

Package ‘duckspatial’

June 6, 2026

Type Package

Title R Interface to 'DuckDB' Database with Spatial Extension

Version 1.1.1

Description Fast & memory-efficient functions to analyze and manipulate large spatial data data sets. It leverages the fast analytical capabilities of 'DuckDB' and its spatial extension (see <https://duckdb.org/docs/stable/core_extensions/spatial/overview>) while maintaining compatibility with R's spatial data ecosystem to work with spatial vector data.

URL <https://cidree.github.io/duckspatial/>,
<https://github.com/Cidree/duckspatial>

BugReports <https://github.com/Cidree/duckspatial/issues>

License GPL (>= 3)

Depends R (>= 4.1.0)

Imports arrow, cli, DBI, dbplyr (>= 2.0.0), dplyr, duckdb (>= 1.5.1),
geoarrow, glue, jsonlite, lifecycle, nanoarrow, rlang, sf,
tibble, tools, units, uuid, withr, wk

Suggests areal, bench, duckdbfs, ggplot2 (>= 3.3.1), knitr, lwgeom,
patchwork, quadkeyr, quarto, rmarkdown, scales, terra, testthat
(>= 3.0.0)

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Adrián Cidre González [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-3310-3052>>),
Egor Kotov [aut] (ORCID: <<https://orcid.org/0000-0001-6690-5345>>),
Rafael H. M. Pereira [aut] (ORCID:
<<https://orcid.org/0000-0003-2125-7465>>)

Maintainer Adrián Cidre González <adrian.cidre@gmail.com>

Repository CRAN

Date/Publication 2026-06-06 11:50:02 UTC

Contents

as_duckspatial_df	3
as_nanoarrow_array_stream.duckspatial_df	5
collect.duckspatial_df	5
ddbbs_affine	6
ddbbs_as_format	8
ddbbs_as_points	10
ddbbs_bbox	12
ddbbs_binary_funs	14
ddbbs_boundary	17
ddbbs_buffer	19
ddbbs_build_area	21
ddbbs_centroid	23
ddbbs_compute	24
ddbbs_concave_hull	26
ddbbs_convex_hull	28
ddbbs_create_conn	30
ddbbs_create_schema	31
ddbbs_crs	32
ddbbs_drivers	34
ddbbs_drop_geometry	35
ddbbs_dump	36
ddbbs_endpoint_startpoint	37
ddbbs_envelope	39
ddbbs_exterior_ring	41
ddbbs_filter	43
ddbbs_flip	46
ddbbs_flip_coordinates	48
ddbbs_force_dim	49
ddbbs_generate_points	52
ddbbs_geometry_type	54
ddbbs_geom_col	55
ddbbs_geom_validation_funs	55
ddbbs_get_npoints	58
ddbbs_glimpse	60
ddbbs_has_dim	61
ddbbs_install	64
ddbbs_interpolate_aw	65
ddbbs_join	68
ddbbs_line_interpolate	71
ddbbs_line_merge	73
ddbbs_line_substring	75

ddbs_list_tables	77
ddbs_load	78
ddbs_locate	79
ddbs_make_envelope	81
ddbs_make_line	83
ddbs_make_polygon	84
ddbs_make_valid	86
ddbs_maximum_inscribed_circle	88
ddbs_measure_funs	90
ddbs_minimum_rotated_rectangle	95
ddbs_multi	97
ddbs_open_dataset	98
ddbs_options	100
ddbs_polygonize	101
ddbs_predicate	103
ddbs_quadkey	107
ddbs_read_meta	109
ddbs_read_table	110
ddbs_register_table	111
ddbs_remove_repeated_points	112
ddbs_rotate	114
ddbs_rotate_3d	116
ddbs_scale	118
ddbs_set_crs	120
ddbs_set_resources	122
ddbs_shear	123
ddbs_shift	125
ddbs_simplify	127
ddbs_sitrep	129
ddbs_stop_conn	129
ddbs_transform	130
ddbs_union_funs	132
ddbs_voronoi	135
ddbs_write_dataset	137
ddbs_write_table	140
ddbs_xy	141
is_duckspatial_df	144

Index**145**

Description

as_duckspatial_df() creates a lazy spatial data frame (duckspatial_df) from various inputs. When x is a table name (character) or an existing DuckDB table (tbl_duckdb_connection), the function creates a zero-copy representation of the data directly from the database without loading it into memory. This is the canonical way to "register" or wrap existing persistent spatial tables.

CRS Persistence: duckspatial reads native DuckDB 1.5.0+ CRS metadata and, for compatibility with files written by older versions of duckspatial, CRS metadata stored in column comments. DuckDB files saved in pre-1.5.0 format without duckspatial-managed comments will not have CRS information and will default to NA with a warning.

Usage

```
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)

## S3 method for class 'duckspatial_df'
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)

## S3 method for class 'sf'
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)

## S3 method for class 'tbl_duckdb_connection'
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)

## S3 method for class 'tbl_lazy'
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)

## S3 method for class 'character'
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)

## S3 method for class 'data.frame'
as_duckspatial_df(x, conn = NULL, crs = NULL, geom_col = NULL, ...)
```

Arguments

x	Object to convert (sf, tbl_lazy, data.frame, or table name)
conn	DuckDB connection (required for character table names)
crs	CRS object or string. Auto-detected from sf objects and persistent DuckDB tables.
geom_col	Geometry column name (default: "geom")
...	Additional arguments passed to methods:
	coords Character vector of length 2 for point ingestion
	wkt Character name of WKT column for ingestion
	remove Logical. If TRUE (default), coordinate/WKT columns are removed
	na.fail Logical. If TRUE (default), errors on missing values

Value

A duckspatial_df object

as_nanoarrow_array_stream.duckspatial_df
Convert a duckspatial_df to a nanoarrow_array_stream

Description

Convert a duckspatial_df to a nanoarrow_array_stream

Usage

```
as_nanoarrow_array_stream.duckspatial_df(x, ..., schema = NULL, native = FALSE)
```

Arguments

x	A duckspatial_df object
...	Additional arguments passed to downstream methods
schema	Optional target schema for the entire stream.
native	If TRUE, transforms WKB to a "Native" GeoArrow layout (e.g., Point, Polygon) using optimized Arrow-to-Arrow kernels. This layout is optimized for high-performance rendering in tools like Deck.GL.

Value

A nanoarrow_array_stream

collect.duckspatial_df
Collect a duckspatial_df with flexible output formats

Description

Materializes a lazy duckspatial_df object by executing the underlying DuckDB query. Supports multiple output formats.

Usage

```
## S3 method for class 'duckspatial_df'
collect(x, ..., as = NULL)

ddb_collect(x, ..., as = c("sf", "tibble", "raw", "geoarrow"))
```

Arguments

<code>x</code>	A duckspatial_df object
<code>...</code>	Additional arguments passed to collect
<code>as</code>	Output format. One of: " <code>sf</code> " (Default) Returns an sf object with sfc geometry " <code>tibble</code> " Returns a tibble with geometry column dropped (fastest) " <code>raw</code> " Returns a tibble with geometry as raw WKB bytes " <code>geoarrow</code> " Returns a tibble with geometry as geoarrow_vctr

Value

Collected data in the specified format

Data in the specified format

Examples

```
## Not run:
library(duckspatial)

# Load lazy spatial data
nc <- ddbb_open_dataset(system.file("shape/nc.shp", package = "sf"))

# Perform lazy operations
result <- nc |> dplyr::filter(AREA > 0.1)

# Collect to sf (default)
result_sf <- ddbb_collect(result)
plot(result_sf["AREA"])

# Collect as tibble without geometry (fast)
result_tbl <- ddbb_collect(result, as = "tibble")

# Collect with raw WKB bytes
result_raw <- ddbb_collect(result, as = "raw")

# Collect as geoarrow for Arrow workflows
result_ga <- ddbb_collect(result, as = "geoarrow")

## End(Not run)
```

ddbs_affine

Apply an affine transformation to geometries

Description

Applies an affine transformation to geometries using a 2x3 or 3x4 matrix.

Usage

```
ddbs_affine(
  x,
  matrix,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
matrix	<p>A numeric matrix specifying the transformation:</p> <ul style="list-style-type: none"> • 2x3: 2D transformation with rows <code>[a, b, xoff]</code> and <code>[d, e, yoff]</code> • 3x4: 3D transformation with rows <code>[a, b, c, xoff]</code>, <code>[d, e, f, yoff]</code>, and <code>[g, h, i, zoff]</code>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
argentina_ddbs <- ddbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## 2D translation (shift x by 1, y by 2)
mat_2d <- matrix(c(1, 0, 1,
                  0, 1, 2), nrow = 2, byrow = TRUE)
ddbs_affine(argentina_ddbs, mat_2d)

## 2D scaling (scale x by 2, y by 3)
mat_scale <- matrix(c(2, 0, 0,
                    0, 3, 0), nrow = 2, byrow = TRUE)
ddbs_affine(argentina_ddbs, mat_scale)

## 3D transformation
mat_3d <- matrix(c(1, 0, 0, 1,
                  0, 1, 0, 2,
                  0, 0, 1, 0), nrow = 3, byrow = TRUE)
ddbs_affine(argentina_ddbs, mat_3d)

## End(Not run)
```

ddbs_as_format

Convert geometries to standard interchange formats

Description

Convert spatial geometries to common interchange formats using DuckDB spatial serialization functions.

- `ddbs_as_text()` – Convert geometries to Well-Known Text (WKT)
- `ddbs_as_wkb()` – Convert geometries to Well-Known Binary (WKB)
- `ddbs_as_hexwkb()` – Convert geometries to hexadecimal Well-Known Binary (HEXWKB)
- `ddbs_as_geojson()` – Convert geometries to GeoJSON

Usage

```
dbs_as_text(x, conn = NULL)
```

```
dbs_as_wkb(x, conn = NULL)
```

```
dbs_as_hexwkb(x, conn = NULL)
```

```
dbs_as_geojson(x, conn = NULL)
```

Arguments

x	Input spatial data. Can be: <ul style="list-style-type: none">• A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>)• An <code>sf</code> object• A <code>tbl_lazy</code> from <code>dbplyr</code>• A character string naming a table/view in <code>conn</code> Data is returned from this object.
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.

Details

These functions are thin wrappers around DuckDB spatial serialization functions (`ST_AsText`, `ST_AsWKB`, `ST_AsHEXWKB`, and `ST_AsGeoJSON`).

They are useful for exporting geometries into widely supported formats for interoperability with external spatial tools, databases, and web services.

Value

Depending on the function:

- `dbs_as_text()` returns a character vector of WKT geometries
- `dbs_as_wkb()` returns a list of raw vectors (binary WKB)
- `dbs_as_hexwkb()` returns a character vector of HEXWKB strings
- `dbs_as_geojson()` returns a character vector of GeoJSON strings

Examples

```
## Not run:
library(duckspatial)

argentina_dbs <- dbs_open_dataset(
  system.file("spatial/argentina.geojson", package = "duckspatial")
)

dbs_as_text(argentina_dbs)
dbs_as_wkb(argentina_dbs)
```

```

ddbbs_as_hexwkb(argentina_ddbs)
ddbbs_as_geojson(argentina_ddbs)

## End(Not run)

```

ddbbs_as_points

Generate point geometries from coordinates

Description

Converts a data frame with coordinate columns into spatial point geometries.

Usage

```

ddbbs_as_points(
  x,
  coords = c("lon", "lat"),
  crs = "EPSG:4326",
  remove = TRUE,
  na.fail = TRUE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE,
  ...
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
coords	Character vector of length 2 specifying the names of the longitude and latitude columns (or X and Y coordinates). Defaults to <code>c("lon", "lat")</code> .
crs	Character or numeric. The Coordinate Reference System (CRS) of the input coordinates. Can be specified as an EPSG code (e.g., "EPSG:4326" or 4326) or a WKT string. Defaults to "EPSG:4326" (WGS84 longitude/latitude).
remove	Logical. If TRUE (default), the coordinate columns specified in <code>coords</code> are removed from the output.
na.fail	Logical. If TRUE (default), the function errors if any missing values (NAs) are found in the coordinate columns.

conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
...	Additional arguments. Currently supports <code>geom_col</code> to specify the name of the geometry column in the output.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)

## create sample data with coordinates
cities_df <- data.frame(
  city = c("Buenos Aires", "Córdoba", "Rosario"),
  lon = c(-58.3816, -64.1811, -60.6393),
  lat = c(-34.6037, -31.4201, -32.9468),
  population = c(3075000, 1391000, 1193605)
)

# option 1: convert data frame to sf object
cities_ddbs <- ddbs_as_points(cities_df)

# specify custom coordinate column names and keep them in output
cities_df2 <- data.frame(
  city = c("Mendoza", "Tucumán"),
  longitude = c(-68.8272, -65.2226),
```

```
  latitude = c(-32.8895, -26.8241)
)

ddbbs_as_points(cities_df2, coords = c("longitude", "latitude"), remove = FALSE)

## option 2: convert table in duckdb to spatial table

# create a duckdb connection and write data
conn <- duckspatial::ddbbs_create_conn()
DBI::dbWriteTable(conn, "cities_tbl", cities_df, overwrite = TRUE)

# convert to spatial table in database
ddbbs_as_points(
  x = "cities_tbl",
  conn = conn,
  name = "cities_spatial",
  overwrite = TRUE
)

# read the spatial table
ddbbs_read_table(conn, "cities_spatial")

## End(Not run)
```

ddbbs_bbox

Get the bounding box of geometries

Description

Returns the minimal rectangle that encloses the geometry

Usage

```
ddbbs_bbox(
  x,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x Input spatial data. Can be:

- A `duckspatial_df` object (lazy spatial data frame via `dbplyr`)
- An `sf` object

- A `tbl_lazy` from `dbplyr`
- A character string naming a table/view in `conn`

Data is returned from this object.

<code>by_feature</code>	Logical. If <code>TRUE</code> , the geometric operation is applied separately to each geometry. If <code>FALSE</code> , the geometric operation is applied to the data as a whole.
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	Character. Controls the return type. Options: <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
<code>quiet</code>	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

A `bbox` numeric vector with `by_feature = FALSE` A data frame or lazy `tbl` when `by_feature = TRUE`

Examples

```
## Not run:
## load packages
library(duckspatial)

## read data
argentina_ddbs <- ddbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

# option 1: passing sf objects
ddbs_bbox(argentina_ddbs)

## option 2: passing the names of tables in a duckdb db

# creates a duckdb write sf to it
conn <- duckspatial::ddbs_create_conn()
ddbs_write_table(conn, argentina_ddbs, "argentina_tbl", overwrite = TRUE)

output2 <- ddbs_bbox(
  conn = conn,
```

```
    x = "argentina_tbl",
    name = "argentina_bbox"
)

DBI::dbReadTable(conn, "argentina_bbox")

## End(Not run)
```

ddbs_binary_funs

Geometry binary operations

Description

Perform geometric set operations between two sets of geometries.

Usage

```
ddbs_intersection(
  x,
  y,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

```
ddbs_difference(
  x,
  y,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

```
ddbs_sym_difference(
  x,
  y,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
```

```

    name = NULL,
    mode = NULL,
    overwrite = FALSE,
    quiet = FALSE
  )

```

```

ddbs_crop(
  x,
  y,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
y	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
conn_x	A <code>DBIConnection</code> object to a DuckDB database for the input <code>x</code> . If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>x</code> .
conn_y	A <code>DBIConnection</code> object to a DuckDB database for the input <code>y</code> . If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>y</code> .
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>

overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

These functions perform different geometric set operations:

`dbs_intersection` Returns the geometric intersection of two sets of geometries, producing the area, line, or point shared by both.

`dbs_crop` Returns the geometric intersection of two sets of geometries, using the bounding box of `y`, rather than its original geometry

`dbs_difference` Returns the portion of the first geometry that does not overlap with the second geometry.

`dbs_sym_difference` Returns the portions of both geometries that do not overlap with each other. Equivalent to $(A - B) \cup (B - A)$.

Value

Depends on the mode argument (or global preference set by `dbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
library(duckspatial)
library(sf)

# Create two overlapping polygons for testing
poly1 <- st_polygon(list(matrix(c(
  0, 0,
  4, 0,
  4, 4,
  0, 4,
  0, 0
), ncol = 2, byrow = TRUE)))

poly2 <- st_polygon(list(matrix(c(
  2, 2,
  6, 2,
  6, 6,
  2, 6,
  2, 2

```

```

), ncol = 2, byrow = TRUE))

x <- st_sf(id = 1, geometry = st_sfc(poly1), crs = 4326)
y <- st_sf(id = 2, geometry = st_sfc(poly2), crs = 4326)

# Visualize the input polygons
plot(st_geometry(x), col = "lightblue", main = "Input Polygons")
plot(st_geometry(y), col = "lightcoral", add = TRUE, alpha = 0.5)

# Intersection: only the overlapping area (2,2 to 4,4)
result_intersect <- ddbb_intersection(x, y)
plot(st_geometry(result_intersect), col = "purple",
     main = "Intersection")

# Difference: part of x not in y (L-shaped area)
result_diff <- ddbb_difference(x, y)
plot(st_geometry(result_diff), col = "lightblue",
     main = "Difference (x - y)")

# Symmetric Difference: parts of both that don't overlap
result_symdiff <- ddbb_sym_difference(x, y)
plot(st_geometry(result_symdiff), col = "orange",
     main = "Symmetric Difference")

# Using with database connection
conn <- ddbb_create_conn(dbdir = "memory")

ddbb_write_vector(conn, x, "poly_x")
ddbb_write_vector(conn, y, "poly_y")

# Perform operations with connection
ddbb_intersection("poly_x", "poly_y", conn)
ddbb_difference("poly_x", "poly_y", conn)
ddbb_sym_difference("poly_x", "poly_y", conn)

# Save results to database table
ddbb_difference("poly_x", "poly_y", conn, name = "diff_result")

## End(Not run)

```

ddbs_boundary

Get the boundary of geometries

Description

Returns the boundary of geometries as a new geometry, e.g., the edges of polygons or the start/end points of lines.

Usage

```
ddbbs_boundary(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

# read data
argentina_ddbs <- ddbb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

# store in duckdb
ddbb_write_table(conn, argentina_ddbs, "argentina")

# boundary
b <- ddbb_boundary(x = "argentina", conn)

## End(Not run)
```

ddbs_buffer

Creates a buffer around geometries

Description

Computes a polygon that represents all locations within a specified distance from the original geometry

Usage

```
ddbs_buffer(
  x,
  distance,
  num_triangles = 8L,
  cap_style = "CAP_ROUND",
  join_style = "JOIN_ROUND",
  mitre_limit = 1,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>distance</code>	a numeric value specifying the buffer distance. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)
<code>num_triangles</code>	an integer representing how many triangles will be produced to approximate a quarter circle. The larger the number, the smoother the resulting geometry. Default is 8.
<code>cap_style</code>	a character string specifying the cap style. Must be one of "CAP_ROUND" (default), "CAP_FLAT", or "CAP_SQUARE". Case-insensitive.
<code>join_style</code>	a character string specifying the join style. Must be one of "JOIN_ROUND" (default), "JOIN_MITRE", or "JOIN_BEVEL". Case-insensitive.
<code>mitre_limit</code>	a numeric value specifying the mitre limit ratio. Only applies when <code>join_style</code> is "JOIN_MITRE". It is the ratio of the distance from the corner to the mitre point to the corner radius. Default is 1.0.
<code>conn</code>	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when <code>name</code> is NULL.
<code>quiet</code>	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```

## Not run:
## load package
library(duckspatial)

## create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbb_write_vector(conn, argentina_ddbs, "argentina")

## basic buffer
ddbb_buffer(conn = conn, "argentina", distance = 1)

## buffer with custom parameters
ddbb_buffer(conn = conn, "argentina", distance = 1,
  num_triangles = 16, cap_style = "CAP_SQUARE")

## buffer without using a connection
ddbb_buffer(argentina_ddbs, distance = 1)

## End(Not run)

```

ddbb_build_area	<i>Build polygon areas from multiple linestrings</i>
-----------------	--

Description

Constructs polygon or multipolygon geometries from a collection of linestrings, handling intersections and creating unified areas. Returns POLYGON or MULTIPOLYGON (not wrapped in a geometry collection). Requires MULTILINESTRING input - for single linestrings, use `ddbb_make_polygon()`.

Usage

```

ddbb_build_area(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

See Also

[ddbbs_make_polygon\(\)](#), [ddbbs_polygonize\(\)](#)

Other polygon construction: [ddbbs_make_polygon\(\)](#), [ddbbs_polygonize\(\)](#)

ddbs_centroid	<i>Calculates the centroid of geometries</i>
---------------	--

Description

Returns the geometric center (centroid) of a geometry as a point, representing its average position.

Usage

```
ddbs_centroid(
  x,
  method = "centroid",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
method	<p>Character string specifying the method to calculate the centroid. Must be one of "centroid" (default) or "surface". "centroid" calculates the default centroid, which may fall outside the geometry for certain shapes (e.g., donuts). "surface" calculates a point guaranteed to fall within the geometry.</p>
conn	<p>A connection object to a DuckDB database. If <code>NULL</code>, the function runs on a temporary DuckDB database.</p>
name	<p>A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object</p>
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	<p>Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code>. This argument is ignored when <code>name</code> is <code>NULL</code>.</p>
quiet	<p>A logical value. If <code>TRUE</code>, suppresses any informational messages. Defaults to <code>FALSE</code>.</p>

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbbs_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbbs_write_vector(conn, argentina_ddbs, "argentina")

## centroid
ddbbs_centroid("argentina", conn)

## centroid without using a connection
ddbbs_centroid(argentina_ddbs)

## End(Not run)
```

`ddbbs_compute`

Force computation of a lazy `duckspatial_df`

Description

Executes the accumulated query and stores the result in a DuckDB temporary table. The result remains lazy (a `duckspatial_df`) but points to the materialized data, avoiding repeated computation of complex query plans.

Usage

```
ddbbs_compute(x, ..., name = NULL, temporary = TRUE)
```

Arguments

x	A duckspatial_df object
...	Additional arguments passed to dplyr::compute
name	Optional name for the result table. If NULL, a unique temporary name is generated.
temporary	If TRUE (default), creates a temporary table that is automatically cleaned up when the connection closes.

Details

This is useful when you want to:

- Cache intermediate results for reuse across multiple subsequent operations
- Simplify complex query plans before heavy operations like spatial joins
- Force execution at a specific point without pulling data into R memory

Value

A new duckspatial_df pointing to the materialized table

Examples

```
## Not run:
library(duckspatial)
library(dplyr)

# Load lazy spatial data
countries <- ddbb_open_dataset(
  system.file("spatial/countries.geojson", package = "duckspatial")
)

# Complex pipeline - ddbb_compute() caches intermediate result
cached <- countries |>
  filter(CNTR_ID %in% c("DE", "FR", "IT")) |>
  ddbb_compute() # Execute and store in temp table

# Check query plan - should reference temp table
show_query(cached)

# Further operations continue from cached result
result <- cached |>
  ddbb_filter(other_layer, predicate = "intersects") |>
  st_as_sf()

## End(Not run)
```

ddbbs_concave_hull *Compute the concave hull of geometries*

Description

Returns the concave hull that tightly encloses the geometry, capturing its overall shape more closely than a convex hull.

Usage

```
ddbbs_concave_hull(
  x,
  ratio = 0.5,
  allow_holes = TRUE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>ratio</code>	Numeric. The ratio parameter dictates the level of concavity; 1 returns the convex hull, while 0 indicates to return the most concave hull possible. Defaults to 0.5.
<code>allow_holes</code>	Boolean. If TRUE (the default), it allows the output to contain holes.
<code>conn</code>	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>

overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
library(sf)

# create points data
n <- 5
points_ddbs <- data.frame(
  id = 1,
  x = runif(n, min = -180, max = 180),
  y = runif(n, min = -90, max = 90)
) |>
  ddbb_as_spatial(coords = c("x", "y")) |>
  ddbb_combine()

# option 1: passing ddbb or sf objects
output1 <- duckspatial::ddbb_concave_hull(points_ddbs, mode = "sf")

plot(output1)

# option 2: passing the name of a table in a duckdb db

# creates a duckdb
conn <- duckspatial::ddbb_create_conn()

# write sf to duckdb
ddbb_write_vector(conn, points_ddbs, "points_tbl")

# spatial join
output2 <- duckspatial::ddbb_concave_hull(
  conn = conn,
  x = "points_tbl",
  mode = "sf"
```

```

)
plot(output2)

## End(Not run)

```

ddbbs_convex_hull *Compute the convex hull of geometries*

Description

Returns the convex hull that encloses the geometry, forming the smallest convex polygon that contains all points of the geometry.

Usage

```

ddbbs_convex_hull(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>

overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
library(sf)

# create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

# read data
argentina_ddbs <- ddb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

# option 1: passing sf objects
output1 <- duckspatial::ddb_convex_hull(x = argentina_ddbs, mode = "sf")

plot(output1["CNTR_NAME"])' # store in duckdb

# option 2: passing the name of a table in a duckdb db

# creates a duckdb
conn <- duckspatial::ddb_create_conn()

# write sf to duckdb
ddb_write_vector(conn, argentina_ddbs, "argentina_tbl")

# spatial join
output2 <- duckspatial::ddb_convex_hull(
  conn = conn,
  x = "argentina_tbl",
  mode = "sf"
)
```

```
plot(output2["CNTR_NAME"])

## End(Not run)
```

ddbbs_create_conn *Create a DuckDB connection with spatial extension*

Description

It creates a DuckDB connection, and then it installs and loads the spatial extension

Usage

```
ddbbs_create_conn(
  dbdir = "memory",
  threads = NULL,
  memory_limit_gb = NULL,
  upgrade = FALSE,
  ...,
  duckdb_storage_version = duckspatial_storage_default()
)
```

Arguments

dbdir	String. Either "tempdir", "memory", or a DuckDB database file path with .duckdb, .db, or .ddb extension. Defaults to "memory".
threads	Integer. Number of threads to use. If NULL (default), the setting is not changed, and DuckDB engine will use all available cores it detects (warning, on some shared HPC nodes the detected number of cores might be total number of cores on the node, not the per-job allocation).
memory_limit_gb	Numeric. Memory limit in GB. If NULL (default), the setting is not changed, and DuckDB engine will use 80% of available operating system memory it detects (warning, on some shared HPC nodes the detected memory might be the full node memory, not the per-job allocation).
upgrade	if TRUE, it upgrades the DuckDB extension to the latest version
...	Additional parameters to be passed to dbConnect
duckdb_storage_version	Storage compatibility for newly created persistent native DuckDB files (.duckdb, .db, .ddb). See https://duckdb.org/docs/internals/storage for more information on DuckDB storage versions and compatibility. <ul style="list-style-type: none"> "v1.5.0" (Native Spatial Storage, Default): Preserves CRS metadata in native DuckDB GEOMETRY columns. Requires DuckDB >= 1.5.0 to open the file.

- "v1.0.0" (**Legacy Compatibility**): Creates files readable by older DuckDB versions ($\geq 1.0.0$). Persists CRS metadata in duckspatial-managed column comments (a convention not recognized by other spatial software).
- "latest": Use the highest storage version supported by your installed DuckDB engine.

Other major version strings like "v1.4.0", "v1.3.0", etc., are also supported.

Value

A duckdb_connection

Examples

```
## Not run:
# load packages
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

# create a duckdb database in disk (with spatial extension)
conn <- ddb_create_conn(dbdir = "tempdir")

# create a connection with 1 thread and 2GB memory limit
conn <- ddb_create_conn(threads = 1, memory_limit_gb = 2)
ddb_stop_conn(conn)

## End(Not run)
```

ddbs_create_schema *Check and create schema*

Description

Check and create schema

Usage

```
ddbs_create_schema(conn, name, quiet = FALSE)
```

Arguments

conn	A DBIConnection object to a DuckDB database
name	A character string with the name of the schema to be created
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

TRUE (invisibly) for successful schema creation

Examples

```
## load packages
## Not run:
library(duckspatial)
library(duckdb)

## connect to in memory database
conn <- ddbs_create_conn(dbdir = "memory")

## create a new schema
ddbs_create_schema(conn, "new_schema")

## check schemas
dbGetQuery(conn, "SELECT * FROM information_schema.schemata;")

## disconnect from db
ddbs_stop_conn(conn)

## End(Not run)
```

ddbs_crs

Check CRS of spatial objects or database tables

Description

This is an S3 generic that extracts CRS information from various spatial objects.

Usage

```
ddbs_crs(x, ...)
```

S3 method for class 'duckspatial_df'

```
ddbs_crs(x, ...)
```

S3 method for class 'sf'

```
ddbs_crs(x, ...)
```

S3 method for class 'tbl_duckdb_connection'

```
ddbs_crs(x, ...)
```

S3 method for class 'character'

```
ddbs_crs(x, conn, ...)
```

S3 method for class 'duckdb_connection'

```

dbs_crs(x, name, ...)

## S3 method for class 'numeric'
dbs_crs(x, ...)

## S3 method for class 'crs'
dbs_crs(x, ...)

## S3 method for class 'data.frame'
dbs_crs(x, ...)

## Default S3 method:
dbs_crs(x, ...)

```

Arguments

x	An object containing spatial data. Can be: <ul style="list-style-type: none"> • duckspatial_df: Lazy spatial data frame (CRS from attributes) • sf: sf object (CRS from sf metadata) • character: Name of table in database (requires conn)
...	Additional arguments passed to methods
conn	A DuckDB connection (required for character method)
name	Table name (for backward compatibility when first arg is connection)

Value

CRS object from sf package

Examples

```

## Not run:
## load packages
library(duckdb)
library(duckspatial)
library(sf)

# Method 1: duckspatial_df objects
nc <- dbs_open_dataset(system.file("shape/nc.shp", package = "sf"))
dbs_crs(nc)

# Method 2: sf objects
nc_sf <- st_read(system.file("shape/nc.shp", package = "sf"))
dbs_crs(nc_sf)

# Method 3: table name in database
conn <- dbs_create_conn(dbdir = "memory")
dbs_write_table(conn, nc_sf, "nc_table")
dbs_crs(conn, "nc_table")
dbs_stop_conn(conn)

```

```
## End(Not run)
```

ddbbs_drivers	<i>Get list of GDAL drivers and file formats</i>
---------------	--

Description

Get list of GDAL drivers and file formats

Usage

```
ddbbs_drivers(conn = NULL)
```

Arguments

conn	A DBIConnection object to a DuckDB database. If not specified (conn = NULL), uses the default connection created with <code>ddbbs_default_conn()</code> (or a temporary one).
------	---

Value

data.frame

Examples

```
## Not run:  
## load package  
library(duckspatial)  
  
## database setup  
conn <- ddbbs_create_conn()  
  
## check drivers  
ddbbs_drivers(conn)  
  
## End(Not run)
```

ddbs_drop_geometry *Drop geometry column from a duckspatial_df object*

Description

Removes the geometry column from a duckspatial_df object, returning a lazy tibble without spatial information.

Usage

```
ddbs_drop_geometry(x)
```

Arguments

x Input spatial data. Can be:

- A duckspatial_df object (lazy spatial data frame via dbplyr)
- An sf object
- A tbl_lazy from dbplyr
- A character string naming a table/view in conn

Data is returned from this object.

Value

A lazy tibble backed by dbplyr

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
)

## drop geometry column
countries_tbl <- ddbs_drop_geometry(countries_ddbs)

## End(Not run)
```

 ddbbs_dump

Dumps geometries into their component parts

Description

Decomposes multi-part or complex geometries into individual simple geometry components, returning one row per component geometry

Usage

```
ddbbs_dump(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
rivers_ddbs <- ddbb_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## aggregate rivers by name
rivers_agg_ddbs <- ddbb_union_agg(rivers_ddbs, by = "RIVER_NAME")

## dump into individual geometries
ddbb_dump(rivers_agg_ddbs)

## End(Not run)
```

`ddbb_endpoint_startpoint`

Extract the start or end point of a linestring geometry

Description

Returns the first or last point of a `LINestring` geometry. These functions only work with `LINestring` geometries (not `MULTILINestring` or other geometry types).

Usage

```
ddbb_line_startpoint(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
```

```

)

ddbbs_line_endpoint(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Details

These functions wrap DuckDB Spatial's `ST_StartPoint` and `ST_EndPoint`. Input geometries must be of type `LINestring` (`MULTILINESTRING` is not supported). For each input feature, the first or last coordinate of the `LINestring` is returned as a `POINT` geometry.

Value

Depends on the `mode` argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.

- sf: An eagerly collected object in R memory, that will return the same data type as the sf equivalent (e.g. sf or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

## read data
rivers_ddbs <- ddbb_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbb_write_vector(conn, rivers_ddbs, "rivers")

## extract start points
ddbb_line_startpoint(conn = conn, "rivers")

## extract end points
ddbb_line_endpoint(conn = conn, "rivers")

## without using a connection
ddbb_line_startpoint(rivers_ddbs)
ddbb_line_endpoint(rivers_ddbs)

## End(Not run)
```

ddbs_envelope

Get the envelope (bounding box) of geometries

Description

Returns the minimum axis-aligned rectangle that fully contains the geometry.

Usage

```
ddbb_envelope(
  x,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
```

```

mode = NULL,
overwrite = FALSE,
quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
by_feature	Logical. If <code>TRUE</code> , the geometric operation is applied separately to each geometry. If <code>FALSE</code> , the geometric operation is applied to the data as a whole.
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>dbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Details

`ST_Envelope` returns the minimum bounding rectangle (MBR) of a geometry as a polygon. For points and lines, this creates a rectangular polygon that encompasses the geometry. For polygons, it returns the smallest rectangle that contains the entire polygon.

When `by_feature = FALSE`, all geometries are combined and a single envelope is returned that encompasses the entire dataset.

Value

Depends on the `mode` argument (or global preference set by `dbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)

# read data
argentina_ddbs <- ddbb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

# input as sf, and output as sf
env <- ddbb_envelope(x = argentina_ddbs, by_feature = TRUE)

# create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

# store in duckdb
ddbb_write_table(conn, argentina_ddbs, "argentina")

# envelope for each feature
env <- ddbb_envelope("argentina", conn, by_feature = TRUE)

# single envelope for entire dataset
env_all <- ddbb_envelope("argentina", conn, by_feature = FALSE)

# create a new table with envelopes
ddbb_envelope("argentina", conn, name = "argentina_bbox", by_feature = TRUE)

## End(Not run)
```

ddbs_exterior_ring *Extract the exterior ring of polygons*

Description

Returns the outer boundary (exterior ring) of polygon geometries. For multi-polygons, returns the exterior ring of each individual polygon.

Usage

```
ddbs_exterior_ring(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbs_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
```

```

)

## store in duckdb
ddbs_write_vector(conn, countries_ddbs, "countries")

## extract exterior ring
ddbs_exterior_ring(conn = conn, "countries")

## extract exterior ring without using a connection
ddbs_exterior_ring(countries_ddbs)

## End(Not run)

```

ddbs_filter

Perform a spatial filter

Description

Filters geometries based on a spatial relationship with another geometry, such as intersection, containment, or proximity.

Usage

```

ddbs_filter(
  x,
  y,
  predicate = "intersects",
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  distance = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	Input spatial data. Can be: <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn Data is returned from this object.
y	Input spatial data. Can be:

	<ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code>
<code>predicate</code>	A geometry predicate function. Defaults to <code>intersects</code> , a wrapper of <code>ST_Intersects</code> . See details for other options.
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>conn_x</code>	A <code>DBIConnection</code> object to a DuckDB database for the input <code>x</code> . If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>x</code> .
<code>conn_y</code>	A <code>DBIConnection</code> object to a DuckDB database for the input <code>y</code> . If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>y</code> .
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>distance</code>	a numeric value specifying the distance for <code>ST_DWithin</code> . The units should be specified in meters
<code>mode</code>	Character. Controls the return type. Options: <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
<code>quiet</code>	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Details

Spatial Join Predicates:

A spatial predicate is really just a function that evaluates some spatial relation between two geometries and returns true or false, e.g., “does a contain b” or “is a within distance x of b”. Here is a quick overview of the most commonly used ones, taking two geometries a and b:

- `"ST_Intersects"`: Whether a intersects b
- `"ST_Contains"`: Whether a contains b
- `"ST_ContainsProperly"`: Whether a contains b without b touching a’s boundary
- `"ST_Within"`: Whether a is within b
- `"ST_Overlaps"`: Whether a overlaps b
- `"ST_Touches"`: Whether a touches b
- `"ST_Equals"`: Whether a is equal to b
- `"ST_Crosses"`: Whether a crosses b
- `"ST_Covers"`: Whether a covers b
- `"ST_CoveredBy"`: Whether a is covered by b
- `"ST_DWithin"`: x) Whether a is within distance x of b

Value

Depends on the mode argument (or global preference set by `dbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
# RECOMMENDED: Efficient lazy workflow using dbs_open_dataset
library(duckspatial)

# Load data directly as lazy spatial data frames (CRS auto-detected)
countries <- dbs_open_dataset(
  system.file("spatial/countries.geojson", package = "duckspatial")
)

argentina <- dbs_open_dataset(
  system.file("spatial/argentina.geojson", package = "duckspatial")
)

# Lazy filter - computation stays in DuckDB
neighbors <- dbs_filter(countries, argentina, predicate = "touches")

# Collect to sf when needed
neighbors_sf <- dplyr::collect(neighbors) |> sf::st_as_sf()

# Alternative: using sf objects directly (legacy compatibility)
library(sf)

countries_sf <- st_read(system.file("spatial/countries.geojson", package = "duckspatial"))
argentina_sf <- st_read(system.file("spatial/argentina.geojson", package = "duckspatial"))

result <- dbs_filter(countries_sf, argentina_sf, predicate = "touches")

# Alternative: using table names in a duckdb connection
conn <- dbs_create_conn(dbdir = "memory")

dbs_write_table(conn, countries_sf, "countries")
dbs_write_table(conn, argentina_sf, "argentina")

dbs_filter(conn = conn, "countries", "argentina", predicate = "touches")

## End(Not run)
```

dbs_flip

*Flip geometries horizontally or vertically***Description**

Reflects geometries across their centroid. By default, flipping is applied relative to the centroid of all geometries; if `by_feature = TRUE`, each geometry is flipped relative to its own centroid.

Usage

```
dbs_flip(
  x,
  direction = c("horizontal", "vertical"),
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>direction</code>	character string specifying the flip direction: "horizontal" (default) or "vertical". Horizontal flips across the Y-axis (left-right), vertical flips across the X-axis (top-bottom)
<code>by_feature</code>	Logical. If <code>TRUE</code> , the geometric operation is applied separately to each geometry. If <code>FALSE</code> , the geometric operation is applied to the data as a whole.
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>dbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>

overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by [ddbs_options](#)):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddb_write_table(conn, argentina_ddbs, "argentina")

## flip all features together as a whole (default)
ddb_flip(conn = conn, "argentina", direction = "horizontal", by_feature = FALSE)

## flip each feature independently
ddb_flip(conn = conn, "argentina", direction = "horizontal", by_feature = TRUE)

## flip without using a connection
ddb_flip(argentina_ddbs, direction = "horizontal")

## End(Not run)
```

ddbbs_flip_coordinates *Flips the X and Y coordinates of geometries*

Description

Returns a geometry with the X and Y coordinates swapped. This is useful for correcting geometries where longitude and latitude are in the wrong order, or for converting between coordinate systems with different axis orders.

Usage

```
ddbbs_flip_coordinates(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbb_write_vector(conn, argentina_ddbs, "argentina")

## flip coordinates
ddbb_flip_coordinates("argentina", conn)

## flip coordinates without using a connection
ddbb_flip_coordinates(argentina_ddbs)

## End(Not run)
```

<code>ddbb_force_dim</code>	<i>Force geometry dimensions</i>
-----------------------------	----------------------------------

Description

Functions to force geometries to have specific coordinate dimensions (X, Y, Z, M)

Usage

```
ddbb_force_2d(
  x,
  conn = NULL,
  name = NULL,
```

```

mode = NULL,
overwrite = FALSE,
quiet = FALSE
)

```

```

ddbs_force_3d(
  x,
  var,
  dim = "z",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

```

ddbs_force_4d(
  x,
  var_z,
  var_m,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

- | | |
|------|---|
| x | <p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p> |
| conn | A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database. |
| name | A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object |
| mode | <p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p> |

overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
var	A numeric variable in x to be converted in the dimension specified in the argument dim
dim	The dimension to add: either "z" (default) for elevation or "m" for measure values.
var_z	A numeric variable in x to be converted in Z dimension
var_m	A numeric variable in x to be converted in M dimension

Details

These functions modify the dimensionality of geometries:

- `ddbs_force_2d()` removes Z and M coordinates from geometries, returning only X and Y coordinates. This is useful for simplifying 3D or measured geometries to 2D.
- `ddbs_force_3d()` forces geometries to have three dimensions. When `dim = "z"` (default), adds or retains Z coordinates (X, Y, Z). When `dim = "m"`, adds or retains M coordinates (X, Y, M). Missing values are typically set to 0. If the input geometry has a third dimension already, it will be replaced by the new one. If the input geometry has 4 dimensions, it will drop the dimension that wasn't specified.
- `ddbs_force_4d()` forces geometries to have all four dimensions (X, Y, Z, M). Missing Z or M values are typically set to 0.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(dplyr)
library(duckspatial)

## load data and add 2 numeric vars
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
dplyr::filter(IS03_CODE != "ATA") |>
```

```

ddbs_area(new_column = "area") |>
ddbs_perimeter(new_column = "perim")

## add a Z dimension
countries_z_ddbs <- ddbs_force_3d(countries_ddbs, "area")
ddbs_has_z(countries_z_ddbs)

## add a M dimension as 3D (removes current Z)
countries_m_ddbs <- ddbs_force_3d(countries_z_ddbs, "area", "m")
ddbs_has_z(countries_m_ddbs)
ddbs_has_m(countries_m_ddbs)

## add both Z and M
countries_zm_ddbs <- ddbs_force_4d(countries_ddbs, "area", "perim")
ddbs_has_z(countries_zm_ddbs)
ddbs_has_m(countries_zm_ddbs)

## drop both ZM
countries_drop_ddbs <- ddbs_force_2d(countries_zm_ddbs)
ddbs_has_z(countries_drop_ddbs)
ddbs_has_m(countries_drop_ddbs)

## End(Not run)

```

ddbs_generate_points *Generate random points within bounding boxes of geometries*

Description

Creates random points within the bounding box of each geometry, which may fall outside the geometry itself.

Usage

```

ddbs_generate_points(
  x,
  n,
  seed = NULL,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x Input spatial data. Can be:

- A `duckspatial_df` object (lazy spatial data frame via `dbplyr`)
- An `sf` object
- A `tbl_lazy` from `dbplyr`
- A character string naming a table/view in `conn`

Data is returned from this object.

<code>n</code>	Number of random points to generate within each geometry
<code>seed</code>	A number for the random number generator
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	Character. Controls the return type. Options: <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
<code>quiet</code>	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbs_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)
```

```
## store in duckdb
ddbbs_write_table(conn, argentina_ddbs, "argentina")

## generate 100 random points within each geometry
ddbbs_generate_points("argentina", n = 100, conn)

## generate points without using a connection
ddbbs_generate_points(argentina_ddbs, n = 100)

## End(Not run)
```

ddbbs_geometry_type *Get the geometry type of features*

Description

Returns the type of each geometry (e.g., POINT, LINESTRING, POLYGON) in the input features.

Usage

```
ddbbs_geometry_type(x, by_feature = TRUE, conn = NULL)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn <p>Data is returned from this object.</p>
by_feature	<p>Logical. If TRUE, the geometric operation is applied separately to each geometry. If FALSE, the geometric operation is applied to the data as a whole.</p>
conn	<p>A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.</p>

Value

A factor with geometry type(s)

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
```

```
countries_ddbs <- ddbb_open_dataset(  
  system.file("spatial/countries.geojson",  
  package = "duckspatial")  
)  
  
# option 1: passing sf objects  
# Get geometry type for each feature  
ddbs_geometry_type(countries_ddbs)  
  
# Get overall geometry type  
ddbs_geometry_type(countries_ddbs, by_feature = FALSE)  
  
## End(Not run)
```

ddbs_geom_col	<i>Get the geometry column name</i>
---------------	-------------------------------------

Description

Get the geometry column name

Usage

```
ddbs_geom_col(x)
```

Arguments

x A duckspatial_df object

Value

Character string with geometry column name

ddbs_geom_validation_funs	<i>Geometry validation functions</i>
---------------------------	--------------------------------------

Description

Functions to check various geometric properties and validity conditions of spatial geometries using DuckDB's spatial extension.

Usage

```
ddbs_is_simple(  
  x,  
  new_column = "is_simple",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbs_is_valid(  
  x,  
  new_column = "is_valid",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbs_is_closed(  
  x,  
  new_column = "is_closed",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbs_is_empty(  
  x,  
  new_column = "is_empty",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbs_is_ring(  
  x,  
  new_column = "is_ring",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,
```

```

    quiet = FALSE
  )

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
new_column	Name of the new column to create on the input data. Ignored with <code>mode = "sf"</code> .
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr/DuckDB</code> • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Details

These functions provide different types of geometric validation. Note that by default, the functions add a new column as a logical vector. This behaviour allows to filter the data within DuckDB without the need or materializing a vector in R (see details).

- `ddbbs_is_valid()` checks if a geometry is valid according to the OGC Simple Features specification. Invalid geometries may have issues like self-intersections in polygons, duplicate points, or incorrect ring orientations.
- `ddbbs_is_simple()` determines whether geometries are simple, meaning they are free of self-intersections. For example, a linestring that crosses itself is not simple.
- `ddbbs_is_ring()` checks if a linestring geometry is closed (first and last points are identical) and simple (no self-intersections), forming a valid ring.
- `ddbbs_is_empty()` tests whether a geometry is empty, containing no points. Empty geometries are valid but represent the absence of spatial information.
- `ddbbs_is_closed()` determines if a linestring geometry is closed, meaning the first and last coordinates are identical. Unlike `ddbbs_is_ring()`, this does not check for simplicity.

Value

- mode = "duckspatial" (default): A duckspatial_df (lazy spatial data frame) backed by dbplyr/DuckDB.
- mode = "sf": An eagerly collected vector in R memory.
- When name is provided: writes the table in the DuckDB connection and returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
library(dplyr)

## create a duckdb database in memory (with spatial extension)
conn <- ddbs_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
)

rivers_ddbs <- ddbs_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## geometry validation
ddbs_is_valid(countries_ddbs)
ddbs_is_simple(countries_ddbs)
ddbs_is_ring(rivers_ddbs)
ddbs_is_empty(countries_ddbs)
ddbs_is_closed(countries_ddbs)

## filter invalid countries
ddbs_is_valid(countries_ddbs) |> filter(!is_valid)

## End(Not run)
```

ddbs_get_npoints

Count geometry components

Description

Functions to count the number of points or sub-geometries in a geometry

Usage

```

ddbbs_get_npoints(
  x,
  new_column = "npoints",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

ddbbs_get_ngeometries(
  x,
  new_column = "ngeometries",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn <p>Data is returned from this object.</p>
new_column	Name of the new column to create on the input data. Ignored with mode = "sf".
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

These functions query structural properties of geometries:

- `ddbbs_get_npoints()` returns the number of points (vertices) in a geometry. For `LINestring` geometries this is the vertex count; for `POLYGON` types it includes all vertices of the exterior ring and any interior rings.
- `ddbbs_get_ngeometries()` returns the number of sub-geometries in a `GEOMETRYCOLLECTION` or `MULTI*` geometry (e.g. `MULTIPOLYGON`, `MULTILINestring`). Returns 1 for simple (non-collection) geometry types.

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load packages
library(dplyr)
library(duckspatial)

## read data
countries_ddbs <- ddbbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
)

## count points and sub-geometries
ddbbs_get_npoints(countries_ddbs)
ddbbs_get_ngeometries(countries_ddbs)

## End(Not run)
```

```
ddbbs_glimpse
```

```
Check first rows of the data
```

Description

Prints a transposed table of the first rows of a DuckDB table, similarly as the S3 `dplyr::glimpse` method.

Usage

```
ddbs_glimpse(conn, name, quiet = FALSE)
```

Arguments

conn	A DBIConnection object to a DuckDB database
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Invisibly duckspatial_df object

Examples

```
## Not run:
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

## read data
argentina_sf <- ddb_open_dataset(system.file("spatial/argentina.geojson", package = "duckspatial"))

## store in duckdb
ddb_write_table(conn, argentina_sf, "argentina")

## glimpse the inserted table
ddb_glimpse(conn, "argentina")

## End(Not run)
```

ddbs_has_dim

Check geometry dimensions

Description

Functions to check whether geometries have Z (elevation) or M (measure) dimensions

Usage

```
ddbbs_has_z(
  x,
  by_feature = TRUE,
  new_column = "has_z",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

```
ddbbs_has_m(
  x,
  by_feature = TRUE,
  new_column = "has_m",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>by_feature</code>	Logical. If <code>TRUE</code> , the geometric operation is applied separately to each geometry. If <code>FALSE</code> , the geometric operation is applied to the data as a whole.
<code>new_column</code>	Name of the new column to create on the input data. Ignored with <code>mode = "sf"</code> .
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>

overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

These functions check for additional coordinate dimensions beyond X and Y:

- `ddbs_has_z()` checks if a geometry has Z coordinates (elevation/altitude values). Geometries with Z dimension are often referred to as 3D geometries and have coordinates in the form (X, Y, Z).
- `ddbs_has_m()` checks if a geometry has M coordinates (measure values). The M dimension typically represents a measurement along the geometry, such as distance or time, and results in coordinates of the form (X, Y, M) or (X, Y, Z, M).

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(dplyr)
library(duckspatial)

## create a duckdb database in memory (with spatial extension)
conn <- ddbs_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
  filter(IS03_CODE != "ATA")

## check if it has Z or M
ddbs_has_m(countries_ddbs)
ddbs_has_z(countries_ddbs)

## End(Not run)
```

ddbbs_install	<i>Checks and installs the Spatial extension</i>
---------------	--

Description

Checks if a spatial extension is available, and installs it in a DuckDB database

Usage

```
ddbbs_install(conn, upgrade = FALSE, quiet = FALSE, extension = "spatial")
```

Arguments

conn	A DBIConnection object to a DuckDB database
upgrade	if TRUE, it upgrades the DuckDB extension to the latest version
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
extension	name of the extension to install, default is "spatial"

Value

TRUE (invisibly) for successful installation

Examples

```
## load packages
library(duckspatial)
library(duckdb)

# connect to in memory database
conn <- duckdb::dbConnect(duckdb::duckdb())

# install the spatial extension
ddbbs_install(conn)

# install the h3 community extension
ddbbs_install(conn, extension = "h3")

# disconnect from db
duckdb::dbDisconnect(conn)
```

 ddbbs_interpolate_aw *Areal-Weighted Interpolation using DuckDB*

Description

Transfers attribute data from a source spatial layer to a target spatial layer based on the area of overlap between their geometries. This function executes all spatial calculations within DuckDB, enabling efficient processing of large datasets without loading all geometries into R memory.

Usage

```
ddbs_interpolate_aw(
  target,
  source,
  tid,
  sid,
  extensive = NULL,
  intensive = NULL,
  weight = "sum",
  mode = NULL,
  keep_NA = TRUE,
  na.rm = FALSE,
  join_crs = NULL,
  conn = NULL,
  name = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

target	An sf object or the name of a persistent table in the DuckDB connection representing the destination geometries.
source	An sf object or the name of a persistent table in the DuckDB connection containing the data to be interpolated.
tid	Character. The name of the column in target that uniquely identifies features.
sid	Character. The name of the column in source that uniquely identifies features.
extensive	Character vector. Names of columns in source to be treated as spatially extensive (e.g., population counts).
intensive	Character vector. Names of columns in source to be treated as spatially intensive (e.g., population density).
weight	Character. Determines the denominator calculation for extensive variables. Either "sum" (default) or "total". See Mass Preservation in Details.
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB

	<ul style="list-style-type: none"> • "sf": Eagerly collected sf object (uses memory) <p>Can be set globally via <code>dbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
keep_NA	Logical. If TRUE (default), returns all features from the target, even those that do not overlap with the source (values will be NA). If FALSE, performs an inner join, dropping non-overlapping target features.
na.rm	Logical. If TRUE, source features with NA values in the interpolated variables are completely removed from the calculation (area calculations will behave as if that polygon did not exist). Defaults to FALSE.
join_crs	Numeric or Character (optional). EPSG code or WKT for the CRS to use for area calculations. If provided, both target and source are transformed to this CRS within the database before interpolation.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

Areal-weighted interpolation is used when the source and target geometries are incongruent (they do not align). It relies on the assumption of **uniform distribution**: values in the source polygons are assumed to be spread evenly across the polygon's area.

Coordinate Systems: Area calculations are highly sensitive to the Coordinate Reference System (CRS). While the function can run on geographic coordinates (lon/lat), it is strongly recommended to use a **projected CRS** (e.g., EPSG:3857, UTM, or Albers) to ensure accurate area measurements. Use the `join_crs` argument to project data on-the-fly during the interpolation.

Extensive vs. Intensive Variables:

- **Extensive** variables are counts or absolute amounts (e.g., total population, number of voters). When a source polygon is split, the value is divided proportionally to the area.
- **Intensive** variables are ratios, rates, or densities (e.g., population density, cancer rates). When a source polygon is split, the value remains constant for each piece.

Mass Preservation (The weight argument): For extensive variables, the choice of weight determines the denominator used in calculations:

- "sum" (default): The denominator is the sum of all overlapping areas for that source feature. This preserves the "mass" of the variable *relative to the target's coverage*. If the target polygons do not completely cover a source polygon, some data is technically "lost" because it falls outside the target area. This matches `areal::aw_interpolate(weight="sum")`.

- "total": The denominator is the full geometric area of the source feature. This assumes the source value is distributed over the entire source polygon. If the target covers only 50% of the source, only 50% of the value is transferred. This is strictly mass-preserving relative to the source. This matches `sf::st_interpolate_aw(extensive=TRUE)`.

Note: Intensive variables are always calculated using the "sum" logic (averaging based on intersection areas) regardless of this parameter.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

References

Prener, C. and Revord, C. (2019). *areal: An R package for areal weighted interpolation*. *Journal of Open Source Software*, 4(37), 1221. Available at: [doi:10.21105/joss.01221](https://doi.org/10.21105/joss.01221)

See Also

`areal::aw_interpolate()` — reference implementation.

Examples

```
library(sf)

# 1. Prepare Data
# Load NC counties (Source) and project to Albers (EPSG:5070)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
nc <- st_transform(nc, 5070)
nc$tid <- seq_len(nrow(nc)) # Create Source ID

# Create a target grid
g <- st_make_grid(nc, n = c(10, 5))
g_sf <- st_as_sf(g)
g_sf$tid <- seq_len(nrow(g_sf)) # Create Target ID

# 2. Extensive Interpolation (Counts)
# Use weight = "total" for strict mass preservation (e.g., total births)
res_ext <- ddbb_interpolate_aw(
  target = g_sf, source = nc,
  tid = "tid", sid = "sid",
  extensive = "BIR74",
  weight = "total",
  mode = "sf"
)
```

```

# Check mass preservation
sum(res_ext$BIR74, na.rm = TRUE) / sum(nc$BIR74) # Should be ~1

# 3. Intensive Interpolation (Density/Rates)
# Calculates area-weighted average (e.g., assumption of uniform density)
res_int <- ddbbs_interpolate_aw(
  target = g_sf, source = nc,
  tid = "tid", sid = "sid",
  intensive = "BIR74",
  mode = "sf"
)

# 4. Quick Visualization
par(mfrow = c(1, 2))
plot(res_ext["BIR74"], main = "Extensive (Total Count)", border = NA)
plot(res_int["BIR74"], main = "Intensive (Weighted Avg)", border = NA)

```

ddbbs_join

Perform a spatial join of two geometries

Description

Combines two sets of geometries based on spatial relationships, such as intersection or containment, attaching attributes from one set to the other.

Usage

```

ddbbs_join(
  x,
  y,
  join = "intersects",
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  distance = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

- x Input spatial data. Can be:
- A `duckspatial_df` object (lazy spatial data frame via `dbplyr`)

	<ul style="list-style-type: none"> • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn
	Data is returned from this object.
y	Input spatial data. Can be: <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn
join	A geometry predicate function. Defaults to "intersects". See the details for other options.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
conn_x	A DBIConnection object to a DuckDB database for the input x. If NULL (default), it is resolved from conn or extracted from x.
conn_y	A DBIConnection object to a DuckDB database for the input y. If NULL (default), it is resolved from conn or extracted from y.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
distance	a numeric value specifying the distance for ST_DWithin. The units should be specified in meters
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

Spatial Join Predicates:

A spatial predicate is really just a function that evaluates some spatial relation between two geometries and returns true or false, e.g., "does a contain b" or "is a within distance x of b". Here is a quick overview of the most commonly used ones, taking two geometries a and b:

- "ST_Intersects": Whether a intersects b
- "ST_Contains": Whether a contains b
- "ST_ContainsProperly": Whether a contains b without b touching a's boundary
- "ST_Within": Whether a is within b

- "ST_Overlaps": Whether a overlaps b
- "ST_Touches": Whether a touches b
- "ST_Equals": Whether a is equal to b
- "ST_Crosses": Whether a crosses b
- "ST_Covers": Whether a covers b
- "ST_CoveredBy": Whether a is covered by b
- "ST_DWithin": x) Whether a is within distance x of b

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
# RECOMMENDED: Efficient lazy workflow using ddbs_open_dataset
library(duckspatial)

# Load data directly as lazy spatial data frames (CRS auto-detected)
countries <- ddbs_open_dataset(
  system.file("spatial/countries.geojson", package = "duckspatial")
)

# Create random points
n <- 100
points <- data.frame(
  id = 1:n,
  x = runif(n, min = -180, max = 180),
  y = runif(n, min = -90, max = 90)
) |>
sf::st_as_sf(coords = c("x", "y"), crs = 4326) |>
as_duckspatial_df()

# Lazy join - computation stays in DuckDB
result <- ddbs_join(points, countries, join = "within")

# Collect to sf when needed
result_sf <- dplyr::collect(result) |> sf::st_as_sf()
plot(result_sf["CNTR_NAME"])

# Alternative: using sf objects directly (legacy compatibility)
library(sf)
```

```
countries_sf <- sf::st_read(system.file("spatial/countries.geojson", package = "duckspatial"))

output <- duckspatial::ddbs_join(
  x = points,
  y = countries_sf,
  join = "within"
)

# Alternative: using table names in a duckdb connection
conn <- duckspatial::ddbs_create_conn()

ddbs_write_table(conn, points, "points", overwrite = TRUE)
ddbs_write_table(conn, countries_sf, "countries", overwrite = TRUE)

output2 <- ddbs_join(
  conn = conn,
  x = "points",
  y = "countries",
  join = "within"
)

## End(Not run)
```

`ddbs_line_interpolate` *Interpolates a point or points along a line geometry*

Description

Returns either a single point at a specified position along a line, or multiple equally-spaced points along a line, depending on the value of `intervals`. When `intervals = FALSE`, this wraps `ST_LineInterpolatePoint`; when `intervals = TRUE`, it wraps `ST_LineInterpolatePoints`.

Usage

```
ddbs_line_interpolate(
  x,
  fraction = 0.5,
  intervals = FALSE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>fraction</code>	a numeric value between 0 and 1. When <code>intervals = FALSE</code> , specifies the position along the line to interpolate, where 0 is the start and 1 is the end. When <code>intervals = TRUE</code> , specifies the spacing between interpolated points as a proportion of the total line length. Defaults to 0.5.
<code>intervals</code>	a logical value. If <code>FALSE</code> (default), returns a single <code>POINT</code> at the position given by <code>fraction</code> . If <code>TRUE</code> , returns a <code>MULTIPOINT</code> of equally-spaced points along the line at intervals defined by <code>fraction</code> .
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr/DuckDB</code> • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
<code>quiet</code>	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
```

```

## read data
rivers_ddbs <- ddbb_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## return the midpoint of a line (default)
ddbs_line_interpolate(rivers_ddbs)

## return the point 25% along the line
ddbs_line_interpolate(rivers_ddbs, fraction = 0.25)

## return equally-spaced points every 10% of the line length
ddbs_line_interpolate(rivers_ddbs, fraction = 0.1, intervals = TRUE)

## return equally-spaced points every 50% of the line length (i.e. midpoint and end)
ddbs_line_interpolate(rivers_ddbs, fraction = 0.5, intervals = TRUE)

## End(Not run)

```

ddbs_line_merge

Merge line geometries into a single line

Description

Merges a collection of line geometries that share endpoints into a single LINESTRING, or MULTILINESTRING if endpoints are not shared

Usage

```

ddbs_line_merge(
  x,
  preserve = TRUE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

- x Input spatial data. Can be:
- A duckspatial_df object (lazy spatial data frame via dbplyr)
 - An sf object
 - A tbl_lazy from dbplyr
 - A character string naming a table/view in conn

	Data is returned from this object.
preserve	a logical value. If TRUE (default), line direction is preserved
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
library(dplyr)

## read data
rivers_ddbs <- ddbbs_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## first, union by river name
rivers_union <- ddbbs_union_agg(rivers_ddbs, by = "RIVER_NAME")

## merge lines, preserving direction
rivers_merged <- ddbbs_line_merge(rivers_union)

## check Rio Eume (union doesn't guarantee the merging)
```

```
rivers_union |> filter(RIVER_NAME == "Rio Eume")
rivers_merged |> filter(RIVER_NAME == "Rio Eume")
```

```
## End(Not run)
```

ddbs_line_substring *Extract a substring of a line geometry*

Description

Returns the portion of a line between two fractional positions along its length

Usage

```
ddbs_line_substring(
  x,
  start = 0,
  end = 0.5,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn <p>Data is returned from this object.</p>
start	a numeric value between 0 and 1. Specifies the starting position along the line as a proportion of its total length, where 0 is the beginning of the line. Defaults to 0.
end	a numeric value between 0 and 1. Specifies the ending position along the line as a proportion of its total length, where 1 is the end of the line. Must be greater than or equal to start. Defaults to 0.5.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object

mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	<p>Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.</p>
quiet	<p>A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.</p>

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
rivers_ddbs <- ddbs_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## return the first half of each line (default)
dbs_line_substring(rivers_ddbs)

## return the middle third of each line
dbs_line_substring(rivers_ddbs, start = 0.33, end = 0.67)

## return the last quarter of each line
dbs_line_substring(rivers_ddbs, start = 0.75, end = 1)

## End(Not run)
```

ddbs_list_tables	<i>Check tables and schemas inside a database</i>
------------------	---

Description

Check tables and schemas inside a database

Usage

```
ddbs_list_tables(conn)
```

Arguments

conn A DBIConnection object to a DuckDB database

Value

data.frame

Examples

```
## Not run:
## load packages
library(duckspatial)

## create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

## read some data
countries_ddbs <- ddbb_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
)

argentina_ddbs <- ddbb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## insert into the database
ddbb_write_table(conn, argentina_ddbs, "argentina")
ddbb_write_table(conn, countries_ddbs, "countries")

## list tables in the database
ddbb_list_tables(conn)

## End(Not run)
```

ddbbs_load	<i>Loads the Spatial extension</i>
------------	------------------------------------

Description

Checks if a spatial extension is installed, and loads it in a DuckDB database

Usage

```
ddbbs_load(conn, quiet = FALSE, extension = "spatial", create_macros = TRUE)
```

Arguments

conn	A DBIConnection object to a DuckDB database
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
extension	name of the extension to load, default is "spatial"
create_macros	if TRUE (default), it creates macros that allow some functions to be used within dplyr pipelines

Value

TRUE (invisibly) for successful installation

Examples

```
## Not run:
## load packages
library(duckspatial)
library(duckdb)

## connect to in memory database
conn <- duckdb::dbConnect(duckdb::duckdb())

## install the spatial extension
ddbbs_install(conn)
ddbbs_load(conn)

## disconnect from db
duckdb::dbDisconnect(conn)

## End(Not run)
```

 ddbb_locate

Locate geometries at specific M values

Description

Return the geometries at a specific M values or range of M values.

Usage

```
ddbs_locate_along(
  x,
  measure,
  offset = 0,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

```
ddbs_locate_between(
  x,
  start_measure,
  end_measure,
  offset = 0,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	Input spatial data. Can be: <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn Data is returned from this object.
measure	A numeric value specifying the M value at which to locate a point along the geometry. Used only by ddbb_locate_along.
offset	A numeric value specifying a lateral offset to apply perpendicular to the line direction at the located point(s). Default is 0 (no offset).

conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
start_measure	A numeric value specifying the lower bound of the M range.
end_measure	A numeric value specifying the upper bound of the M range.

Details

- `ddbs_locate_along()`: returns a point or multi-point, containing the point(s) at the geometry with the given measure
- `ddbs_locate_between()`: returns a geometry or geometry collection created by filtering and interpolating vertices within a range of "M" values

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data (must contain M-enabled linestring geometries)
rivers_ddbs <- ddbs_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)
```

```
## Calculate the length of the rivers
rivers_agg_ddbs <- rivers_ddbs |>
  ddbb_union_agg("RIVER_NAME") |>
  ddbb_length()

## Add M dimension to the rivers
rivers_m_ddbs <- rivers_agg_ddbs |>
  ddbb_force_3d("length", dim = "M")

## Locate rivers with M between 10000 and 20000
ddbb_locate_between(rivers_m_ddbs, 10000, 20000)

## End(Not run)
```

ddbs_make_envelope *Create a rectangular polygon from bounding coordinates*

Description

Creates a rectangular POLYGON geometry from four bounding coordinates.

Usage

```
ddbs_make_envelope(  
  xmin,  
  ymin,  
  xmax,  
  ymax,  
  crs = "EPSG:4326",  
  name = NULL,  
  conn = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

Arguments

xmin	A numeric value for the minimum X (longitude) coordinate.
ymin	A numeric value for the minimum Y (latitude) coordinate.
xmax	A numeric value for the maximum X (longitude) coordinate.
ymax	A numeric value for the maximum Y (latitude) coordinate.
crs	A character string specifying the coordinate reference system of the output geometry. Default is "EPSG:4326".

name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## without storing in duckdb
finland_bbox_ddbs <- ddbs_make_envelope(
  xmin = 19.1, ymin = 59.7,
  xmax = 31.6, ymax = 70.1
)

## End(Not run)
```

ddbs_make_line	<i>Create lines from point geometries</i>
----------------	---

Description

Aggregates point geometries into a single LINESTRING by connecting them in their original order. Optionally, lines can be created per group using the `by` argument.

Usage

```
ddbs_make_line(
  x,
  by = NULL,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>by</code>	A character vector of column names to group by before aggregating points into lines. If <code>NULL</code> (default), all points are combined into a single line.
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
<code>quiet</code>	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Details

Connects input POINT geometries into a LINESTRING in row order (row 1 → row 2 → ...). To control the connection order, sort the data beforehand with `dplyr::arrange()`.

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
points_ddbs <- ddbbs_open_dataset(
  system.file("spatial/points.gpkg", package = "duckspatial")
)

## create a single line from all points
ddbbs_make_line(points_ddbs)

## create lines grouped by a column
ddbbs_make_line(points_ddbs, by = "type")

## return as sf object
ddbbs_make_line(points_ddbs, by = "type", mode = "sf")

## screate lines grouping by 2 columns
ddbbs_make_line(points_ddbs, by = c("type", "class"))

## End(Not run)
```

`ddbbs_make_polygon` *Create a polygon from a single closed linestring*

Description

Converts a single closed linestring geometry into a polygon. The linestring must be closed (first and last points identical). Does not work with MULTILINESTRING inputs - use `ddbbs_polygonize()` or `ddbbs_build_area()` instead.

Usage

```
ddbs_make_polygon(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

See Also

[ddbs_polygonize\(\)](#), [ddbs_build_area\(\)](#)

Other polygon construction: [ddbs_build_area\(\)](#), [ddbs_polygonize\(\)](#)

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbb_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbb_write_vector(conn, argentina_ddbs, "argentina")

## extract exterior ring as linestring, then convert back to polygon
ring_ddbs <- ddbb_exterior_ring(conn = conn, "argentina")
ddbb_make_polygon(conn = conn, ring_ddbs, name = "argentina_poly")

## create polygon without using a connection
ddbb_make_polygon(ring_ddbs)

## End(Not run)
```

ddbb_make_valid

Make invalid geometries valid

Description

Attempts to correct invalid geometries so they conform to the rules of well-formed geometries (e.g., fixing self-intersections or improper rings) and returns the corrected geometries.

Usage

```
ddbb_make_valid(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr/DuckDB</code> • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbs_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
```

```

)

## store in duckdb
ddbs_write_vector(conn, countries_ddbs, "countries")

## make valid
ddbs_make_valid("countries", conn)

## make valid without using a connection
ddbs_make_valid(countries_ddbs)

## End(Not run)

```

```
ddbs_maximum_inscribed_circle
```

Computes the maximum inscribed circle of a geometry

Description

Returns the largest circle that fits inside the input geometry. The result is derived from a struct containing the circle's center point, the nearest point on the geometry boundary to that center, and the circle's radius.

Usage

```

ddbs_maximum_inscribed_circle(
  x,
  geom = "center",
  tolerance = NULL,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
geom	<p>a character string specifying which component of the inscribed circle to return. Must be one of "center" (default) or "nearest". "center" returns the center point of the maximum inscribed circle; "nearest" returns the closest point on the geometry boundary to that center.</p>

tolerance	a numeric value specifying the tolerance used when computing the inscribed circle. If NULL (default), tolerance is computed automatically as $\max(\text{width}, \text{height}) / 1000$.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- duckspatial (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- sf: An eagerly collected object in R memory, that will return the same data type as the sf equivalent (e.g. sf or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
argentina_ddbs <- ddbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## return the center point of the maximum inscribed circle
ddbs_maximum_inscribed_circle(argentina_ddbs)

## return the nearest boundary point instead
ddbs_maximum_inscribed_circle(argentina_ddbs, geom = "nearest")

## use a custom tolerance
```

```
ddbs_maximum_inscribed_circle(argentina_ddbs, tolerance = 0.01)

## without a connection
ddbs_maximum_inscribed_circle(argentina_ddbs)

## End(Not run)
```

ddbs_measure_funs *Calculate geometric measurements*

Description

Compute area, length, perimeter, or distance of geometries with automatic method selection based on the coordinate reference system (CRS).

Usage

```
ddbs_area(
  x,
  new_column = "area",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

ddbs_length(
  x,
  new_column = "length",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

ddbs_perimeter(
  x,
  new_column = "perimeter",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

```

ddbbs_distance(
  x,
  y,
  dist_type = NULL,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  id_x = NULL,
  id_y = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	Input geometry (sf object, duckspatial_df, or table name in DuckDB)
new_column	Name of the new column to create on the input data. Ignored with mode = "sf".
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
y	Second input geometry for distance calculations (sf object, duckspatial_df, or table name)
dist_type	Character. Distance type to be calculated. By default it uses the best option for the input CRS (see details).
conn_x	A DBIConnection object to a DuckDB database for the input x. If NULL (default), it is resolved from conn or extracted from x.
conn_y	A DBIConnection object to a DuckDB database for the input y. If NULL (default), it is resolved from conn or extracted from y.
id_x	Character; optional name of the column in x whose values will be used to name the list elements. If NULL, integer row numbers of x are used.
id_y	Character; optional name of the column in y whose values will replace the integer indices returned in each element of the list.

Details

These functions automatically select the appropriate calculation method based on the input CRS:

For EPSG:4326 (geographic coordinates):

- Uses ST_*_Spheroid functions (e.g., ST_Area_Spheroid, ST_Length_Spheroid)
- Leverages GeographicLib library for ellipsoidal earth model calculations
- Highly accurate but slower than planar calculations
- For ddbbs_distance with POINT geometries: defaults to "haversine"
- For ddbbs_distance with other geometries: defaults to "spheroid"

For projected CRS (e.g., UTM, Web Mercator):

- Uses planar ST_* functions (e.g., ST_Area, ST_Length)
- Faster performance with accurate results in meters
- For ddbbs_distance: defaults to "planar"

Distance calculation methods (dist_type argument):

- NULL (default): Automatically selects best method for input CRS
- "planar": Planar distance (for projected CRS)
- "geos": Planar distance using GEOS library (for projected CRS)
- "haversine": Great circle distance (requires EPSG:4326 and POINT geometries)
- "spheroid": Ellipsoidal model using GeographicLib (most accurate, slowest)

Distance type requirements:

- "planar" and "geos": Require projected coordinates (not degrees)
- "haversine" and "spheroid": Require POINT geometries and EPSG:4326

Value

For ddbbs_area, ddbbs_length, and ddbbs_perimeter:

- mode = "duckspatial" (default): A duckspatial_df (lazy spatial data frame) backed by dbplyr/DuckDB.
- mode = "sf": An eagerly collected vector in R memory.
- When name is provided: writes the table in the DuckDB connection and returns TRUE (invisibly).

For ddbbs_distance: A units matrix in meters with dimensions nrow(x), nrow(y).

Performance

Speed comparison (fastest to slowest):

1. Planar calculations on projected CRS
2. Haversine (spherical approximation)
3. Spheroid functions (ellipsoidal model)

References

<https://geographiclib.sourceforge.io/>

Examples

```
## Not run:
library(duckspatial)
library(dplyr)

# Create a DuckDB connection
conn <- ddbb_create_conn(dbdir = "memory")

# ===== AREA CALCULATIONS =====

# Load polygon data
countries_ddbs <- ddbb_open_dataset(
  system.file("spatial/countries.geojson", package = "duckspatial")
) |>
  ddbb_transform("EPSG:3857") |>
  filter(NAME_ENGL != "Antarctica")

# Store in DuckDB
ddbb_write_table(conn, countries_ddbs, "countries")

# Calculate area (adds a new column - area by default)
ddbb_area("countries", conn)

# Calculate area with custom column name
ddbb_area("countries", conn, new_column = "area_sqm")

# Create new table with area calculations
ddbb_area("countries", conn, name = "countries_with_area", new_column = "area_sqm")

# Calculate area from sf object directly
ddbb_area(countries_ddbs)

# Calculate area using dplyr syntax
countries_ddbs |>
  mutate(area = ddbb_area(geom))

# Calculate total area
countries_ddbs |>
  mutate(area = ddbb_area(geom)) |>
  summarise(
    area = sum(area),
    geom = ddbb_union(geom)
  )

# ===== LENGTH CALCULATIONS =====

# Load line data
rivers_ddbs <- sf::read_sf(
```

```

  system.file("spatial/rivers.geojson", package = "duckspatial")
) |>
  as_duckspatial_df()

# Store in DuckDB
dubs_write_table(conn, rivers_dubs, "rivers")

# Calculate length (add a new column - length by default)
dubs_length("rivers", conn)

# Calculate length with custom column name
dubs_length(rivers_dubs, new_column = "length_meters")

# Calculate length by river name
rivers_dubs |>
  dubs_union_agg("RIVER_NAME") |>
  dubs_length()

# Add length within dplyr
rivers_dubs |>
  mutate(length = dubs_length(geometry))

# ===== PERIMETER CALCULATIONS =====

# Calculate perimeter (returns sf object with perimeter column)
dubs_perimeter(countries_dubs)

# Calculate perimeter within dplyr
countries_dubs |>
  mutate(perim = dubs_perimeter(geom))

# ===== DISTANCE CALCULATIONS =====

# Create sample points in EPSG:4326
n <- 10
points_sf <- data.frame(
  id = 1:n,
  x = runif(n, min = -180, max = 180),
  y = runif(n, min = -90, max = 90)
) |>
  dubs_as_spatial(coords = c("x", "y"), crs = "EPSG:4326")

# Option 1: Using sf objects (auto-selects haversine for EPSG:4326 points)
dist_matrix <- dubs_distance(x = points_sf, y = points_sf)
head(dist_matrix)

# Option 2: Explicitly specify distance type
dist_matrix_harv <- dubs_distance(
  x = points_sf,
  y = points_sf,
  dist_type = "haversine"

```

```

)

# Option 3: Using DuckDB tables
ddbs_write_table(conn, points_sf, "points", overwrite = TRUE)
dist_matrix_sph <- ddbs_distance(
  conn = conn,
  x = "points",
  y = "points",
  dist_type = "spheroid" # Most accurate for geographic coordinates
)
head(dist_matrix_sph)

# Close connection
ddbs_stop_conn(conn)

## End(Not run)

```

```
ddbs_minimum_rotated_rectangle
```

Computes the minimum rotated rectangle enclosing a geometry

Description

Returns the smallest rectangle that fully contains the input geometry. Unlike `ddbs_envelope()`, the rectangle is not constrained to be axis-aligned and may be rotated to minimize its area.

Usage

```

ddbs_minimum_rotated_rectangle(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.

name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
argentina_ddbs <- ddbbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## without a connection
ddbbs_minimum_rotated_rectangle(argentina_ddbs)

## End(Not run)
```

ddbs_multi	<i>Convert geometries to multi-type</i>
------------	---

Description

Converts single geometries to their multi-type equivalent (e.g., POLYGON to MULTIPOLYGON). Geometries that are already multi-type are returned unchanged.

Usage

```
ddbs_multi(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A duckspatial_df object (lazy spatial data frame via dbplyr) • An sf object • A tbl_lazy from dbplyr • A character string naming a table/view in conn <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
)

## convert to multi-type
ddbs_multi(countries_ddbs)

## End(Not run)
```

<code>ddbs_open_dataset</code>	<i>Open spatial dataset lazily via DuckDB</i>
--------------------------------	---

Description

Reads spatial data directly from disk using DuckDB's spatial extension or native Parquet reader, returning a `duckspatial_df` object for lazy processing.

Usage

```
ddbs_open_dataset(
  path,
  crs = NULL,
  layer = NULL,
  geom_col = NULL,
  conn = NULL,
  parquet_binary_as_string = NULL,
  parquet_file_row_number = NULL,
  parquet_filename = NULL,
  parquet_hive_partitioning = NULL,
  parquet_union_by_name = NULL,
```

```

parquet_encryption_config = NULL,
read_shp_mode = c("ST_ReadSHP", "GDAL"),
read_osm_mode = c("GDAL", "ST_ReadOSM"),
shp_encoding = NULL,
gdal_spatial_filter = NULL,
gdal_spatial_filter_box = NULL,
gdal_keep_wkb = NULL,
gdal_max_batch_size = NULL,
gdal_sequential_layer_scan = NULL,
gdal_sibling_files = NULL,
gdal_allowed_drivers = NULL,
gdal_open_options = NULL
)

```

Arguments

path	Path to spatial file. Supports Parquet (.parquet, with optional GeoParquet metadata), GeoJSON, GeoPackage, Shapefile, FlatGeoBuf, OSM PBF, DuckDB database files (.duckdb, .db, and .ddb), and other GDAL-supported formats.
crs	Coordinate reference system. Can be an EPSG code (e.g., 4326), a CRS string, or an sf crs object. If NULL (default), attempts to auto-detect from the file, including native DuckDB CRS metadata and duckspatial-managed column-comment metadata for compatibility .duckdb files.
layer	Layer name or index to read (ST_Read). For DuckDB database files, this is required and specifies the table name to read. Default is NULL (first layer for ST_Read).
geom_col	Name of the geometry column. Default is NULL, which attempts auto-detection.
conn	DuckDB connection to use. If NULL, uses the default connection.
parquet_binary_as_string	Logical. (Parquet) If TRUE, load binary columns as strings.
parquet_file_row_number	Logical. (Parquet) If TRUE, include a file_row_number column.
parquet_filename	Logical. (Parquet) If TRUE, include a filename column.
parquet_hive_partitioning	Logical. (Parquet) If TRUE, interpret path as Hive partitioned.
parquet_union_by_name	Logical. (Parquet) If TRUE, unify columns by name.
parquet_encryption_config	List/Struct. (Parquet) Encryption configuration (advanced).
read_shp_mode	Mode for reading Shapefiles. "ST_ReadSHP" (default, fast native reader) or "GDAL" (ST_Read).
read_osm_mode	Mode for reading OSM PBF files. "GDAL" (default, ST_Read) or "ST_ReadOSM" (fast native reader, no geometry).
shp_encoding	Encoding for Shapefiles when using "ST_ReadSHP" (e.g., "UTF-8", "ISO-8859-1").

gdal_spatial_filter	Optional WKB geometry (as raw vector or hex string) to filter spatially (ST_Read only).
gdal_spatial_filter_box	Optional bounding box (as numeric vector c(minx, miny, maxx, maxy)) (ST_Read only).
gdal_keep_wkb	Logical. If TRUE, return WKB blobs instead of GEOMETRY type (ST_Read only).
gdal_max_batch_size	Integer. Maximum batch size for reading (ST_Read only).
gdal_sequential_layer_scan	Logical. If TRUE, scan layers sequentially (ST_Read only).
gdal_sibling_files	Character vector. List of sibling files (ST_Read only).
gdal_allowed_drivers	Character vector. List of allowed GDAL drivers (ST_Read only).
gdal_open_options	Character vector. Driver-specific open options (ST_Read only).

Value

A duckspatial_df object.

References

This function is inspired by the dataset opening logic in the duckdbfs package (<https://github.com/cboettig/duckdbfs>).

ddb_options	<i>Get or set global duckspatial options</i>
-------------	--

Description

Get or set global duckspatial options

Usage

```
ddb_options(output_type = NULL, mode = NULL, duckdb_storage_version = NULL)
```

Arguments

output_type	Character string. Controls the default return type for <code>ddb_collect</code> . Must be one of: <ul style="list-style-type: none"> "sf" (default): Eagerly collected sf object (in-memory). "tibble": Eagerly collected tibble without geometry. "raw": Eagerly collected tibble with geometry as raw WKB bytes.
-------------	---

- "geoarrow": Eagerly collected tibble with geometry as geoarrow_vctr. If NULL (the default), the existing option is not changed.
- mode Character. Controls the return type. Options:
- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
 - "sf": Eagerly collected sf object (uses memory)
- If NULL (the default), the existing option is not changed.
- duckdb_storage_version Character. The default DuckDB storage compatibility version for newly created database files. See <https://duckdb.org/docs/internals/storage> for details.

Value

Invisibly returns a list containing the currently set options.

Examples

```
## Not run:
# Set default mode to geoarrow
ddbs_options(mode = "geoarrow")

# Set default output to tibble
ddbs_options(output_type = "tibble")

# Check current settings
ddbs_options()

## End(Not run)
```

ddbs_polygonize	<i>Assemble polygons from multiple linestrings</i>
-----------------	--

Description

Takes a collection of linestrings or polygons and assembles them into polygons by finding all closed rings formed by the network. Returns a GEOMETRYCOLLECTION containing the resulting polygons.

Usage

```
ddbs_polygonize(
  x,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>dbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `dbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

See Also

[dbs_make_polygon\(\)](#), [dbs_build_area\(\)](#)

Other polygon construction: [dbs_build_area\(\)](#), [dbs_make_polygon\(\)](#)

ddbs_predicate	<i>Evaluate spatial predicates between geometries</i>
----------------	---

Description

Determines which geometries in one dataset satisfy a specified spatial relationship with geometries in another dataset, such as intersection, containment, or touching.

Usage

```
ddbs_predicate(  
  x,  
  y,  
  predicate = "intersects",  
  conn = NULL,  
  conn_x = NULL,  
  conn_y = NULL,  
  name = NULL,  
  id_x = NULL,  
  id_y = NULL,  
  sparse = TRUE,  
  distance = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = TRUE  
)  
  
ddbs_intersects(x, y, ...)  
  
ddbs_covers(x, y, ...)  
  
ddbs_touches(x, y, ...)  
  
ddbs_is_within_distance(x, y, distance = NULL, ...)  
  
ddbs_disjoint(x, y, ...)  
  
ddbs_within(x, y, ...)  
  
ddbs_contains(x, y, ...)  
  
ddbs_overlaps(x, y, ...)  
  
ddbs_crosses(x, y, ...)  
  
ddbs_equals(x, y, ...)
```

`dbs_covered_by(x, y, ...)`

`dbs_intersects_extent(x, y, ...)`

`dbs_contains_properly(x, y, ...)`

`dbs_within_properly(x, y, ...)`

Arguments

<code>x</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
<code>y</code>	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code>
<code>predicate</code>	<p>A geometry predicate function. Defaults to <code>intersects</code>, a wrapper of <code>ST_Intersects</code>. See details for other options.</p>
<code>conn</code>	<p>A connection object to a DuckDB database. If <code>NULL</code>, the function runs on a temporary DuckDB database.</p>
<code>conn_x</code>	<p>A <code>DBIConnection</code> object to a DuckDB database for the input <code>x</code>. If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>x</code>.</p>
<code>conn_y</code>	<p>A <code>DBIConnection</code> object to a DuckDB database for the input <code>y</code>. If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>y</code>.</p>
<code>name</code>	<p>A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object</p>
<code>id_x</code>	<p>Character; optional name of the column in <code>x</code> whose values will be used to name the list elements. If <code>NULL</code>, integer row numbers of <code>x</code> are used.</p>
<code>id_y</code>	<p>Character; optional name of the column in <code>y</code> whose values will replace the integer indices returned in each element of the list.</p>
<code>sparse</code>	<p>A logical value. If <code>TRUE</code>, it returns a sparse table/list. If <code>FALSE</code>, it returns a wide table/matrix.</p>
<code>distance</code>	<p>a numeric value specifying the distance for <code>ST_DWithin</code>. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)</p>
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory)

	Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
<code>quiet</code>	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
<code>...</code>	Passed to <code>ddbs_predicate</code>

Details

This function provides a unified interface to all spatial predicate operations in DuckDB's spatial extension. It performs pairwise comparisons between all geometries in `x` and `y` using the specified predicate.

Available Predicates:

- **intersects**: Geometries share at least one point
- **covers**: Geometry `x` completely covers geometry `y`
- **touches**: Geometries share a boundary but interiors do not intersect
- **disjoint**: Geometries have no points in common
- **within**: Geometry `x` is completely inside geometry `y`
- **dwithin**: Geometry `x` is completely within a distance of geometry `y`
- **contains**: Geometry `x` completely contains geometry `y`
- **overlaps**: Geometries share some but not all points
- **crosses**: Geometries have some interior points in common
- **equals**: Geometries are spatially equal
- **covered_by**: Geometry `x` is completely covered by geometry `y`
- **intersects_extent**: Bounding boxes of geometries intersect (faster but less precise)
- **contains_properly**: Geometry `x` contains geometry `y` without boundary contact
- **within_properly**: Geometry `x` is within geometry `y` without boundary contact

If `x` or `y` are not DuckDB tables, they are automatically copied into a temporary in-memory DuckDB database (unless a connection is supplied via `conn`).

`id_x` or `id_y` may be used to replace the default integer indices with the values of an identifier column in `x` or `y`, respectively.

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `tbl_duckdb_connection` (lazy data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected list.

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```

## Not run:
## Load packages
library(duckspatial)
library(dplyr)

## create in-memory DuckDB database
conn <- ddb_create_conn(dbdir = "memory")

## read countries data, and rivers
countries_ddbs <- ddb_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
  filter(CNTR_ID %in% c("PT", "ES", "FR", "IT"))

rivers_ddbs <- ddb_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
) |>
  ddb_transform(ddb_crs(countries_ddbs))

## Store in DuckDB
ddb_write_vector(conn, countries_ddbs, "countries")
ddb_write_vector(conn, rivers_ddbs, "rivers")

## Example 1: Check which rivers intersect each country
ddb_predicate(countries_ddbs, rivers_ddbs, predicate = "intersects")
ddb_intersects(countries_ddbs, rivers_ddbs)

## Example 2: Find neighboring countries
ddb_predicate(
  countries_ddbs,
  countries_ddbs,
  predicate = "touches",
  id_x = "NAME_ENGL",
  id_y = "NAME_ENGL"
)

ddb_touches(
  countries_ddbs,
  countries_ddbs,
  id_x = "NAME_ENGL",
  id_y = "NAME_ENGL"
)

## Example 3: Find rivers that don't intersect countries
ddb_predicate(
  countries_ddbs,
  rivers_ddbs,
  predicate = "disjoint",
  id_x = "NAME_ENGL",

```

```

    id_y = "RIVER_NAME"
  )

## Example 4: Use table names inside duckdb
ddbs_predicate("countries", "rivers", predicate = "within", conn, id_x = "NAME_ENGL")
ddbs_within("countries", "rivers", conn, id_x = "NAME_ENGL")

## End(Not run)

```

ddbs_quadkey

Convert point geometries to QuadKey tiles

Description

Transforms point geometries into QuadKey identifiers at a specified zoom level, a hierarchical spatial indexing system used by mapping services.

Usage

```

ddbs_quadkey(
  x,
  level = 10,
  field = NULL,
  fun = "mean",
  background = NA,
  conn = NULL,
  name = NULL,
  output = "polygon",
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	Input spatial data. Can be: <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> Data is returned from this object.
level	An integer specifying the zoom level for QuadKey generation (1-23). Higher values provide finer spatial resolution. Default is 10.
field	Character string specifying the field name for aggregation.
fun	aggregation function for when there are multiple quadkeys (e.g. "mean", "min", "max", "sum").

background	numeric. Default value in raster cells without values. Only used when output = "raster"
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
output	Character string specifying output format. One of: <ul style="list-style-type: none"> • "polygon" - Returns QuadKey tile boundaries as duckspatial_df (default) • "raster" - Returns QuadKey values as a SpatRaster • "tilexy" - Returns tile XY coordinates as a tibble
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

QuadKeys divide the world into a hierarchical grid of tiles, where each tile is subdivided into four smaller tiles at the next zoom level. This function wraps DuckDB's ST_QuadKey spatial function to generate these tiles from input geometries.

Note that creating a table inside the connection will generate a non-spatial table, and therefore, it cannot be read with [ddbbs_read_table](#).

Value

Depends on the output argument

- polygon (default): A lazy spatial data frame backed by dbplyr/DuckDB.
- raster: An eagerly collected SpatRaster object in R memory.
- tilexy: An eagerly collected tibble without geometry in R memory.

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)
library(sf)
library(terra)

# create a duckdb database in memory (with spatial extension)
conn <- ddbbs_create_conn(dbdir = "memory")

## create random points in Argentina
```

```

argentina_sf <- st_read(system.file("spatial/argentina.geojson", package = "duckspatial"))
rand_sf <- st_sample(argentina_sf, 100) |> st_as_sf()
rand_sf["var"] <- runif(100)

## store in duckdb
ddbs_write_vector(conn, rand_sf, "rand_sf")

## generate QuadKey polygons at zoom level 8
qkey_ddbs <- ddbs_quadkey(conn = conn, "rand_sf", level = 8, output = "polygon")

## generate QuadKey raster with custom field name
qkey_rast <- ddbs_quadkey(conn = conn, "rand_sf", level = 6, output = "raster", field = "var")

## generate Quadkey XY tiles
qkey_tiles_tbl <- ddbs_quadkey(conn = conn, "rand_sf", level = 10, output = "tilexy")

## End(Not run)

```

ddbs_read_meta	<i>Read metadata from a spatial file</i>
----------------	--

Description

Retrieves file-level metadata from a spatial vector file (e.g. GeoPackage, Shapefile, GeoJSON) using DuckDB's `ST_Read_Meta()` function. Returns information about the file's driver and its layers as a tibble.

Usage

```
ddbs_read_meta(path, conn = NULL)
```

Arguments

<code>path</code>	character, path to the spatial file to inspect.
<code>conn</code>	A <code>DBIConnection</code> object to a DuckDB database

Value

A tibble with one row per file and the following columns:

file_name Path to the file.

driver_short_name Short name of the GDAL driver (e.g. "GPKG").

driver_long_name Full name of the GDAL driver (e.g. "GeoPackage").

layers A list-column of data frames, one per file, each describing the layers contained in the file. Unnest with `unnest` to access individual layer attributes such as name, geometry type, and feature count.

Examples

```
## Not run:
## Read metadata from a GeoPackage
meta <- ddbbs_read_meta(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## View file-level metadata
meta

## Inspect layer details
tidyr::unnest(meta, layers)

## End(Not run)
```

ddbbs_read_table	<i>Reads a vectorial table from DuckDB into R</i>
------------------	---

Description

Retrieves the data from a DuckDB table, view, or Arrow view with a geometry column, and converts it to an R sf object. This function works with both persistent tables created by `ddbbs_write_table` and temporary Arrow views created by `ddbbs_register_table`.

Usage

```
ddbbs_read_table(conn, name, clauses = NULL, quiet = FALSE)
```

Arguments

conn	A DBIConnection object to a DuckDB database
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
clauses	character, additional SQL code to modify the query from the table (e.g. "WHERE ...", "ORDER BY...")
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

an sf object

Examples

```
## Not run:
## load packages
library(duckspatial)
library(sf)

# create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

## create random points
random_points <- data.frame(
  id = 1:5,
  x = runif(5, min = -180, max = 180),
  y = runif(5, min = -90, max = 90)
)

## convert to sf
sf_points <- st_as_sf(random_points, coords = c("x", "y"), crs = 4326)

## Example 1: Write and read persistent table
ddb_write_vector(conn, sf_points, "points")
ddb_read_table(conn, "points")

## Example 2: Register and read Arrow view (faster, temporary)
ddb_register_vector(conn, sf_points, "points_view")
ddb_read_table(conn, "points_view")

## disconnect from db
ddb_stop_conn(conn)

## End(Not run)
```

ddbs_register_table *Register an SF Object as an Arrow Table in DuckDB*

Description

This function registers a Simple Features (SF) object as a temporary Arrow-backed view in a DuckDB database. This is a zero-copy operation and is significantly faster than `ddb_write_table` for workflows that do not require data to be permanently materialized in the database.

Usage

```
ddb_register_table(conn, data, name, overwrite = FALSE, quiet = FALSE)
```

Arguments

conn A DBIConnection object to a DuckDB database

data	A sf object to write to the DuckDB database, or the path to a local file that can be read with ST_READ
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

TRUE (invisibly) on successful registration.

Examples

```
## Not run:
library(duckdb)
library(duckspatial)
library(sf)

conn <- ddbs_create_conn("memory")

nc <- st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)

ddbs_register_table(conn, nc, "nc_arrow_view")

dbGetQuery(conn, "SELECT COUNT(*) FROM nc_arrow_view;")

ddbs_stop_conn(conn)

## End(Not run)
```

```
ddbs_remove_repeated_points
```

Remove repeated points from a geometry

Description

Removes duplicate consecutive vertices from geometries, optionally within a tolerance distance.

Usage

```
ddbs_remove_repeated_points(
  x,
  tolerance = 0,
  conn = NULL,
  name = NULL,
```

```

mode = NULL,
overwrite = FALSE,
quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
tolerance	<p>A numeric value specifying the minimum distance between consecutive vertices. Vertices closer than this threshold are considered repeated and removed. Default is 0, which removes only exact duplicates.</p>
conn	<p>A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.</p>
name	<p>A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an <code>sf</code> object</p>
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	<p>Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when <code>name</code> is NULL.</p>
quiet	<p>A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.</p>

Value

Depends on the `mode` argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
library(sf)

## Create a polygon with repeated points
poly <- st_polygon(list(matrix(
  c(0, 0,
    1, 0,
    1, 0, # repeated point
    1, 1,
    0, 1,
    0, 0),
  ncol = 2, byrow = TRUE
)))

poly_sf <- st_as_sf(st_sfc(poly))

## remove exact duplicate consecutive vertices
ddbs_remove_repeated_points(poly_sf)

## remove vertices within a tolerance of 1 unit
ddbs_remove_repeated_points(poly_sf, tolerance = 1)

## End(Not run)
```

ddbs_rotate

Rotate geometries around their centroid

Description

Rotates geometries by a specified angle around their centroid (or another center), preserving their shape.

Usage

```
ddbs_rotate(
  x,
  angle,
  units = c("degrees", "radians"),
  by_feature = FALSE,
  center_x = NULL,
  center_y = NULL,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
angle	a numeric value specifying the rotation angle
units	character string specifying angle units: "degrees" (default) or "radians"
by_feature	Logical. If <code>TRUE</code> , the geometric operation is applied separately to each geometry. If <code>FALSE</code> , the geometric operation is applied to the data as a whole.
center_x	numeric value for the X coordinate of rotation center. If <code>NULL</code> , rotates around the centroid of each geometry
center_y	numeric value for the Y coordinate of rotation center. If <code>NULL</code> , rotates around the centroid of each geometry
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```

## Not run:
## load packages
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbbs_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbbs_write_table(conn, argentina_ddbs, "argentina")

## rotate 45 degrees
ddbbs_rotate(conn = conn, "argentina", angle = 45)

## rotate 90 degrees around a specific point
ddbbs_rotate(conn = conn, "argentina", angle = 90, center_x = -64, center_y = -34)

## rotate without using a connection
ddbbs_rotate(argentina_ddbs, angle = 45)

## End(Not run)

```

ddbbs_rotate_3d

Rotate 3D geometries around an axis

Description

Rotates 3D geometries by a specified angle around the X, Y, or Z axis, preserving their shape.

Usage

```

ddbbs_rotate_3d(
  x,
  angle,
  units = c("degrees", "radians"),
  axis = "x",
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
angle	a numeric value specifying the rotation angle
units	character string specifying angle units: "degrees" (default) or "radians"
axis	character string specifying the rotation axis: "x", "y", or "z" (default = "x"). The geometry rotates around this axis
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)
library(dplyr)

# create a duckdb database in memory (with spatial extension)
```

```

conn <- ddbbs_create_conn(dbdir = "memory")

## read 3D data
countries_ddbs <- ddbbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
  filter(CNTR_ID %in% c("PT", "ES", "FR", "IT"))

## store in duckdb
ddbbs_write_table(conn, countries_ddbs, "countries")

## rotate 45 degrees around X axis (pitch)
ddbbs_rotate_3d(conn = conn, "countries", angle = 45, axis = "x")

## rotate 90 degrees around Y axis (yaw)
ddbbs_rotate_3d(conn = conn, "countries", angle = 30, axis = "y")

## rotate 180 degrees around Z axis (roll)
ddbbs_rotate_3d(conn = conn, "countries", angle = 180, axis = "z")

## rotate without using a connection
ddbbs_rotate_3d(countries_ddbs, angle = 45, axis = "z")

## End(Not run)

```

ddbbs_scale

Scale geometries by X and Y factors

Description

Resizes geometries by specified X and Y scale factors. By default, scaling is performed relative to the centroid of all geometries; if `by_feature = TRUE`, each geometry is scaled relative to its own centroid.

Usage

```

ddbbs_scale(
  x,
  x_scale = 1,
  y_scale = 1,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
x_scale	numeric value specifying the scaling factor in the X direction (default = 1)
y_scale	numeric value specifying the scaling factor in the Y direction (default = 1)
by_feature	Logical. If TRUE, the geometric operation is applied separately to each geometry. If FALSE, the geometric operation is applied to the data as a whole.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when <code>name</code> is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)
library(dplyr)

# create a duckdb database in memory (with spatial extension)
```

```
conn <- ddbbs_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddbbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
  filter(CNTR_ID %in% c("PT", "ES", "FR", "IT"))

## store in duckdb
ddbbs_write_table(conn, countries_ddbs, "countries")

## scale to 150% in both directions
ddbbs_scale(conn = conn, "countries", x_scale = 1.5, y_scale = 1.5)

## scale to 200% horizontally, 50% vertically
ddbbs_scale(conn = conn, "countries", x_scale = 2, y_scale = 0.5)

## scale all features together (default)
ddbbs_scale(countries_ddbs, x_scale = 1.5, y_scale = 1.5, by_feature = FALSE)

## scale each feature independently
ddbbs_scale(countries_ddbs, x_scale = 1.5, y_scale = 1.5, by_feature = TRUE)

## End(Not run)
```

ddbbs_set_crs

Set the coordinate reference system of geometries

Description

Assigns or replaces the coordinate reference system (CRS) of geometries without transforming their coordinates. This is useful when the CRS is missing or incorrectly defined.

Usage

```
ddbbs_set_crs(
  x,
  y,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
y	<p>Target CRS. Can be:</p> <ul style="list-style-type: none"> • A character string with EPSG code (e.g., "EPSG:4326") • A <code>crs</code> object created with <code>sf::st_crs</code>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by <code>dbplyr</code>/DuckDB • "sf": Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr`/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)
library(sf)

## read data
rivers_ddbs <- ddbs_open_dataset(
```

```

    system.file("spatial/rivers.geojson",
    package = "duckspatial")
)

## Remove CRS
rivers_no_crs_ddbs <- ddbbs_set_crs(rivers_ddbs, st_crs(NA))

## Set the CRS back
ddbbs_set_crs(rivers_no_crs_ddbs, "EPSG:3035")

## End(Not run)

```

ddbbs_set_resources *Get or set connection resources*

Description

Configure technical system settings for a DuckDB connection, such as memory limits and CPU threads.

Usage

```

ddbbs_set_resources(conn, threads = NULL, memory_limit_gb = NULL)

ddbbs_get_resources(conn)

```

Arguments

conn	A DBIConnection object to a DuckDB database
threads	Integer. Number of threads to use. If NULL (default), the setting is not changed, and DuckDB engine will use all available cores it detects (warning, on some shared HPC nodes the detected number of cores might be total number of cores on the node, not the per-job allocation).
memory_limit_gb	Numeric. Memory limit in GB. If NULL (default), the setting is not changed, and DuckDB engine will use 80% of available operating system memory it detects (warning, on some shared HPC nodes the detected memory might be the full node memory, not the per-job allocation).

Value

For `ddbbs_set_resources()`, invisibly returns a list containing the current system settings; for `ddbbs_get_resources()`, visibly returns the same list for direct inspection.

Examples

```
## Not run:
# Create a connection
conn <- ddbb_create_conn()

# Set resources: 1 thread and 4GB
ddbb_set_resources(conn, threads = 1, memory_limit_gb = 4)

# Check current settings
ddbb_get_resources(conn)

ddbb_stop_conn(conn)

## End(Not run)
```

ddbb_shear

Shear geometries

Description

Applies a shear transformation to geometries, shifting coordinates proportionally in the X and Y directions. By default, shearing is applied relative to the centroid of all geometries; if `by_feature = TRUE`, each geometry is sheared relative to its own centroid.

Usage

```
ddbb_shear(
  x,
  x_shear = 0,
  y_shear = 0,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

- x Input spatial data. Can be:
- A `duckspatial_df` object (lazy spatial data frame via `dbplyr`)
 - An `sf` object
 - A `tbl_lazy` from `dbplyr`
 - A character string naming a table/view in `conn`
- Data is returned from this object.

x_shear	numeric value specifying the shear factor in the X direction (default = 0). For each unit in Y, X coordinates are shifted by this amount
y_shear	numeric value specifying the shear factor in the Y direction (default = 0). For each unit in X, Y coordinates are shifted by this amount
by_feature	Logical. If TRUE, the geometric operation is applied separately to each geometry. If FALSE, the geometric operation is applied to the data as a whole.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)
library(dplyr)

# create a duckdb database in memory (with spatial extension)
conn <- ddbbs_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddbbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
```

```

    filter(CNTR_ID %in% c("PT", "ES", "FR", "IT"))

## store in duckdb
ddbs_write_table(conn, countries_ddbs, "countries")

## shear in X direction (creates italic-like effect)
ddbs_shear(conn = conn, "countries", x_shear = 0.3, y_shear = 0)

## shear in Y direction
ddbs_shear(conn = conn, "countries", x_shear = 0, y_shear = 0.3)

## shear in both directions
ddbs_shear(conn = conn, "countries", x_shear = 0.2, y_shear = 0.2)

## shear without using a connection
ddbs_shear(countries_ddbs, x_shear = 0.3, y_shear = 0)

## End(Not run)

```

ddbs_shift

Shift geometries by X and Y offsets

Description

Translates geometries by specified X and Y distances, moving them without altering their shape or orientation.

Usage

```

ddbs_shift(
  x,
  dx = 0,
  dy = 0,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x Input spatial data. Can be:

- A `duckspatial_df` object (lazy spatial data frame via `dbplyr`)
- An `sf` object
- A `tbl_lazy` from `dbplyr`
- A character string naming a table/view in `conn`

Data is returned from this object.

dx	numeric value specifying the shift in the X direction (longitude/easting)
dy	numeric value specifying the shift in the Y direction (latitude/northing)
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load packages
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- ddbs_create_conn(dbdir = "memory")

## read data
argentina_ddbs <- ddbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)

## store in duckdb
ddbs_write_table(conn, argentina_ddbs, "argentina")

## shift 10 degrees east and 5 degrees north
ddbs_shift(conn = conn, "argentina", dx = 10, dy = 5)
```

```
## shift without using a connection
ddbs_shift(argentina_ddbs, dx = 10, dy = 5)

## End(Not run)
```

ddbs_simplify

Simplify geometries

Description

Reduces the complexity of geometries by removing unnecessary vertices while preserving the overall shape.

Usage

```
ddbs_simplify(
  x,
  tolerance = 0,
  preserve_topology = FALSE,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
tolerance	Tolerance distance for simplification. Larger values result in more simplified geometries.
preserve_topology	If <code>FALSE</code> , uses the Douglas-Peucker algorithm, which reduces the vertices by removing points that are within a given distance. If <code>TRUE</code> , uses a topology-preserving variant of Douglas-Peucker that guarantees the output geometry remains valid (slower).
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.

name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>dbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Value

Depends on the mode argument (or global preference set by `dbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- dbs_create_conn(dbdir = "memory")

## read data
countries_dbs <- dbs_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
)

## store in duckdb
dbs_write_vector(conn, countries_dbs, "countries")

## simplify with tolerance of 0.01
dbs_simplify("countries", tolerance = 0.01, conn = conn)

## simplify without using a connection
dbs_simplify(countries_dbs, tolerance = 0.01)

## End(Not run)
```

ddbs_sitrep	<i>Report duckspatial configuration status</i>
-------------	--

Description

Displays useful information about the current configuration, including global options and the status of the default DuckDB connection.

Usage

```
ddbs_sitrep()
```

Value

Invisibly returns a list with the current status configuration.

Examples

```
ddbs_sitrep()
```

ddbs_stop_conn	<i>Close a DuckDB connection</i>
----------------	----------------------------------

Description

Close a DuckDB connection

Usage

```
ddbs_stop_conn(conn)
```

Arguments

conn A DBIConnection object to a DuckDB database

Value

TRUE (invisibly) for successful disconnection

Examples

```
## Not run:
## load packages
library(duckspatial)

## create an in-memory duckdb database
conn <- ddbbs_create_conn(dbdir = "memory")

## close the connection
ddbbs_stop_conn(conn)

## End(Not run)
```

ddbbs_transform

Transform the coordinate reference system of geometries

Description

Converts geometries to a different coordinate reference system (CRS), updating their coordinates accordingly.

Usage

```
ddbbs_transform(
  x,
  y,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

- x Input spatial data. Can be:
- A `duckspatial_df` object (lazy spatial data frame via `dbplyr`)
 - An `sf` object
 - A `tbl_lazy` from `dbplyr`
 - A character string naming a table/view in `conn`
- Data is returned from this object.
- y Target CRS. Can be:
- A character string with EPSG code (e.g., "EPSG:4326")
 - A `crs` object created with `sf::st_crs`

	<ul style="list-style-type: none"> • An <code>sf</code> object (uses its CRS) • Name of a DuckDB table (uses its CRS)
<code>conn</code>	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
<code>conn_x</code>	A <code>DBIConnection</code> object to a DuckDB database for the input <code>x</code> . If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>x</code> .
<code>conn_y</code>	A <code>DBIConnection</code> object to a DuckDB database for the input <code>y</code> . If <code>NULL</code> (default), it is resolved from <code>conn</code> or extracted from <code>y</code> .
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
<code>mode</code>	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr/DuckDB</code> • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>dbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
<code>quiet</code>	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `dbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

# create a duckdb database in memory (with spatial extension)
conn <- dbs_create_conn(dbdir = "memory")

## read data
argentina_dbs <- dbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)
```

```

## store in duckdb
ddbs_write_vector(conn, argentina_ddbs, "argentina")

## transform to different CRS using EPSG code
ddbs_transform("argentina", "EPSG:3857", conn)

## transform to match CRS of another object
argentina_3857_ddbs <- ddbs_transform(argentina_ddbs, "EPSG:3857")
ddbs_write_vector(conn, argentina_3857_ddbs, "argentina_3857")
ddbs_transform("argentina", argentina_3857_ddbs, conn)

## transform to match CRS of another DuckDB table
ddbs_transform("argentina", "argentina_3857", conn)

## transform without using a connection
ddbs_transform(argentina_ddbs, "EPSG:3857")

## End(Not run)

```

ddbs_union_funs

Union and combine geometries

Description

Perform union and combine operations on spatial geometries in DuckDB.

- `ddbs_union()` - Union all geometries into one, or perform pairwise union between two datasets
- `ddbs_union_agg()` - Union geometries grouped by one or more columns
- `ddbs_combine()` - Combine geometries into a MULTI-geometry without dissolving boundaries

Usage

```

ddbs_union(
  x,
  y = NULL,
  by_feature = FALSE,
  conn = NULL,
  conn_x = NULL,
  conn_y = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

ddbs_combine(

```

```

    x,
    conn = NULL,
    name = NULL,
    mode = NULL,
    overwrite = FALSE,
    quiet = FALSE
)

dbs_union_agg(
  x,
  by,
  conn = NULL,
  name = NULL,
  mode = NULL,
  overwrite = FALSE,
  quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
y	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • NULL (default): performs only the union of x • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code>
by_feature	<p>Logical. When y is provided:</p> <ul style="list-style-type: none"> • FALSE (default) - Union all geometries from both x and y into a single geometry • TRUE - Perform row-by-row union between matching features from x and y (requires same number of rows)
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
conn_x	A <code>DBIConnection</code> object to a DuckDB database for the input x. If NULL (default), it is resolved from <code>conn</code> or extracted from x.
conn_y	A <code>DBIConnection</code> object to a DuckDB database for the input y. If NULL (default), it is resolved from <code>conn</code> or extracted from y.

name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
mode	Character. Controls the return type. Options: <ul style="list-style-type: none"> • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
by	Character vector specifying one or more column names to group by when computing unions. Geometries will be unioned within each group. Default is NULL

Details

ddbbs_union(x, y, by_feature):

Performs geometric union operations that dissolve internal boundaries:

- When `y = NULL`: Unions all geometries in `x` into a single geometry
- When `y != NULL` and `by_feature = FALSE`: Unions all geometries from both `x` and `y` into a single geometry
- When `y != NULL` and `by_feature = TRUE`: Performs row-wise union, pairing the first geometry from `x` with the first from `y`, second with second, etc.

ddbbs_union_agg(x, by):

Groups geometries by one or more columns, then unions geometries within each group. Useful for dissolving boundaries between features that share common attributes.

ddbbs_combine(x):

Combines all geometries into a single MULTI-geometry (e.g., MULTIPOLYGON, MULTILINESTRING) without dissolving shared boundaries. This is faster than union but preserves all original geometry boundaries.

Value

Depends on the mode argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```

## Not run:
## load packages
library(dplyr)
library(duckspatial)

## create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

## read data
countries_ddbs <- ddb_open_dataset(
  system.file("spatial/countries.geojson",
    package = "duckspatial")
) |>
  filter(IS03_CODE != "ATA")

rivers_ddbs <- ddb_open_dataset(
  system.file("spatial/rivers.geojson",
    package = "duckspatial")
) |>
  ddb_transform("EPSG:4326")

## combine countries into a single MULTI-geometry
## (without solving boundaries)
combined_countries_ddbs <- ddb_combine(countries_ddbs)

## combine countries into a single MULTI-geometry
## (solving boundaries)
union_countries_ddbs <- ddb_union(countries_ddbs)

## union of geometries of two objects, into 1 geometry
union_countries_rivers_ddbs <- ddb_union(countries_ddbs, rivers_ddbs)

## End(Not run)

```

ddbs_voronoi

Computes a Voronoi diagram from point geometries

Description

Returns a Voronoi diagram (Thiessen polygons) from a collection of points. Each polygon represents the region closer to one point than to any other point in the set. This function only works with MULTIPOINT geometries.

Usage

```

ddbs_voronoi(
  x,

```

```

conn = NULL,
name = NULL,
mode = NULL,
overwrite = FALSE,
quiet = FALSE
)

```

Arguments

x	<p>Input spatial data. Can be:</p> <ul style="list-style-type: none"> • A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>) • An <code>sf</code> object • A <code>tbl_lazy</code> from <code>dbplyr</code> • A character string naming a table/view in <code>conn</code> <p>Data is returned from this object.</p>
conn	A connection object to a DuckDB database. If <code>NULL</code> , the function runs on a temporary DuckDB database.
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If <code>NULL</code> (the default), the function returns the result as an <code>sf</code> object
mode	<p>Character. Controls the return type. Options:</p> <ul style="list-style-type: none"> • <code>"duckspatial"</code> (default): Lazy spatial data frame backed by <code>dbplyr/DuckDB</code> • <code>"sf"</code>: Eagerly collected <code>sf</code> object (uses memory) <p>Can be set globally via <code>ddbbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.</p>
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to <code>FALSE</code> . This argument is ignored when <code>name</code> is <code>NULL</code> .
quiet	A logical value. If <code>TRUE</code> , suppresses any informational messages. Defaults to <code>FALSE</code> .

Value

Depends on the `mode` argument (or global preference set by `ddbbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by `dbplyr/DuckDB`.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or units vector).

When `name` is provided, the result is also written as a table or view in DuckDB and the function returns `TRUE` (invisibly).

Examples

```

## Not run:
## load package
library(duckspatial)

```

```

## create a duckdb database in memory (with spatial extension)
conn <- ddbb_create_conn(dbdir = "memory")

## create some points, and combine them to MULTIPOINT
set.seed(42)
n <- 1000
points_ddbs <- data.frame(
  id = 1:n,
  x = runif(n, min = -20, max = 20),
  y = runif(n, min = -20, max = 02)
) |>
  ddbb_as_spatial(coords = c("x", "y"), crs = 4326) |>
  ddbb_combine()

## create voronoi diagrama
ddbb_voronoi(points_ddbs)

## End(Not run)

```

ddbs_write_dataset *Write spatial dataset to disk*

Description

Writes spatial data to disk using DuckDB's COPY command for Parquet and GDAL spatial formats, or as a native DuckDB database for .duckdb, .db, and .ddb paths. Format is auto-detected from file extension for common formats, or can be specified explicitly via `gdal_driver`.

Usage

```

ddbb_write_dataset(
  data,
  path,
  gdal_driver = NULL,
  conn = NULL,
  overwrite = FALSE,
  crs = NULL,
  layer = "spatial",
  options = list(),
  partitioning = if (inherits(data, c("tbl_lazy", "duckspatial_df")))
    dplyr::group_vars(data) else NULL,
  parquet_compression = NULL,
  parquet_row_group_size = NULL,
  layer_creation_options = NULL,
  quiet = FALSE,
  duckdb_storage_version = duckspatial_storage_default()
)

```

Arguments

data	A duckspatial_df, tbl_lazy (DuckDB), or sf object.
path	Path to output file.
gdal_driver	GDAL driver name for writing spatial formats. If NULL (default), the driver is auto-detected from the file extension for common formats: <ul style="list-style-type: none"> • .geojson, .json → "GeoJSON" • .shp → "ESRI Shapefile" • .gpkg → "GPKG" • .fgb → "FlatGeobuf" • .kml → "KML" • .gpx → "GPX" • .gml → "GML" • .sqlite → "SQLite" <p>For non-standard file extensions (e.g., .dat, .xyz) or to explicitly override auto-detection, specify the exact driver name as it appears in ddb_drivers()\$short_name. Examples: gdal_driver = "GeoJSON", gdal_driver = "ESRI Shapefile".</p> <p>Note: If you specify a driver that doesn't match the file extension (e.g., path = "output.shp" with gdal_driver = "GeoJSON"), a warning will be issued but your explicit driver choice will be honored (creating a GeoJSON file with .shp extension).</p> <p>The function validates that the specified driver is available and writable on your system. Note: .parquet and .csv files use native DuckDB writers and do not require a GDAL driver.</p>
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
overwrite	Logical. If TRUE, overwrites existing file.
crs	Output CRS (e.g., "EPSG:4326"). Passed to GDAL as SRS option. Ignored for Parquet.
layer	Table name for native DuckDB database output.
options	Named list of additional options passed to COPY.
partitioning	Character vector of columns to partition by (Parquet/CSV only).
parquet_compression	Compression codec for Parquet.
parquet_row_group_size	Row group size for Parquet.
layer_creation_options	GDAL layer creation options.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.
duckdb_storage_version	Storage compatibility for newly created native DuckDB database files (.duckdb, .db, .ddb). See https://duckdb.org/docs/internals/storage for more information on DuckDB storage versions and compatibility.

- "v1.5.0" (**Native Spatial Storage**, Default): Preserves CRS metadata in native DuckDB GEOMETRY columns. Requires DuckDB \geq 1.5.0 to open the file.
- "v1.0.0" (**Legacy Compatibility**): Creates files readable by older DuckDB versions (\geq 1.0.0). Persists CRS metadata in duckspatial-managed column comments (a convention not recognized by other spatial software).
- "latest": Use the highest storage version supported by your installed DuckDB engine.

Other major version strings like "v1.4.0", "v1.3.0", etc., are also supported.

Details

Persistent DuckDB database files created by duckspatial use **Native Spatial Storage** (`storage_version = "v1.5.0"`) by default so CRS metadata is retained in native GEOMETRY columns. These files require DuckDB \geq 1.5.0 to open; use **Legacy Compatibility** (`storage_version = "v1.0.0"`) when the output must be readable by older DuckDB versions.

Value

The path invisibly.

References

This function is inspired by and builds upon the logic found in the duckdbfs package (<https://github.com/cboettig/duckdbfs>), particularly its `write_dataset` and `write_geo` functions. For advanced features like cloud storage (S3) support, the duckdbfs package is highly recommended.

See Also

[ddbs_drivers\(\)](#) to list all available GDAL drivers and formats.

Examples

```
## Not run:
library(duckspatial)

# Read example data
path <- system.file("spatial/countries.geojson", package = "duckspatial")
ds <- ddbs_open_dataset(path)

# Auto-detect format from extension
ddbs_write_dataset(ds, "output.geojson")
ddbs_write_dataset(ds, "output.gpkg")
ddbs_write_dataset(ds, "output.parquet")

# Explicit GDAL driver for non-standard extension
ddbs_write_dataset(ds, "mydata.dat", gdal_driver = "GeoJSON")

# See available drivers on your system
drivers <- ddbs_drivers()
```

```
writable <- drivers[drivers$can_create == TRUE, ]
head(writable)

# CRS override
ddbs_write_dataset(ds, "output_3857.geojson", crs = "EPSG:3857")

# Overwrite existing file
ddbs_write_dataset(ds, "output.gpkg", overwrite = TRUE)

## End(Not run)
```

ddbs_write_table	<i>Write an SF Object to a DuckDB Database</i>
------------------	--

Description

This function writes a Simple Features (SF) object into a DuckDB database as a new table. The table is created in the specified schema of the DuckDB database.

Usage

```
ddbs_write_table(
  conn,
  data,
  name,
  overwrite = FALSE,
  temp_view = FALSE,
  quiet = FALSE
)
```

Arguments

conn	A DBIConnection object to a DuckDB database
data	A sf object to write to the DuckDB database, or the path to a local file that can be read with ST_READ
name	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
overwrite	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
temp_view	If TRUE, registers the sf object as a temporary Arrow-backed database 'view' using ddbs_register_table instead of creating a persistent table. This is much faster but the view will not persist. Defaults to FALSE.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

CRS Persistence: duckspatial ensures CRS metadata is retained across sessions using two strategies: **Native Spatial Storage** (for DuckDB 1.5.0+ databases) and **Legacy Compatibility** (using column comments for older database versions). For file-based spatial data interchange, `ddbs_write_dataset()` to GeoParquet (.parquet) is recommended.

Value

TRUE (invisibly) for successful import

Examples

```
## Not run:
## load packages
library(duckspatial)
library(sf)

# create a duckdb database in memory (with spatial extension)
conn <- ddb_create_conn(dbdir = "memory")

## create random points
random_points <- data.frame(
  id = 1:5,
  x = runif(5, min = -180, max = 180), # Random longitude values
  y = runif(5, min = -90, max = 90)    # Random latitude values
)

## convert to sf
sf_points <- st_as_sf(random_points, coords = c("x", "y"), crs = 4326)

## insert data into the database
ddbs_write_table(conn, sf_points, "points")

## read data back into R
ddbs_read_table(conn, "points")

## disconnect from db
ddbs_stop_conn(conn)

## End(Not run)
```

ddbs_xy

Extract coordinates from geometries

Description

Extracts the X, Y, M, or Z coordinates from POINT geometries

Usage

```
ddbbs_x(  
  x,  
  new_column = "X",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbbs_y(  
  x,  
  new_column = "Y",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbbs_m(  
  x,  
  new_column = "M",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

```
ddbbs_z(  
  x,  
  new_column = "Z",  
  conn = NULL,  
  name = NULL,  
  mode = NULL,  
  overwrite = FALSE,  
  quiet = FALSE  
)
```

Arguments

- | | |
|---|---|
| x | Input spatial data. Can be: <ul style="list-style-type: none">• A <code>duckspatial_df</code> object (lazy spatial data frame via <code>dbplyr</code>)• An <code>sf</code> object• A <code>tbl_lazy</code> from <code>dbplyr</code>• A character string naming a table/view in <code>conn</code> |
|---|---|

	Data is returned from this object.
<code>new_column</code>	Name of the new column to store the extracted coordinate. Defaults to "X", "Y", "M", or "Z".
<code>conn</code>	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
<code>name</code>	A character string of length one specifying the name of the table, or a character string of length two specifying the schema and table names. If NULL (the default), the function returns the result as an sf object
<code>mode</code>	Character. Controls the return type. Options: <ul style="list-style-type: none"> "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB "sf": Eagerly collected sf object (uses memory) Can be set globally via <code>ddbs_options(mode = "...")</code> or per-function via this argument. Per-function overrides global setting.
<code>overwrite</code>	Boolean. whether to overwrite the existing table if it exists. Defaults to FALSE. This argument is ignored when name is NULL.
<code>quiet</code>	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

- `ddbs_x()`: Extracts the X coordinate (longitude).
- `ddbs_y()`: Extracts the Y coordinate (latitude).
- `ddbs_m()`: Extracts the M coordinate (measure).
- `ddbs_z()`: Extracts the Z coordinate (elevation).

Value

Depends on the mode argument (or global preference set by `ddbs_options`):

- `duckspatial` (default): A `duckspatial_df` (lazy spatial data frame) backed by dbplyr/DuckDB.
- `sf`: An eagerly collected object in R memory, that will return the same data type as the `sf` equivalent (e.g. `sf` or `units` vector).

When name is provided, the result is also written as a table or view in DuckDB and the function returns TRUE (invisibly).

Examples

```
## Not run:
## load package
library(duckspatial)

## read data
argentina_ddbs <- ddbs_open_dataset(
  system.file("spatial/argentina.geojson",
    package = "duckspatial")
)
```

```
## extract coordinates
dbs_x(argentina_dbs)
dbs_y(argentina_dbs)

## End(Not run)
```

is_duckspatial_df *Check if object is a duckspatial_df*

Description

Check if object is a duckspatial_df

Usage

```
is_duckspatial_df(x)
```

Arguments

x Object to test

Value

Logical

Index

* polygon construction

- ddbs_build_area, 21
 - ddbs_make_polygon, 84
 - ddbs_polygonize, 101
- areal::aw_interpolate(), 67
- as_duckspatial_df, 3
- as_nanoarrow_array_stream.duckspatial_df, 5
- collect.duckspatial_df, 5
- dbConnect, 30
- ddbs_affine, 6
- ddbs_area (ddbs_measure_funs), 90
- ddbs_as_format, 8
- ddbs_as_geojson (ddbs_as_format), 8
- ddbs_as_hexwkb (ddbs_as_format), 8
- ddbs_as_points, 10
- ddbs_as_text (ddbs_as_format), 8
- ddbs_as_wkb (ddbs_as_format), 8
- ddbs_bbox, 12
- ddbs_binary_funs, 14
- ddbs_boundary, 17
- ddbs_buffer, 19
- ddbs_build_area, 21, 86, 102
- ddbs_build_area(), 84, 86, 102
- ddbs_centroid, 23
- ddbs_collect, 100
- ddbs_collect (collect.duckspatial_df), 5
- ddbs_combine (ddbs_union_funs), 132
- ddbs_compute, 24
- ddbs_concave_hull, 26
- ddbs_contains (ddbs_predicate), 103
- ddbs_contains_properly (ddbs_predicate), 103
- ddbs_convex_hull, 28
- ddbs_covered_by (ddbs_predicate), 103
- ddbs_covers (ddbs_predicate), 103
- ddbs_create_conn, 30
- ddbs_create_schema, 31
- ddbs_crop (ddbs_binary_funs), 14
- ddbs_crosses (ddbs_predicate), 103
- ddbs_crs, 32
- ddbs_difference (ddbs_binary_funs), 14
- ddbs_disjoint (ddbs_predicate), 103
- ddbs_distance (ddbs_measure_funs), 90
- ddbs_drivers, 34
- ddbs_drivers(), 139
- ddbs_drop_geometry, 35
- ddbs_dump, 36
- ddbs_endpoint_startpoint, 37
- ddbs_envelope, 39
- ddbs_envelope(), 95
- ddbs_equals (ddbs_predicate), 103
- ddbs_exterior_ring, 41
- ddbs_filter, 43
- ddbs_flip, 46
- ddbs_flip_coordinates, 48
- ddbs_force_2d (ddbs_force_dim), 49
- ddbs_force_3d (ddbs_force_dim), 49
- ddbs_force_4d (ddbs_force_dim), 49
- ddbs_force_dim, 49
- ddbs_generate_points, 52
- ddbs_geom_col, 55
- ddbs_geom_validation_funs, 55
- ddbs_geometry_type, 54
- ddbs_get_ngeometries (ddbs_get_npoints), 58
- ddbs_get_npoints, 58
- ddbs_get_resources (ddbs_set_resources), 122
- ddbs_glimpse, 60
- ddbs_has_dim, 61
- ddbs_has_m (ddbs_has_dim), 61
- ddbs_has_z (ddbs_has_dim), 61
- ddbs_install, 64
- ddbs_interpolate_aw, 65
- ddbs_intersection (ddbs_binary_funs), 14

- ddbs_intersects (ddbs_predicate), 103
- ddbs_intersects_extent
 - (ddbs_predicate), 103
- ddbs_is_closed
 - (ddbs_geom_validation_funs), 55
- ddbs_is_empty
 - (ddbs_geom_validation_funs), 55
- ddbs_is_ring
 - (ddbs_geom_validation_funs), 55
- ddbs_is_simple
 - (ddbs_geom_validation_funs), 55
- ddbs_is_valid
 - (ddbs_geom_validation_funs), 55
- ddbs_is_within_distance
 - (ddbs_predicate), 103
- ddbs_join, 68
- ddbs_length (ddbs_measure_funs), 90
- ddbs_line_endpoint
 - (ddbs_endpoint_startpoint), 37
- ddbs_line_interpolate, 71
- ddbs_line_merge, 73
- ddbs_line_startpoint
 - (ddbs_endpoint_startpoint), 37
- ddbs_line_substring, 75
- ddbs_list_tables, 77
- ddbs_load, 78
- ddbs_locate, 79
- ddbs_locate_along (ddbs_locate), 79
- ddbs_locate_between (ddbs_locate), 79
- ddbs_m (ddbs_xy), 141
- ddbs_make_envelope, 81
- ddbs_make_line, 83
- ddbs_make_polygon, 22, 84, 102
- ddbs_make_polygon(), 21, 22, 102
- ddbs_make_valid, 86
- ddbs_maximum_inscribed_circle, 88
- ddbs_measure_funs, 90
- ddbs_minimum_rotated_rectangle, 95
- ddbs_multi, 97
- ddbs_open_dataset, 98
- ddbs_options, 7, 11, 13, 15, 16, 18, 20,
 - 22–24, 26–29, 36–38, 40, 42, 44–51,
 - 53, 57, 59, 60, 62, 63, 66, 67, 69, 70,
 - 72, 74, 76, 80, 82–85, 87, 89, 91,
 - 96–98, 100, 102, 105, 113, 115, 117,
 - 119, 121, 124, 126, 128, 131, 134,
 - 136, 143
- ddbs_overlaps (ddbs_predicate), 103
- ddbs_perimeter (ddbs_measure_funs), 90
- ddbs_polygonize, 22, 86, 101
- ddbs_polygonize(), 22, 84, 86
- ddbs_predicate, 103, 105
- ddbs_quadkey, 107
- ddbs_read_meta, 109
- ddbs_read_table, 108, 110
- ddbs_register_table, 111, 140
- ddbs_remove_repeated_points, 112
- ddbs_rotate, 114
- ddbs_rotate_3d, 116
- ddbs_scale, 118
- ddbs_set_crs, 120
- ddbs_set_resources, 122
- ddbs_shear, 123
- ddbs_shift, 125
- ddbs_simplify, 127
- ddbs_sitrep, 129
- ddbs_stop_conn, 129
- ddbs_sym_difference (ddbs_binary_funs),
 - 14
- ddbs_touches (ddbs_predicate), 103
- ddbs_transform, 130
- ddbs_union (ddbs_union_funs), 132
- ddbs_union_agg (ddbs_union_funs), 132
- ddbs_union_funs, 132
- ddbs_voronoi, 135
- ddbs_within (ddbs_predicate), 103
- ddbs_within_properly (ddbs_predicate),
 - 103
- ddbs_write_dataset, 137
- ddbs_write_dataset(), 141
- ddbs_write_table, 140
- ddbs_x (ddbs_xy), 141
- ddbs_xy, 141
- ddbs_y (ddbs_xy), 141
- ddbs_z (ddbs_xy), 141
- dplyr::arrange(), 84
- dplyr::glimpse, 60
- is_duckspatial_df, 144
- sf::st_crs, 121, 130
- unnest, 109