Package 'duckspatial'

May 8, 2026

Type Package

Title R Interface to 'DuckDB' Database with Spatial Extension

Version 1.0.0

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, 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 2026-03-30 17:50:02 UTC

Contents

`as_duckspatial_df` *Convert objects to duckspatial_df*

Description

Convert objects to duckspatial_df

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)
geom_col	Geometry column name (default: "geom")
...	Additional arguments passed to methods

Value

A duckspatial_df object

---

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)

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

Arguments

x	A duckspatial_df object
...	Additional arguments passed to collect
as	Output format. One of:
	"sf" (Default) Returns an sf object with sfc geometry
	"tibble" Returns a tibble with geometry column dropped (fastest)
	"raw" Returns a tibble with geometry as raw WKB bytes
	"geoarrow" 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 <- ddbs_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 <- ddbs_collect(result)
plot(result_sf["AREA"])

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

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

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

## 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

```
ddbs_as_text(x, conn = NULL)

ddbs_as_wkb(x, conn = NULL)

ddbs_as_hexwkb(x, conn = NULL)

ddbs_as_geojson(x, conn = NULL)
```

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.
conn	A connection object to a DuckDB database. If NULL, 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:

- ddbs_as_text() returns a character vector of WKT geometries
- ddbs_as_wkb() returns a list of raw vectors (binary WKB)
- ddbs_as_hexwkb() returns a character vector of HEXWKB strings
- ddbs_as_geojson() returns a character vector of GeoJSON strings

Examples

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

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

ddbs_as_text(argentina_ddbs)
ddbs_as_wkb(argentina_ddbs)
```

```
ddbs_as_hexwkb(argentina_ddbs)
ddbs_as_geojson(argentina_ddbs)

## End(Not run)
```

`ddbs_as_points` *Generate point geometries from coordinates*

Description

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

Usage

```
ddbs_as_points(
  x,
  coords = c("lon", "lat"),
  crs = "EPSG:4326",
  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.
`coords`	Character vector of length 2 specifying the names of the longitude and latitude columns (or X and Y coordinates). Defaults to `c("lon", "lat")`.
`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).
`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:

- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") 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 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
cities_df2 <- data.frame(
  city = c("Mendoza", "Tucumán"),
  longitude = c(-68.8272, -65.2226),
  latitude = c(-32.8895, -26.8241)
)

ddbs_as_points(cities_df2, coords = c("longitude", "latitude"))


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

# create a duckdb connection and write data
conn <- duckspatial::ddbs_create_conn()
```

```
DBI::dbWriteTable(conn, "cities_tbl", cities_df, overwrite = TRUE)

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

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

## End(Not run)
```

---

ddbs_bbox *Get the bounding box of geometries*

Description

Returns the minimal rectangle that encloses the geometry

Usage

```
ddbs_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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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

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
)
```

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 — 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

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

mode — Character. Controls the return type. Options:

- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") 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

These functions perform different geometric set operations:

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

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

ddbs_sym_difference Returns the portions of both geometries that do not overlap with each other. Equivalent to (A - B) UNION (B - A).

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:
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 <- ddbs_intersection(x, y)
plot(st_geometry(result_intersect), col = "purple",
     main = "Intersection")

# Difference: part of x not in y (L-shaped area)
result_diff <- ddbs_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 <- ddbs_sym_difference(x, y)
```

```
plot(st_geometry(result_symdiff), col = "orange",
     main = "Symmetric Difference")

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

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

# Perform operations with connection
ddbs_intersection("poly_x", "poly_y", conn)
ddbs_difference("poly_x", "poly_y", conn)
ddbs_sym_difference("poly_x", "poly_y", conn)

# Save results to database table
ddbs_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

```
ddbs_boundary(
  x,
  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.

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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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")

# boundary
b <- ddbs_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

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.
distance	a numeric value specifying the buffer distance. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)
num_triangles	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.
cap_style	a character string specifying the cap style. Must be one of "CAP_ROUND" (default), "CAP_FLAT", or "CAP_SQUARE". Case-insensitive.
join_style	a character string specifying the join style. Must be one of "JOIN_ROUND" (default), "JOIN_MITRE", or "JOIN_BEVEL". Case-insensitive.
mitre_limit	a numeric value specifying the mitre limit ratio. Only applies when join_style is "JOIN_MITRE". It is the ratio of the distance from the corner to the mitre point to the corner radius. Default is 1.0.

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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)

## 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_vector(conn, argentina_ddbs, "argentina")

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

## buffer with custom parameters
ddbs_buffer(conn = conn, "argentina", distance = 1,
```

```
            num_triangles = 16, cap_style = "CAP_SQUARE")

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

## End(Not run)
```

`ddbs_build_area` *Build polygon areas from multiple linestrings*

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 `ddbs_make_polygon()`.

Usage

```
ddbs_build_area(
  x,
  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.
`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: • `"duckspatial"` (default): Lazy spatial data frame backed by dbplyr/DuckDB • `"sf"`: Eagerly collected sf object (uses memory) Can be set globally via `ddbs_options(mode = "...")` 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).

See Also

ddbs_make_polygon(), ddbs_polygonize()

Other polygon construction: ddbs_make_polygon(), ddbs_polygonize()

---

ddbs_centroid *Calculates the centroid of geometries*

Description

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

Usage

```
ddbs_centroid(
  x,
  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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)

# 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_vector(conn, argentina_ddbs, "argentina")

## centroid
ddbs_centroid("argentina", conn)

## centroid without using a connection
```

```
ddbs_centroid(argentina_ddbs)

## End(Not run)
```

---

ddbs_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

```
ddbs_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 <- ddbs_open_dataset(
  system.file("spatial/countries.geojson", package = "duckspatial")
```

```
)

# Complex pipeline - ddbs_compute() caches intermediate result
cached <- countries |>
  filter(CNTR_ID %in% c("DE", "FR", "IT")) |>
  ddbs_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 |>
  ddbs_filter(other_layer, predicate = "intersects") |>
  st_as_sf()

## End(Not run)
```

`ddbs_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

```
ddbs_concave_hull(
  x,
  ratio = 0.5,
  allow_holes = 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.

ratio	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.
allow_holes	Boolean. If TRUE (the default), it allows the output to contain holes.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)
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)
) |>
  ddbs_as_spatial(coords = c("x", "y")) |>
  ddbs_combine()

# option 1: passing ddbs or sf objects
```

```
output1 <- duckspatial::ddbs_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::ddbs_create_conn()

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

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

plot(output2)


## End(Not run)
```

`ddbs_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

```
ddbs_convex_hull(
  x,
  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.

`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:

- `"duckspatial"` (default): Lazy spatial data frame backed by dbplyr/DuckDB
- `"sf"`: Eagerly collected sf object (uses memory)

Can be set globally via `ddbs_options(mode = "...")` 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)
library(sf)

# 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")
)

# option 1: passing sf objects
output1 <- duckspatial::ddbs_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::ddbs_create_conn()

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

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

plot(output2["CNTR_NAME"])

## End(Not run)
```

`ddbs_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

```
ddbs_create_conn(dbdir = "memory", threads = NULL, memory_limit_gb = NULL, ...)
```

Arguments

`dbdir`	String. Either `"tempdir"`, `"memory"`, or file path with `.duckdb` or `.db` 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).
`...`	Additional parameters to be passed to `dbConnect`

Value

A duckdb_connection

Examples

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

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

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

# create a connection with 1 thread and 2GB memory limit
conn <- ddbs_create_conn(threads = 1, memory_limit_gb = 2)
ddbs_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'
ddbs_crs(x, name, ...)

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

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

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

## Default S3 method:
ddbs_crs(x, ...)
```

Arguments

x	An object containing spatial data. Can be: • 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 <- ddbs_open_dataset(system.file("shape/nc.shp", package = "sf"))
ddbs_crs(nc)

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

# Method 3: table name in database
conn <- ddbs_create_conn(dbdir = "memory")
ddbs_write_table(conn, nc_sf, "nc_table")
ddbs_crs(conn, "nc_table")
ddbs_stop_conn(conn)

## End(Not run)
```

ddbs_drivers *Get list of GDAL drivers and file formats*

Description

Get list of GDAL drivers and file formats

Usage

```
ddbs_drivers(conn = NULL)
```

Arguments

conn	A DBIConnection object to a DuckDB database. If not specified (conn = NULL), uses the default connection created with ddbs_default_conn() (or a temporary one).

Value

data.frame

Examples

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

## database setup
conn <- ddbs_create_conn()

## check drivers
ddbs_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)
```

---

ddbs_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

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

```
)

ddbs_endpoint(
  x,
  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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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

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

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

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

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

## without using a connection
ddbs_startpoint(rivers_ddbs)
ddbs_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

```
ddbs_envelope(
  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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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

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 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)

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

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

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

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

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

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

# create a new table with envelopes
ddbs_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	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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)

# 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: • 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:

- 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

predicate — A geometry predicate function. Defaults to intersects, a wrapper of ST_Intersects. See 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:

- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") 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")
)

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

# Lazy filter - computation stays in DuckDB
neighbors <- ddbs_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 <- ddbs_filter(countries_sf, argentina_sf, predicate = "touches")


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

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

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

## End(Not run)
```

ddbs_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

```
ddbs_flip(
  x,
  direction = c("horizontal", "vertical"),
  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.
direction	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)
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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")

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

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

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

## End(Not run)
```

`ddbs_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

```
ddbs_flip_coordinates(
  x,
  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.
`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: • `"duckspatial"` (default): Lazy spatial data frame backed by dbplyr/DuckDB • `"sf"`: Eagerly collected sf object (uses memory) Can be set globally via `ddbs_options(mode = "...")` 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)

# 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_vector(conn, argentina_ddbs, "argentina")

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

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

## End(Not run)
```

---

ddbs_force_dim *Force geometry dimensions*

Description

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

Usage

```
ddbs_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	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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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.
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 convered in Z dimension
var_m	A numeric variable in x to be convered 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(ISO3_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.

n	Number of random points to generate within each geometry
seed	A number for the random number generator
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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")

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

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

## End(Not run)
```

`ddbs_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

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

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.
`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.

Value

A factor with geometry type(s)

Examples

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

## read data
```

```
countries_ddbs <- ddbs_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` *Get the geometry column name*

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`
Geometry validation functions

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	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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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

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 materializating a vector in R (see details).

- ddbs_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.
- ddbs_is_simple() determines whether geometries are simple, meaning they are free of self-intersections. For example, a linestring that crosses itself is not simple.
- ddbs_is_ring() checks if a linestring geometry is closed (first and last points are identical) and simple (no self-intersections), forming a valid ring.
- ddbs_is_empty() tests whether a geometry is empty, containing no points. Empty geometries are valid but represent the absence of spatial information.
- ddbs_is_closed() determines if a linestring geometry is closed, meaning the first and last coordinates are identical. Unlike ddbs_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_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 <- ddbs_create_conn(dbdir = "memory")

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

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

## glimpse the inserted table
ddbs_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

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

ddbs_has_m(
  x,
  by_feature = TRUE,
  new_column = "has_m",
  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.
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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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

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(ISO3_CODE != "ATA")

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

## End(Not run)
```

ddbs_install *Checks and installs the Spatial extension*

Description

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

Usage

```
ddbs_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
ddbs_install(conn)

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

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

ddbs_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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB

- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") or per-function via this argument. Per-function overrides global setting.

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

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$sid <- 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 <- ddbs_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 <- ddbs_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)
```

---

ddbs_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

```
ddbs_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)

- 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:

- 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:

- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") 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_list_tables` *Check tables and schemas inside a database*

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 <- ddbs_create_conn(dbdir = "memory")

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

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

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

## list tables in the database
ddbs_list_tables(conn)

## End(Not run)
```

---

ddbs_load *Loads the Spatial extension*

Description

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

Usage

```
ddbs_load(conn, quiet = FALSE, extension = "spatial")
```

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"

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 exntesion
ddbs_install(conn)
ddbs_load(conn)

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

## End(Not run)
```

---

ddbs_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 ddbs_polygonize() or ddbs_build_area() instead.

Usage

```
ddbs_make_polygon(
  x,
  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.

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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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).

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 <- ddbs_create_conn(dbdir = "memory")

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

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

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

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

## End(Not run)
```

`ddbs_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

```
ddbs_make_valid(
  x,
  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.
`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: • `"duckspatial"` (default): Lazy spatial data frame backed by dbplyr/DuckDB • `"sf"`: Eagerly collected sf object (uses memory)

	Can be set globally via ddbs_options(mode = "...") 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)

# 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_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
)

ddbs_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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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 `ddbs_distance` with POINT geometries: defaults to `"haversine"`
- For `ddbs_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 `ddbs_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 `ddbs_area`, `ddbs_length`, and `ddbs_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 `ddbs_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 <- ddbs_create_conn(dbdir = "memory")

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

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

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

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

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

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

# Calculate area from sf object directly
ddbs_area(countries_ddbs)

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

# Calculate total area
countries_ddbs |>
  mutate(area = ddbs_area(geom)) |>
  summarise(
    area = sum(area),
    geom = ddbs_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
ddbs_write_table(conn, rivers_ddbs, "rivers")

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

# Calculate length with custom column name
ddbs_length(rivers_ddbs, new_column = "length_meters")

# Calculate length by river name
rivers_ddbs |>
  ddbs_union_agg("RIVER_NAME") |>
  ddbs_length()

# Add length within dplyr
rivers_ddbs |>
  mutate(length = ddbs_length(geometry))


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

# Calculate perimeter (returns sf object with perimeter column)
ddbs_perimeter(countries_ddbs)

# Calculate perimeter within dplyr
countries_ddbs |>
  mutate(perim = ddbs_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)
) |>
  ddbs_as_spatial(coords = c("x", "y"), crs = "EPSG:4326")

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

# Option 2: Explicitly specify distance type
dist_matrix_harv <- ddbs_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_multi` *Convert geometries to multi-type*

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`	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.
`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:

- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") 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
countries_ddbs <- ddbs_open_dataset(
  system.file("spatial/countries.geojson",
  package = "duckspatial")
)

## convert to multi-type
ddbs_multi(countries_ddbs)

## End(Not run)
```

---

ddbs_open_dataset *Open spatial dataset lazily via DuckDB*

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, 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.
`layer`	Layer name or index to read (ST_Read only). Default is NULL (first layer).
`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).

---

ddbs_options *Get or set global duckspatial options*

Description

Get or set global duckspatial options

Usage

```
ddbs_options(output_type = NULL, mode = NULL)
```

Arguments

output_type	Character string. Controls the default return type for ddbs_collect. Must be one of: • "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.

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 *Assemble polygons from multiple linestrings*

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	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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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).

See Also

ddbs_make_polygon(), ddbs_build_area()

Other polygon construction: ddbs_build_area(), ddbs_make_polygon()

---

ddbs_predicate *Evaluate spatial predicates between geometries*

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, ...)

ddbs_covered_by(x, y, ...)

ddbs_intersects_extent(x, y, ...)

ddbs_contains_properly(x, y, ...)

ddbs_within_properly(x, y, ...)
```

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	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
predicate	A geometry predicate function. Defaults to intersects, a wrapper of ST_Intersects. See 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
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.
sparse	A logical value. If TRUE, it returns a sparse table/list. If FALSE, it returns a wide table/matrix.

distance	a numeric value specifying the distance for ST_DWithin. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)
mode	Character. Controls the return type. Options: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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.
...	Passed to ddbs_predicate

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 <- ddbs_create_conn(dbdir = "memory")

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

rivers_ddbs <- ddbs_open_dataset(
  system.file("spatial/rivers.geojson",
  package = "duckspatial")
) |>
  ddbs_transform(ddbs_crs(countries_ddbs))

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

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

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

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

## Example 3: Find rivers that don't intersect countries
ddbs_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: • 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.
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: • "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 ddbs_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 <- ddbs_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_table` *Reads a vectorial table from DuckDB into R*

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 `ddbs_write_table` and temporary Arrow views created by `ddbs_register_table`.

Usage

```
ddbs_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 <- ddbs_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
ddbs_write_vector(conn, sf_points, "points")
ddbs_read_table(conn, "points")

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

## disconnect from db
ddbs_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 `ddbs_write_table` for workflows that do not require data to be permanently materialized in the database.

Usage

```
ddbs_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_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	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.
angle	a numeric value specifying the rotation angle
units	character string specifying angle units: "degrees" (default) or "radians"
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.
center_x	numeric value for the X coordinate of rotation center. If NULL, rotates around the centroid of each geometry
center_y	numeric value for the Y coordinate of rotation center. If NULL, rotates around the centroid of each geometry
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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")

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

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

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

## End(Not run)
```

---

ddbs_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

```
ddbs_rotate_3d(
  x,
  angle,
  units = c("degrees", "radians"),
  axis = "x",
  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.
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 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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)
library(dplyr)

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

## read 3D data
countries_ddbs <- ddbs_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")

## rotate 45 degrees around X axis (pitch)
ddbs_rotate_3d(conn = conn, "countries", angle = 45, axis = "x")

## rotate 90 degrees around Y axis (yaw)
ddbs_rotate_3d(conn = conn, "countries", angle = 30, axis = "y")

## rotate 180 degrees around Z axis (roll)
ddbs_rotate_3d(conn = conn, "countries", angle = 180, axis = "z")

## rotate without using a connection
ddbs_rotate_3d(countries_ddbs, angle = 45, axis = "z")

## End(Not run)
```

---

ddbs_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

```
ddbs_scale(
  x,
  x_scale = 1,
  y_scale = 1,
  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_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 sf object
mode	Character. Controls the return type. Options: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)
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")
) |>
  filter(CNTR_ID %in% c("PT", "ES", "FR", "IT"))

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

## scale to 150% in both directions
ddbs_scale(conn = conn, "countries", x_scale = 1.5, y_scale = 1.5)

## scale to 200% horizontally, 50% vertically
ddbs_scale(conn = conn, "countries", x_scale = 2, y_scale = 0.5)

## scale all features together (default)
ddbs_scale(countries_ddbs, x_scale = 1.5, y_scale = 1.5, by_feature = FALSE)

## scale each feature independently
ddbs_scale(countries_ddbs, x_scale = 1.5, y_scale = 1.5, by_feature = TRUE)


## End(Not run)
```

---

ddbs_set_resources *Get or set connection resources*

Description

Configure technical system settings for a DuckDB connection, such as memory limits and CPU threads.

Usage

```
ddbs_set_resources(conn, threads = NULL, memory_limit_gb = NULL)

ddbs_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 ddbs_set_resources(), invisibly returns a list containing the current system settings; for ddbs_get_resources(), visibly returns the same list for direct inspection.

Examples

```
## Not run:
# Create a connection
conn <- ddbs_create_conn()

# Set resources: 1 thread and 4GB
ddbs_set_resources(conn, threads = 1, memory_limit_gb = 4)

# Check current settings
ddbs_get_resources(conn)

ddbs_stop_conn(conn)

## End(Not run)
```

---

ddbs_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

```
ddbs_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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)
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")
) |>
  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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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	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.
tolerance	Tolerance distance for simplification. Larger values result in more simplified geometries.
preserve_topology	If FALSE, uses the Douglas-Peucker algorithm, which reduces the vertices by removing points that are within a given distance. If TRUE, 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 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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)
```

```
# 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")

## simplify with tolerance of 0.01
ddbs_simplify("countries", tolerance = 0.01, conn = conn)

## simplify without using a connection
ddbs_simplify(countries_ddbs, tolerance = 0.01)

## End(Not run)
```

`ddbs_sitrep` *Report duckspatial configuration status*

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 *Close a DuckDB connection*

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 <- ddbs_create_conn(dbdir = "memory")

## close the connection
ddbs_stop_conn(conn)

## End(Not run)
```

ddbs_transform *Transform the coordinate reference system of geometries*

Description

Converts geometries to a different coordinate reference system (CRS), updating their coordinates accordingly.

Usage

```
ddbs_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") • An sf object (uses its CRS) • Name of a DuckDB table (uses its CRS)
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
mode	Character. Controls the return type. Options: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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)

# 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_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
)

ddbs_union_agg(
  x,
  by,
  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.

y — Input spatial data. Can be:

- NULL (default): performs only the union of x
- 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

by_feature — Logical. When y is provided:

- 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 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

mode — Character. Controls the return type. Options:

- "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB
- "sf": Eagerly collected sf object (uses memory)

Can be set globally via ddbs_options(mode = "...") 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

ddbs_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.

ddbs_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.

ddbs_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 `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(ISO3_CODE != "ATA")

rivers_ddbs <- ddbs_open_dataset(
  system.file("spatial/rivers.geojson",
  package = "duckspatial")
) |>
 ddbs_transform("EPSG:4326")

## combine countries into a single MULTI-geometry
## (without solving boundaries)
combined_countries_ddbs <- ddbs_combine(countries_ddbs)

## combine countries into a single MULTI-geometry
## (solving boundaries)
```

```
union_countries_ddbs <- ddbs_union(countries_ddbs)

## union of geometries of two objects, into 1 geometry
union_countries_rivers_ddbs <- ddbs_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	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.
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory)

	Can be set globally via ddbs_options(mode = "...") 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)

## create a duckdb database in memory (with spatial extension)
conn <- ddbs_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)
) |>
  ddbs_as_spatial(coords = c("x", "y"), crs = 4326) |>
  ddbs_combine()

## create voronoi diagrama
ddbs_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. Supports Parquet (native) and various GDAL spatial formats. Format is auto-detected from file extension for common formats, or can be specified explicitly via gdal_driver.

Usage

```
ddbs_write_dataset(
  data,
  path,
  gdal_driver = NULL,
  conn = NULL,
  overwrite = FALSE,
  crs = NULL,
  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
)
```

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: • .geojson, .json → "GeoJSON" • .shp → "ESRI Shapefile" • .gpkg → "GPKG" • .fgb → "FlatGeobuf" • .kml → "KML" • .gpx → "GPX" • .gml → "GML" • .sqlite → "SQLite" For **non-standard file extensions** (e.g., .dat, .xyz) or to **explicitly override** auto-detection, specify the exact driver name as it appears in ddbs_drivers()$short_name. Examples: gdal_driver = "GeoJSON", gdal_driver = "ESRI Shapefile". **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). 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.

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.
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.

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 *Write an SF Object to a DuckDB Database*

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.

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 <- ddbs_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 X and Y coordinates from geometries*

Description

`ddbs_x()` extracts the X coordinate (longitude) and `ddbs_y()` extracts the Y coordinate (latitude) from point geometries, adding them as a new column to the dataset.

Usage

```
ddbs_x(
  x,
  new_column = "X",
  conn = NULL,
  name = NULL,
```

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

ddbs_y(
  x,
  new_column = "Y",
  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.
new_column	Name of the new column to store the extracted coordinate. Defaults to "X" for ddbs_x() and "Y" for ddbs_y().
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: • "duckspatial" (default): Lazy spatial data frame backed by dbplyr/DuckDB • "sf": Eagerly collected sf object (uses memory) Can be set globally via ddbs_options(mode = "...") 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")
)

## extract coordinates without using a connection
ddbs_x(argentina_ddbs)
ddbs_y(argentina_ddbs)

## 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