Package 'duckspatial'

April 29, 2025

Type Package

Title R Interface to 'DuckDB' Database with Spatial Extension

Version 0.2.0

Description Provides an interface between R and the 'DuckDB' (see <https://duckdb.org>) database with spatial extensions. It supports reading, writing, and performing some geometric operations.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports cli, DBI, glue, sf

RoxygenNote 7.3.2

Suggests duckdb

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

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

NeedsCompilation no

Author Adrián Cidre González [aut, cre]

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

Repository CRAN

Date/Publication 2025-04-29 18:40:02 UTC

Contents

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

Arguments

`conn`	a connection object to a DuckDB database
`x`	a table with a geometry column within the DuckDB database
`distance`	a numeric value specifying the buffer distance. Units correspond to the coordinate system of the geometry (e.g. degrees or meters)
`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`	whether to overwrite the existing table if it exists. Ignored when `name` is NULL

Value

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

Examples

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

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

## 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, "argentina", 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(
  conn,
  x,
  name = NULL,
  crs = NULL,
  crs_column = "crs_duckspatial",
  overwrite = FALSE
)
```

Arguments

conn	a connection object to a DuckDB database
x	a table with a geometry column within the 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	whether to overwrite the existing table if it exists. Ignored when name is NULL

Value

an sf object or TRUE (invisibly) for table creation

Examples

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

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

## 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(conn, "argentina")

## End(Not run)
```

---

ddbs_create_schema *Check and create schema*

Description

Check and create schema

Usage

```
ddbs_create_schema(conn, name)
```

Arguments

conn	a connection object to a DuckDB database
name	a character string with the name of the schema to be created

Value

TRUE (invisibly) for successful schema creation

Examples

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

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

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

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

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

---

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)

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

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

Arguments

conn	a connection object to a DuckDB database

x	a table with geometry column within the DuckDB database. Data is returned from this object
y	a table with geometry column within the 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 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	whether to overwrite the existing table if it exists. Ignored when name is NULL

Value

an sf object or TRUE (invisibly) for table creation

Examples

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

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

## 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")

## diffrence
ddbs_difference(conn, "countries", "argentina")

## End(Not run)
```

---

ddbs_filter *Spatial Filter*

Description

Filters data spatially based on a spatial predicate

Usage

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

Arguments

conn	a connection object to a DuckDB database
x	a table with geometry column within the DuckDB database. Data is returned from this object
y	a table with geometry column within the 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
predicate	geometry predicate to use for filtering the data
crs	the coordinates reference system of the data. Specify if the data doesn't have 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	whether to overwrite the existing table if it exists. Ignored when name is NULL

Value

an sf object or TRUE (invisibly) for table creation

Examples

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

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

## 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, "countries", "argentina", predicate = "touches")

## 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")
```

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 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`)

Value

`sf` object

Examples

```
## TODO
```

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)
```

Arguments

conn	a connection object to a DuckDB database
upgrade	if TRUE, it upgrades the DuckDB extension to the latest version

Value

TRUE (invisibly) for successful installation

Examples

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

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

## install the spatial exntesion
ddbs_install(conn)

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

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

Arguments

conn	a connection object to a DuckDB database
x	a table with geometry column within the DuckDB database. Data is returned from this object
y	a table with geometry column within the 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 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	whether to overwrite the existing table if it exists. Ignored when name is NULL

Value

an sf object or TRUE (invisibly) for table creation

Examples

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

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

## 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
ddbs_intersection(conn, "countries", "argentina")

## 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
```

`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)
```

Arguments

`conn`	a connection object to a DuckDB database

Value

TRUE (invisibly) for successful installation

Examples

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

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

## install the spatial exntesion
ddbs_install(conn)
ddbs_load(conn)

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

---

ddbs_read_vector *Load vectorial data from DuckDB into R*

Description

Retrieves the data from a DuckDB table with a geometry column, and convert it to an R `sf` object.

Usage

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

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 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...")

Value

an sf object

Examples

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

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

## install the spatial exntesion
ddbs_install(conn)
ddbs_load(conn)

## 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)

## 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)
```

---

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)
```

Arguments

conn	a connection object to a DuckDB database
data	a sf object to write to the DuckDB database, or a local file
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.
overwrite	whether to overwrite the existing table if it exists

Value

TRUE (invisibly) for successful import

Examples

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

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

## install the spatial exntesion
ddbs_install(conn)
ddbs_load(conn)

## 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