, 122
sd_rotate_y, 123
sd_scale, 123
sd_segmentize, 124
sd_set_crs, 125
sd_set_srid, 125
sd_shortest_line, 126
sd_simplify, 126
sd_simplify_preserve_topology, 127
sd_snap, 128
sd_srid, 128
sd_start_point, 129
sd_sym_difference, 129
sd_tessellate_geog, 130
sd_tessellate_geom, 131
sd_to_geography, 131
sd_to_geometry, 132
sd_touches, 133
sd_transform, 133
sd_translate, 134
sd_unary_union, 135
sd_union, 135
sd_union_agg, 136
sd_within, 136
sd_x, 137
sd_x_max, 137
sd_x_min, 138
sd_y, 138
sd_y_max, 139
sd_y_min, 139
sd_z, 140
sd_z_max, 141
sd_z_min, 141
sd_zmflag, 142