Package 'duckspatial'

January 10, 2026

Type Package

Title R Interface to 'DuckDB' Database with Spatial Extension

Version 0.9.0

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, duckdb, geoarrow, DBI, glue, sf, wk, uuid, rlang, lifecycle

Suggests areal, bench, dplyr, ggplot2 (>= 3.3.1), knitr, lwgeom, quadkeyr, rmarkdown, terra, testthat (>= 3.0.0)

VignetteBuilder knitr

RoxygenNote 7.3.3

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2026-01-10 15:00:07 UTC

Contents

`ddbs_area` *Calculates the area of geometries*

Description

Calculates the area of geometries from a DuckDB table or a `sf` object Returns the result as an `sf` object with an area column or creates a new table in the database. Note: Area units depend on the CRS of the input geometries (e.g., square meters for projected CRS, or degrees for geographic CRS).

Usage

```
ddbs_area(
  x,
  conn = NULL,
  name = NULL,
  new_column = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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
new_column	Name of the new column to create on the input data. If NULL, the function will return a vector with the result
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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 vector, an sf object or TRUE (invisibly) for table creation

Examples

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

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

## read data
argentina_sf <- st_read(system.file("spatial/argentina.geojson", package = "duckspatial")) |>
    st_transform("EPSG:3857")

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

## calculate area (returns sf object with area column)
ddbs_area("argentina", conn)

## calculate area with custom column name
ddbs_area("argentina", conn, new_column = "area_sqm")

## create a new table with area calculations
ddbs_area("argentina", conn, name = "argentina_with_area")

## calculate area in a sf object
ddbs_area(argentina_sf)

## End(Not run)
```

ddbs_as_hexwkb *Convert geometries to hexadecimal Well-Known Binary (HEXWKB) format*

Description

Converts spatial geometries to their hexadecimal Well-Known Binary (HEXWKB) representation. This function wraps DuckDB's ST_AsHEXWKB spatial function.

Usage

```
ddbs_as_hexwkb(x, conn = NULL, quiet = FALSE)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

HEXWKB is a hexadecimal string representation of Well-Known Binary (WKB) format. This encoding is human-readable (unlike raw WKB) while maintaining the compact binary structure. HEXWKB is commonly used in databases and web services for transmitting spatial data as text strings.

Value

A character vector containing hexadecimal-encoded WKB representations of the geometries

Examples

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

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

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

## store in duckdb
```

```
ddbs_write_vector(conn, argentina_sf, "argentina")

## convert geometries to HEXWKB
hexwkb_text <- ddbs_as_hexwkb(conn = conn, "argentina")

## convert without using a connection
hexwkb_text <- ddbs_as_hexwkb(argentina_sf)

## End(Not run)
```

---

ddbs_as_text *Convert geometries to Well-Known Text (WKT) format*

Description

Converts spatial geometries to their Well-Known Text (WKT) representation. This function wraps DuckDB's ST_AsText spatial function.

Usage

```
ddbs_as_text(x, conn = NULL, quiet = FALSE)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

Well-Known Text (WKT) is a text markup language for representing vector geometry objects. This function is useful for exporting geometries in a portable text format that can be used with other spatial tools and databases.

Value

A character vector containing WKT representations of the geometries

Examples

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

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

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

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

## convert geometries to WKT
wkt_text <- ddbs_as_text(conn = conn, "argentina")

## convert without using a connection
wkt_text <- ddbs_as_text(argentina_sf)

## End(Not run)
```

---

ddbs_as_wkb *Convert geometries to Well-Known Binary (WKB) format*

Description

Converts spatial geometries to their Well-Known Binary (WKB) representation. This function wraps DuckDB's ST_AsWkb spatial function.

Usage

```
ddbs_as_wkb(x, conn = NULL, quiet = FALSE)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

Well-Known Binary (WKB) is a binary representation of vector geometry objects. WKB is more compact than WKT and is commonly used for efficient storage and transfer of spatial data between systems. Each geometry is returned as a raw vector of bytes.

Value

A list of raw vectors, where each element contains the WKB representation of a geometry

Examples

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

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

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

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

## convert geometries to WKB
wkb_list <- ddbs_as_wkb(conn = conn, "argentina")

## convert without using a connection
wkb_list <- ddbs_as_wkb(argentina_sf)

## End(Not run)
```

---

ddbs_bbox *Returns the minimal bounding box enclosing the input geometry*

Description

Returns the minimal bounding box enclosing the input geometry from a `sf` object or a DuckDB table. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_bbox(
  x,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
by_feature	Boolean. The function defaults to FALSE, and returns a single bounding box for x. If TRUE, it return one bounding box for each feature.
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

# option 1: passing sf objects
ddbs_bbox(argentina_sf)


## 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_vector(conn, argentina_sf, "argentina_tbl", overwrite = TRUE)

output2 <- ddbs_bbox(
    conn = conn,
    x = "argentina_tbl",
    name = "argentina_bbox"
```

```
)

DBI::dbReadTable(conn, "argentina_bbox")

## End(Not run)
```

---

ddbs_boundary *Returns the boundary of geometries*

Description

Returns the boundary of geometries from a `sf` object or a DuckDB table. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_boundary(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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
`crs`	The coordinates reference system of the data. Specify if the data doesn't have a `crs_column`, and you know the CRS.
`crs_column`	a character string of length one specifying the column storing the CRS (created automatically by `ddbs_write_vector`). Set to `NULL` if absent.
`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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

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

## End(Not run)
```

---

ddbs_buffer *Creates a buffer around geometries*

Description

Calculates the buffer of geometries from a DuckDB table using the spatial extension. Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_buffer(
  x,
  distance,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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)
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

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

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

## End(Not run)
```

ddbs_centroid *Calculates the centroid of geometries*

Description

Calculates the centroids of geometries from a DuckDB table using the spatial extension. Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_centroid(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

## centroid
ddbs_centroid("argentina", conn)

## centroid without using a connection
ddbs_centroid(argentina_sf)

## End(Not run)
```

---

ddbs_combine *Combine geometries into a single MULTI-geometry*

Description

Combines all geometries from a `sf` object or a DuckDB table into a single MULTI-geometry using the spatial extension. This is equivalent to `sf::st_combine()`. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_combine(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	character string specifying the name of the CRS column. Default is "crs_duckspatial"
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

# combine all geometries into one
ddbs_combine(conn = conn, "countries")

# combine without using a connection
ddbs_combine(countries_sf)

# store result in a new table
ddbs_combine(conn = conn, "countries", name = "countries_combined")

## End(Not run)
```

ddbs_concave_hull *Returns the concave hull enclosing the geometry*

Description

Returns the concave hull enclosing the geometry from an `sf` object or from a DuckDB table using the spatial extension. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_concave_hull(
  x,
  ratio = 0.5,
  allow_holes = TRUE,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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
`crs`	The coordinates reference system of the data. Specify if the data doesn't have a `crs_column`, and you know the CRS.
`crs_column`	a character string of length one specifying the column storing the CRS (created automatically by `ddbs_write_vector`). Set to `NULL` if absent.
`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

an sf object or TRUE (invisibly) for table creation

Examples

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

# create points data
n <- 5
points_sf <- data.frame(
    id = 1,
    x = runif(n, min = -180, max = 180),
    y = runif(n, min = -90, max = 90)
    ) |>
    sf::st_as_sf(coords = c("x", "y"), crs = 4326) |>
    st_geometry() |>
    st_combine() |>
    st_cast("MULTIPOINT") |>
    st_as_sf()

# option 1: passing sf objects
output1 <- duckspatial::ddbs_concave_hull(x = points_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_sf, "points_tbl")

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

plot(output2)


## End(Not run)
```

---

ddbs_contains *Spatial contains predicate*

Description

Tests if geometries in x contain geometries in y. Returns TRUE if geometry x completely contains geometry y.

Usage

```
ddbs_contains(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "contains".

Value

A list where each element contains indices (or IDs) of geometries in y that are contained by the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_contains(countries_sf, rivers_sf, id_x = "NAME_ENGL", id_y = "RIVER_NAME")

## End(Not run)
```

---

ddbs_contains_properly

Spatial contains properly predicate

Description

Tests if geometries in x properly contain geometries in y. Returns TRUE if geometry y is completely inside geometry x and does not touch its boundary.

Usage

```
ddbs_contains_properly(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.

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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "contains_properly".

Value

A list where each element contains indices (or IDs) of geometries in y that are properly contained by the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_contains_properly(countries_sf, rivers_sf, id_x = "NAME_ENGL", id_y = "RIVER_NAME")

## End(Not run)
```

---

ddbs_convex_hull *Returns the convex hull enclosing the geometry*

Description

Returns the convex hull enclosing the geometry from an sf object or from a DuckDB table using the spatial extension. Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_convex_hull(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

# option 1: passing sf objects
```

```
output1 <- duckspatial::ddbs_convex_hull(x = argentina_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_sf, "argentina_tbl")

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

plot(output2["CNTR_NAME"])

## End(Not run)
```

---

ddbs_covered_by *Spatial covered by predicate*

Description

Tests if geometries in x are covered by geometries in y. Returns TRUE if geometry x is completely covered by geometry y (no point of x lies outside y).

Usage

```
ddbs_covered_by(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.

conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "covered_by".

Value

A list where each element contains indices (or IDs) of geometries in y that cover the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_covered_by(rivers_sf, countries_sf, id_x = "RIVER_NAME", id_y = "NAME_ENGL")

## End(Not run)
```

ddbs_covers *Spatial covers predicate*

Description

Tests if geometries in x cover geometries in y. Returns TRUE if geometry x completely covers geometry y (no point of y lies outside x).

Usage

```
ddbs_covers(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "covers".

Value

A list where each element contains indices (or IDs) of geometries in y that are covered by the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_covers(countries_sf, rivers_sf, id_x = "NAME_ENGL")

## 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")
```

Arguments

dbdir	String. Either "tempdir" or "memory". Defaults to "memory".

Value

A duckdb_connection

Examples

```
# 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")
```

---

ddbs_create_schema *Check and create schema*

Description

Check and create schema

Usage

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

Arguments

conn	A connection 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_crosses *Spatial crosses predicate*

Description

Tests if geometries in x cross geometries in y. Returns TRUE if geometries have some but not all interior points in common.

Usage

```
ddbs_crosses(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "crosses".

Value

A list where each element contains indices (or IDs) of geometries in y that cross the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_crosses(rivers_sf, countries_sf, id_x = "RIVER_NAME", id_y = "NAME_ENGL")

## End(Not run)
```

---

ddbs_crs *Check CRS of a table*

Description

Check CRS of a table

Usage

```
ddbs_crs(conn, name, crs_column = "crs_duckspatial")
```

Arguments

conn	A connection 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.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector)

Value

CRS object

Examples

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

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

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

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

## check CRS
ddbs_crs(conn, "countries")
```

---

ddbs_difference *Calculates the difference of two geometries*

Description

Calculates the geometric difference of two geometries, and returns a `sf` object or creates a new table

Usage

```
ddbs_difference(
  x,
  y,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`. Data is returned from this object.
y	A table with geometry column within the DuckDB database
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

An sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, countries_sf, "countries")
ddbs_write_vector(conn, argentina_sf, "argentina")

## difference with a connection
ddbs_difference("countries", "argentina", conn)

## difference without a connection
ddbs_difference(countries_sf, argentina_sf)

## End(Not run)
```

---

ddbs_disjoint *Spatial disjoint predicate*

Description

Tests if geometries in x are disjoint from geometries in y. Returns TRUE if geometries have no points in common.

Usage

```
ddbs_disjoint(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "disjoint".

Value

A list where each element contains indices (or IDs) of geometries in y that are disjoint from the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

```
library(duckspatial)
library(sf)

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

ddbs_disjoint(countries_sf, rivers_sf, id_x = "NAME_ENGL")

## End(Not run)
```

---

ddbs_distance *Returns the distance between two geometries*

Description

Returns the planar or haversine distance between two geometries, and returns a `data.frame` object or creates a new table in a DuckDB database.

Usage

```
ddbs_distance(x, y, dist_type = "haversine", conn = NULL, quiet = FALSE)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`. Data is returned from this object.
`y`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`.
`dist_type`	String. One of `c("planar", "haversine")`. Defaults to `"haversine"` and returns distance in meters, but the input is expected to be in WGS84 (EPSG:4326) coordinates. The option `"haversine"` only accepts `POINT` geometries. When `dist_type = "planar"`, distances estimates are in the same unit as the coordinate reference system (CRS) of the input.
`conn`	A connection object to a DuckDB database. If `NULL`, the function runs on a temporary DuckDB database.
`quiet`	A logical value. If `TRUE`, suppresses any informational messages. Defaults to `FALSE`.

Value

A `data.frame` object or `TRUE` (invisibly) for table creation

Examples

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

# create points data
n <- 10
points_sf <- 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)

# option 1: passing sf objects
output1 <- duckspatial::ddbs_distance(
    x = points_sf,
    y = points_sf,
    dist_type = "haversine"
)

head(output1)


## option 2: passing the names of tables in a duckdb db and output as sf

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

# write sf to duckdb
ddbs_write_vector(conn, points_sf, "points", overwrite = TRUE)

output2 <- ddbs_distance(
    conn = conn,
    x = "points",
    y = "points",
    dist_type = "haversine"
)
head(output2)


## 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)
```

Arguments

conn	A connection object to a DuckDB database

Value

data.frame

Examples

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

## database setup
conn <- dbConnect(duckdb())
ddbs_install(conn)
ddbs_load(conn)

## check drivers
ddbs_drivers(conn)
```

---

ddbs_envelope *Returns the envelope (bounding box) of geometries*

Description

Returns the minimum bounding rectangle (envelope) of geometries from a sf object or a DuckDB table. Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_envelope(
  x,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
by_feature	Logical. If TRUE, returns one envelope per feature. If FALSE (default), returns a single envelope for all geometries combined.
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

```
# store in duckdb
ddbs_write_vector(conn, argentina_sf, "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_equals *Spatial equals predicate*

Description

Tests if geometries in x are spatially equal to geometries in y. Returns TRUE if geometries are topologically equivalent (same shape and location).

Usage

```
ddbs_equals(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "equals".

Value

A list where each element contains indices (or IDs) of geometries in y that are equal to the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_equals(countries_sf, countries_sf, id_x = "NAME_ENGL")

## End(Not run)
```

---

ddbs_exterior_ring *Extracts the exterior ring of polygon geometries*

Description

Returns the exterior ring (outer boundary) of polygon geometries from a DuckDB table using the spatial extension. For multi-polygons, returns the exterior ring of each polygon component. Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_exterior_ring(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
```

```
ddbs_write_vector(conn, countries_sf, "countries")

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

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

## End(Not run)
```

---

ddbs_filter *Performs spatial filter of two geometries*

Description

Filters data spatially based on a spatial predicate

Usage

```
ddbs_filter(
  x,
  y,
  predicate = "intersects",
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  distance = NULL,
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	Y table with geometry column within the DuckDB database
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.
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.

crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
distance	a numeric value specifying the distance for ST_DWithin. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)
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

An sf object or TRUE (invisibly) for table creation

Examples

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

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

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

```
## store in duckdb
ddbs_write_vector(conn, countries_sf, "countries")
ddbs_write_vector(conn, argentina_sf, "argentina")

## filter countries touching argentina
ddbs_filter(conn = conn, "countries", "argentina", predicate = "touches")

## filter without using a connection
ddbs_filter(countries_sf, argentina_sf, predicate = "touches")

## End(Not run)
```

---

ddbs_flip *Flip geometries horizontally or vertically*

Description

Flips (reflects) geometries around the centroid. Returns the result as an `sf` object or creates a new table in the database. This function is equivalent to `terra::flip()`.

Usage

```
ddbs_flip(
  x,
  direction = c("horizontal", "vertical"),
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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` (default), 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, argentina_sf, "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_sf, direction = "horizontal")

## End(Not run)
```

---

ddbs_generate_points *Generate random points within geometries*

Description

Generates random points within geometries from a DuckDB table using the spatial extension. Works similarly to generating random points within polygons in `sf`. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_generate_points(
  x,
  n,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`. Data is returned from this object.
`n`	Number of random points to generate within 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
`crs`	The coordinates reference system of the data. Specify if the data doesn't have a `crs_column`, and you know the CRS.
`crs_column`	a character string of length one specifying the column storing the CRS (created automatically by `ddbs_write_vector`). Set to `NULL` if absent.
`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

an `sf` object or `TRUE` (invisibly) for table creation

Examples

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

```
library(sf)

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

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

## store in duckdb
ddbs_write_vector(conn, argentina_sf, "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_sf, n = 100)

## End(Not run)
```

`ddbs_glimpse` *Check first rows of the data*

Description

Check first rows of the data

Usage

```
ddbs_glimpse(
  conn,
  name,
  crs = NULL,
  crs_column = "crs_duckspatial",
  quiet = FALSE
)
```

Arguments

`conn`	A connection 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.
`crs`	The coordinates reference system of the data. Specify if the data doesn't have a `crs_column`, and you know the CRS.
`crs_column`	a character string of length one specifying the column storing the CRS (created automatically by `ddbs_write_vector`). Set to `NULL` if absent.
`quiet`	A logical value. If `TRUE`, suppresses any informational messages. Defaults to `FALSE`.

Value

sf object

Examples

```
library(duckspatial)
library(sf)

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

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

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

ddbs_glimpse(conn, "argentina")
```

---

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)
```

Arguments

conn	A connection 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.

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)

# 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",
  output = "sf",
  keep_NA = TRUE,
  na.rm = FALSE,
  join_crs = NULL,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  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.
output	Character. One of "sf" (default) or "tibble". • "sf": The result includes the geometry column of the target. • "tibble": The result **excludes the geometry column**. This is significantly faster and consumes less storage. **Note:** This argument also controls the schema of the created table if name is provided.
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

- If `name` is `NULL` (default): Returns an `sf` object (if `output="sf"`) or a `tibble` (if `output="tibble"`).
- If `name` is provided: Returns `TRUE` invisibly and creates a persistent table in the DuckDB database.
  - If `output="sf"`, the table **includes** the geometry column.
  - If `output="tibble"`, the table **excludes** the geometry column (pure attributes).

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"
)

# 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"
)

# 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_intersection *Calculates the intersection of two geometries*

Description

Calculates the intersection of two geometries, and return a `sf` object or creates a new table

Usage

```
ddbs_intersection(
  x,
  y,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	A table with geometry column within the DuckDB database
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

## read data
```

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

## store in duckdb
ddbs_write_vector(conn, countries_sf, "countries")
ddbs_write_vector(conn, argentina_sf, "argentina")

## intersection inside the connection
ddbs_intersection("countries", "argentina", conn)

## intersection without using a connection
ddbs_intersection(countries_sf, argentina_sf)

## End(Not run)
```

`ddbs_intersects` *Spatial intersects predicate*

Description

Tests if geometries in `x` intersect geometries in `y`. Returns `TRUE` if geometries share at least one point in common.

Usage

```
ddbs_intersects(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`. Data is returned from this object.
`y`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`.
`conn`	A connection object to a DuckDB database. If `NULL`, the function runs on a temporary DuckDB database.
`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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "intersects".

Value

A list where each element contains indices (or IDs) of geometries in y that intersect the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_intersects(countries_sf, rivers_sf, id_x = "NAME_ENGL")

## End(Not run)
```

---

ddbs_intersects_extent

Spatial intersects extent predicate

Description

Tests if the bounding box of geometries in x intersect the bounding box of geometries in y. Returns TRUE if the extents (bounding boxes) overlap. This is faster than full geometry intersection but less precise.

Usage

```
ddbs_intersects_extent(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "intersects_extent".

Value

A list where each element contains indices (or IDs) of geometries in y whose bounding box intersects the bounding box of the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

```
library(duckspatial)
library(sf)

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

# Fast bounding box intersection check
ddbs_intersects_extent(countries_sf, rivers_sf, id_x = "NAME_ENGL")

## End(Not run)
```

`ddbs_is_simple` *Check if geometries are simple*

Description

Checks if geometries are simple (no self-intersections) from a DuckDB table using the spatial extension. Returns the result as an `sf` object with a boolean simplicity column or creates a new table in the database.

Usage

```
ddbs_is_simple(
  x,
  conn = NULL,
  name = NULL,
  new_column = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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
`new_column`	Name of the new column to create on the input data. If NULL, the function will return a vector with the result

crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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 vector, an sf object with simplicity information or TRUE (invisibly) for table creation

Examples

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

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

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

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

## check simplicity
ddbs_is_simple("argentina", conn)

## check simplicity without using a connection
ddbs_is_simple(argentina_sf)

## End(Not run)
```

---

ddbs_is_valid *Check if geometries are valid*

Description

Checks the validity of geometries from a DuckDB table using the spatial extension. Returns the result as an sf object with a boolean validity column or creates a new table in the database.

Usage

```
ddbs_is_valid(
  x,
  conn = NULL,
  name = NULL,
  new_column = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
new_column	Name of the new column to create on the input data. If NULL, the function will return a vector with the result
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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 vector, an sf object with validity information or TRUE (invisibly) for table creation

Examples

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

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

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

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

## check validity
ddbs_is_valid("argentina", conn)

## check validity without using a connection
ddbs_is_valid(argentina_sf)

## End(Not run)
```

---

ddbs_is_within_distance

Within Distance predicate

Description

Tests if geometries in x are within a specified distance of y. Returns TRUE if geometries are within the distance.

Usage

```
ddbs_is_within_distance(
  x,
  y,
  distance = NULL,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
distance	a numeric value specifying the distance for ST_DWithin. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.

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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "dwithin".

Value

A list where each element contains indices (or IDs) of geometries in y that touch the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

## check countries within 1 degree of distance
ddbs_is_within_distance(countries_filter_sf, countries_sf, 1)

## End(Not run)
```

---

ddbs_join *Performs spatial joins of two geometries*

Description

Performs spatial joins of two geometries, and returns a sf object or creates a new table in a DuckDB database.

Usage

```
ddbs_join(
  x,
  y,
  join = "intersects",
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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.
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 it's NULL (the default), it will return the result as an sf object.
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

# create points data
n <- 100
points_sf <- 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)



# option 1: passing sf objects
output1 <- duckspatial::ddbs_join(
    x = points_sf,
    y = countries_sf,
    join = "within"
)

plot(output1["CNTR_NAME"])


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

# creates a duckdb
conn <- duckspatial::ddbs_create_conn()
```

```
# write sf to duckdb
ddbs_write_vector(conn, points_sf, "points", overwrite = TRUE)
ddbs_write_vector(conn, countries_sf, "countries", overwrite = TRUE)

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

plot(output2["CNTR_NAME"])


## End(Not run)
```

---

ddbs_length *Calculates the length of geometries*

Description

Calculates the length of geometries from a DuckDB table or a `sf` object Returns the result as an `sf` object with a length column or creates a new table in the database. Note: Length units depend on the CRS of the input geometries (e.g., meters for projected CRS, or degrees for geographic CRS).

Usage

```
ddbs_length(
  x,
  conn = NULL,
  name = NULL,
  new_column = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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
new_column	Name of the new column to create on the input data. If NULL, the function will return a vector with the result
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

## calculate length (returns sf object with length column)
ddbs_length("rivers", conn)

## calculate length with custom column name
ddbs_length("rivers", conn, new_column = "length_meters")

## create a new table with length calculations
ddbs_length("rivers", conn, name = "rivers_with_length")

## calculate length in a sf object (without a connection)
ddbs_length(rivers_sf)

## 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 connection object to a DuckDB database

Value

data.frame

Examples

```
## TODO
2+2
```

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)
```

Arguments

conn	A connection object to a DuckDB database
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

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 exntesion
ddbs_install(conn)
ddbs_load(conn)

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

`ddbs_make_polygon` *Creates polygons from linestring geometries*

Description

Constructs polygon geometries from linestring geometries in a DuckDB table using the spatial extension. The input linestrings must be closed (first and last points must be identical). Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_make_polygon(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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

crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

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

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

## End(Not run)
```

---

ddbs_make_valid *Make invalid geometries valid*

Description

Attempts to make invalid geometries valid from a DuckDB table using the spatial extension. Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_make_valid(
  x,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	Name of the column to store CRS information. Default is "crs_duckspatial".
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

an sf object with valid geometries or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, countries_sf, "countries")
```

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

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

## End(Not run)
```

---

ddbs_overlaps *Spatial overlaps predicate*

Description

Tests if geometries in x overlap geometries in y. Returns TRUE if geometries share some but not all points, and the intersection has the same dimension as the geometries.

Usage

```
ddbs_overlaps(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "overlaps".

Value

A list where each element contains indices (or IDs) of geometries in y that overlap the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

spain_sf <- st_read(system.file("spatial/countries.geojson", package = "duckspatial")) |>
  filter(CNTR_ID %in% c("PT", "ES", "FR", "FI"))

ddbs_overlaps(countries_sf, spain_sf)

## End(Not run)
```

---

ddbs_predicate *Spatial predicate operations*

Description

Computes spatial relationships between two geometry datasets using DuckDB's spatial extension. Returns a list where each element corresponds to a row of x, containing the indices (or IDs) of rows in y that satisfy the specified spatial predicate.

Usage

```
ddbs_predicate(
  x,
  y,
  predicate = "intersects",
  conn = NULL,
  id_x = NULL,
```

```
  id_y = NULL,
  sparse = TRUE,
  distance = NULL,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
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.
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 index list. If FALSE, it returns a dense logical 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)
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

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

A **list** of length equal to the number of rows in x.

- Each element contains:
  - **integer vector** of row indices of y that satisfy the predicate with the corresponding geometry of x, or
  - **character vector** if id_y is supplied.
- The names of the list elements:
  - are integer row numbers of x, or
  - the values of id_x if provided.

If there's no match between x and y it returns NULL

Examples

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

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

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

## Store in DuckDB
ddbs_write_vector(conn, countries_sf, "countries")
ddbs_write_vector(conn, rivers_sf, "rivers")

## Example 1: Check which rivers intersect each country
ddbs_predicate(countries_sf, rivers_sf, predicate = "intersects", conn)

## Example 2: Find neighboring countries
```

```
ddbs_predicate(countries_sf, countries_sf, predicate = "touches",
               id_x = "NAME_ENGL", id_y = "NAME_ENGL")

## Example 3: Find rivers that don't intersect countries
ddbs_predicate(countries_sf, rivers_sf, predicate = "disjoint",
               id_x = "NAME_ENGL", id_y = "RIVER_NAME")

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

## End(Not run)
```

---

ddbs_quadkey *Convert geometries to QuadKey tiles*

Description

Converts POINT geometries to QuadKey tile representations at a specified zoom level. QuadKeys are a hierarchical spatial indexing system used by mapping services like Bing Maps.

Usage

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

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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.
output	Character string specifying output format. One of: • "polygon" - Returns QuadKey tile boundaries as polygons (default) • "raster" - Returns QuadKey values as a raster grid

- "tilexy" - Returns tile XY coordinates

field	Character string specifying the field name for raster output. Only used when output = "raster"
fun	summarizing function for when there are multiple geometries in one cell (e.g. "mean", "min", "max", "sum"). Only used when output = "raster"
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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.

Value

An sf object or TRUE (invisibly) for table creation

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_sf <- 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_vector` *Load spatial vector data 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_vector` and temporary Arrow views created by `ddbs_register_vector`.

Usage

```
ddbs_read_vector(
  conn,
  name,
  crs = NULL,
  crs_column = "crs_duckspatial",
  clauses = NULL,
  quiet = FALSE
)
```

Arguments

`conn`	A connection 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
`crs`	The coordinates reference system of the data. Specify if the data doesn't have a `crs_column`, and you know the CRS.
`crs_column`	a character string of length one specifying the column storing the CRS (created automatically by `ddbs_write_vector`). Set to `NULL` if absent.
`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

```
## 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_vector(conn, "points", crs = 4326)

## Example 2: Register and read Arrow view (faster, temporary)
ddbs_register_vector(conn, sf_points, "points_view")
ddbs_read_vector(conn, "points_view", crs = 4326)

## disconnect from db
ddbs_stop_conn(conn)
```

---

ddbs_register_vector *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_vector` for workflows that do not require data to be permanently materialized in the database.

Usage

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

Arguments

conn	A connection 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(duckspatial)
library(sf)

conn <- ddbs_create_conn("memory")

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

ddbs_register_vector(conn, nc, "nc_arrow_view")

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

ddbs_stop_conn(conn, shutdown = TRUE)

## End(Not run)
```

---

ddbs_rotate *Rotate geometries around centroid*

Description

Rotates geometries from from a sf object or a DuckDB table. Returns the result as an sf object or creates a new table in the database.

Usage

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

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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 (default), 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, argentina_sf, "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_sf, angle = 45)

## End(Not run)
```

---

ddbs_rotate_3d *Rotate 3D geometries around an axis*

Description

Rotates 3D geometries from from a `sf` object or a DuckDB table around the X, Y, or Z axis. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_rotate_3d(
  x,
  angle,
  units = c("degrees", "radians"),
  axis = "x",
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, countries_sf, "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_sf, angle = 45, axis = "z")

## End(Not run)
```

`ddbs_scale` *Scale geometries by X and Y factors*

Description

Scales geometries around the centroid of the geometry. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_scale(
  x,
  x_scale = 1,
  y_scale = 1,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `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` (default), 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

crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, countries_sf, "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_sf, x_scale = 1.5, y_scale = 1.5, by_feature = FALSE)

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


## End(Not run)
```

ddbs_shear *Shear geometries*

Description

Applies a shear transformation to geometries from a sf object or a DuckDB table. Returns the result as an sf object or creates a new table in the database. Shearing skews the geometry by shifting coordinates proportionally.

Usage

```
ddbs_shear(
  x,
  x_shear = 0,
  y_shear = 0,
  by_feature = FALSE,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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 (default), 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.

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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, countries_sf, "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_sf, x_shear = 0.3, y_shear = 0)

## End(Not run)
```

---

ddbs_shift *Shift geometries by X and Y offsets*

Description

Shifts (translates) geometries from a sf object or a DuckDB table. Returns the result as an sf object or creates a new table in the database. This function is equivalent to terra::shift().

Usage

```
ddbs_shift(
  x,
  dx = 0,
  dy = 0,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

## store in duckdb
ddbs_write_vector(conn, argentina_sf, "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_sf, dx = 10, dy = 5)

## End(Not run)
```

`ddbs_simplify` *Simplify geometries*

Description

Simplifies geometries from a DuckDB table using the Douglas-Peucker algorithm via the spatial extension. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_simplify(
  x,
  tolerance,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

`x`	An `sf` spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database `conn`. Data is returned from this object.
`tolerance`	Tolerance distance for simplification. Larger values result in more simplified geometries.
`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

crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object with simplified geometries or TRUE (invisibly) for table creation

Examples

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

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

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

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

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

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

## End(Not run)
```

---

ddbs_stop_conn *Close a duckdb connection*

Description

Close a duckdb connection

Usage

```
ddbs_stop_conn(conn)
```

Arguments

conn	A connection object to a DuckDB database

Value

TRUE (invisibly) for successful disconnection

Examples

```
## load packages
library(duckspatial)

## create an in-memory duckdb database
conn <- ddbs_create_conn(dbdir = "memory")

## close the connection
ddbs_stop_conn(conn)
```

---

ddbs_touches *Spatial touches predicate*

Description

Tests if geometries in x touch geometries in y. Returns TRUE if geometries share a boundary but their interiors do not intersect.

Usage

```
ddbs_touches(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.

conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "touches".

Value

A list where each element contains indices (or IDs) of geometries in y that touch the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

# Find neighboring countries
ddbs_touches(countries_filter_sf, countries_sf, id_x = "NAME_ENGL", id_y = "NAME_ENGL")

## End(Not run)
```

---

ddbs_transform *Transform coordinate reference system of geometries*

Description

Transforms geometries from a DuckDB table to a different coordinate reference system using the spatial extension. Works similarly to sf::st_transform(). Returns the result as an sf object or creates a new table in the database.

Usage

```
ddbs_transform(
  x,
  y,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database 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.
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	a character string of length one specifying the column storing the CRS (created automatically by ddbs_write_vector). Set to NULL if absent.
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

## transform to different CRS using EPSG code
ddbs_transform("argentina", "EPSG:3857", conn)

## transform to match CRS of another sf object
argentina_3857_sf <- st_transform(argentina_sf, "EPSG:3857")
ddbs_write_vector(conn, argentina_3857_sf, "argentina_3857")
ddbs_transform("argentina", argentina_3857_sf, conn)

## transform to match CRS of another DuckDB table
ddbs_transform("argentina", "argentina_3857", conn)

## transform without using a connection
ddbs_transform(argentina_sf, "EPSG:3857")

## End(Not run)
```

`ddbs_union` *Union of geometries*

Description

Computes the union of geometries from a `sf` objects or a DuckDB tables using. This is equivalent to `sf::st_union()`. The function supports three modes: (1) union all geometries from a single object into one geometry, (2) union geometries from a single object grouped by one or more columns, (3) union geometries from two different objects. Returns the result as an `sf` object or creates a new table in the database.

Usage

```
ddbs_union(
  x,
  y = NULL,
  by = NULL,
  conn = NULL,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	optional. A second table name, sf object, or DuckDB connection to compute the pairwise union between geometries in x and y. Default is NULL
by	optional. Character vector specifying one or more column names to group by when computing unions. Geometries will be unioned within each group. Default is NULL
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
crs	The coordinates reference system of the data. Specify if the data doesn't have a crs_column, and you know the CRS.
crs_column	character string specifying the name of the CRS column. Default is "crs_duckspatial"
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

an sf object or TRUE (invisibly) for table creation

Examples

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

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

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

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

# union all geometries into one
ddbs_union(conn = conn, "rivers")

# union without using a connection
ddbs_union(rivers_sf)
```

```
# union geometries grouped by a column
ddbs_union(conn = conn, "rivers", by = "RIVER_NAME")

# store result in a new table
ddbs_union(conn = conn, "rivers", name = "rivers_union")

## End(Not run)
```

---

ddbs_within *Spatial within predicate*

Description

Tests if geometries in x are within geometries in y. Returns TRUE if geometry x is completely inside geometry y.

Usage

```
ddbs_within(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "within".

Value

A list where each element contains indices (or IDs) of geometries in y that contain the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_within(rivers_sf, countries_sf, id_x = "RIVER_NAME", id_y = "NAME_ENGL")

## End(Not run)
```

---

ddbs_within_properly *Spatial within properly predicate*

Description

Tests if geometries in x are properly within geometries in y. Returns TRUE if geometry x is completely inside geometry y and does not touch its boundary.

Usage

```
ddbs_within_properly(
  x,
  y,
  conn = NULL,
  id_x = NULL,
  id_y = NULL,
  sparse = TRUE,
  quiet = FALSE
)
```

Arguments

x	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn. Data is returned from this object.
y	An sf spatial object. Alternatively, it can be a string with the name of a table with geometry column within the DuckDB database conn.
conn	A connection object to a DuckDB database. If NULL, the function runs on a temporary DuckDB database.
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 index list. If FALSE, it returns a dense logical matrix.
quiet	A logical value. If TRUE, suppresses any informational messages. Defaults to FALSE.

Details

This is a convenience wrapper around ddbs_predicate() with predicate = "within_properly".

Value

A list where each element contains indices (or IDs) of geometries in y that properly contain the corresponding geometry in x. See ddbs_predicate() for details.

See Also

ddbs_predicate() for other spatial predicates.

Examples

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

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

ddbs_within_properly(countries_sf, rivers_sf, id_x = "NAME_ENGL", id_y = "RIVER_NAME")

## End(Not run)
```

ddbs_write_vector *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_vector(
  conn,
  data,
  name,
  overwrite = FALSE,
  temp_view = FALSE,
  quiet = FALSE
)
```

Arguments

conn	A connection 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_vector 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

```
## 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_vector(conn, sf_points, "points")

## read data back into R
ddbs_read_vector(conn, "points", crs = 4326)

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

Index