

# Package: dbSpatial (via r-universe)

May 28, 2026

**Title** Spatial Data Operations for Database-Backed Geometries

**Version** 0.1.2

**Description** Provides database-backed spatial geometry classes and methods for working with vector spatial data in 'DuckDB'. The package supports loading, converting, querying, joining, and measuring spatial geometries through familiar 'sf'-style interfaces while keeping geometry columns lazy inside the database. It integrates with 'dbProject' to preserve database paths, live connections, and spatial table metadata across interactive sessions. The package follows the Simple Features framework described by Pebesma (2018) [doi:10.32614/RJ-2018-009](https://doi.org/10.32614/RJ-2018-009) and uses DuckDB's spatial extension [https://duckdb.org/docs/stable/core\\_extensions/spatial/overview.html](https://duckdb.org/docs/stable/core_extensions/spatial/overview.html).

**License** GPL-3 | MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Depends** R (>= 4.1.0)

**Imports** methods, tools, DBI, duckdb (>= 1.4.0), dplyr, dbplyr, terra, sf, e1071, lifecycle, crayon, checkmate, cli, rlang, glue, tidysselect, dbProject

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**URL** <https://github.com/dbverse-org/dbspatial-r>,  
<https://dbverse-org.github.io/dbspatial-r/>

**BugReports** <https://github.com/dbverse-org/dbspatial-r/issues>

**Config/testthat/edition** 3

**Collate** 'utils.R' 'generics.R' 'classes.R' 'accessors.R'  
'as\_dbSpatial.R' 'coerce.R' 'constructors.R' 'dbSpatial.R'  
'extract.R' 'geometry\_ops.R' 'global.R' 'imports.R'  
'input\_validation.R' 'loadSpatial.R' 'measurements.R' 'show.R'

'sim\_dbSpatial.R' 'sql\_gen.R' 'st\_as\_sf.R' 'st\_bbox.R'  
 'st\_geometry\_type.R' 'st\_geometrytype.R' 'st\_is\_valid.R'  
 'st\_spatial\_join.R' 'st\_join.R' 'st\_read.R' 'st\_translate.R'  
 'st\_xmax.R' 'st\_ymax.R' 'vect.R'

**Config/pak/sysreqs** libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libuv1-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev xz-utils

**Repository** <https://dbverse-org.r-universe.dev>

**Date/Publication** 2026-05-28 00:47:05 UTC

**RemoteUrl** <https://github.com/dbverse-org/dbspatial-r>

**RemoteRef** HEAD

**RemoteSha** 7114a51db211acdfdefcba24b8ba570643640974

## Contents

\$,dbSpatial-method . . . . .	3
as_dbSpatial . . . . .	3
dbSpatial . . . . .	4
dbSpatial_options . . . . .	6
head,dbSpatial-method . . . . .	7
loadSpatial . . . . .	7
show,dbSpatial-method . . . . .	8
st_area . . . . .	8
st_as_geojson . . . . .	9
st_as_sf.dbSpatial . . . . .	10
st_as_text . . . . .	11
st_bbox . . . . .	12
st_buffer . . . . .	12
st_centroid . . . . .	13
st_geometrytype . . . . .	14
st_join.dbSpatial . . . . .	14
st_length . . . . .	16
st_npoints . . . . .	17
st_perimeter . . . . .	18
st_simplify . . . . .	19
st_translate . . . . .	20
st_x . . . . .	21
st_xmax . . . . .	22
st_y . . . . .	23
st_ymax . . . . .	24
tail,dbSpatial-method . . . . .	25
vect,dbSpatial-method . . . . .	26

**Index**

**27**

---

\$.dbSpatial-method      *Column extraction for dbSpatial*

---

**Description**

Extract a column from a dbSpatial object

**Usage**

```
## S4 method for signature 'dbSpatial'
x$name
```

**Arguments**

x	A dbSpatial object
name	Column name to extract

**Value**

A vector of values from the specified column

---

as\_dbSpatial      *Convert an sf or terra object to a dbSpatial object*

---

**Description**

Create a [dbSpatial](#) object from an sf or terra object.

**Usage**

```
as_dbSpatial(
  rSpatial,
  conn,
  name,
  overwrite = FALSE,
  temporary = TRUE,
  ...
)
```

**Arguments**

rSpatial	sf or terra object.
conn	A DBI connection object, as returned by DBI::dbConnect().
name	a character string with the unquoted DBMS table name, e.g. "table_name"
overwrite	logical. Overwrite existing table. default = FALSE.
temporary	logical. Create a temporary table. default = TRUE.
...	Additional arguments to be passed

**Details**

Writes the `rSpatial` object to a temporary DuckDB table and computes the table in the database with the specified name and the geometry column as `geom`.

**Value**

A `dbSpatial` object backed by table name in `conn`, with the geometry column stored as `geom`.

**See Also**

Other `dbSpatial`: [dbSpatial](#), [show,dbSpatial-method](#), [st\\_as\\_sf.dbSpatial\(\)](#), [vect,dbSpatial-method](#)

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  coordinates <- data.frame(x = c(100, 200, 300), y = c(500, 600, 700))
  attributes <- data.frame(id = 1:3, name = c("A", "B", "C"))

  # Combine the coordinates and attributes
  dummy_data <- cbind(coordinates, attributes)

  # Create a SpatVector from the data.frame
  dummy_spatvector <- terra::vect(dummy_data, geom = c("x", "y"))

  # Set db connection
  duckdb_conn = DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")

  dbSpatial <- as_dbSpatial(rSpatial = dummy_spatvector,
                           conn = duckdb_conn,
                           name = "dummy_spatvector",
                           overwrite = TRUE)

  dbSpatial
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

---

`dbSpatial`

Create a `dbSpatial` object with geometry data type

---

**Description**

Constructor function to ingest diverse spatial data sources and create a `dbSpatial` object containing a geometry data type based on the [Simple Features](#) standard.

If `x_colName` and `y_colName` are both provided, a POINT geometry will be constructed based on these columns.

**Usage**

```
dbSpatial(
  value,
  name,
  conn,
  x_colName = NULL,
  y_colName = NULL,
  geomName = "geom",
  overwrite = FALSE,
  ...
)
```

**Arguments**

value	data.frame, tbl_duckdb_connection, character (valid file path), sf object, or terra object. Data to construct <code>dbSpatial</code> object with geometry data type. See details for more information.
name	Table name.
conn	A DBI connection object.
x_colName	character. Name of column containing numerical X coordinates. default = NULL.
y_colName	character. Name of column containing numerical Y coordinates. default = NULL.
geomName	character string. The geometry column name in the <code>dbSpatial</code> object. Default: "geom".
overwrite	logical. Overwrite existing table. default = FALSE.
...	Additional arguments to be passed

**Details**

For list of files supported see link below. [https://DuckDB.org/docs/extensions/spatial.html#st\\_read---read-spatial-value-from-files](https://DuckDB.org/docs/extensions/spatial.html#st_read---read-spatial-value-from-files)

**Value**

`dbSpatial` object.

**See Also**

Other `dbSpatial`: `as_dbSpatial()`, `show_dbSpatial-method`, `st_as_sf.dbSpatial()`, `vect_dbSpatial-method`

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  # create in-memory DuckDB db
  duckdb_conn = DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
}
```

```

# test value
test_data = data.frame(x = 1:10, y = 1:10, id = 1:10)

test_file <- tempfile(fileext = ".csv")
write.csv(test_data, test_file, row.names = FALSE)

# read data.frame and create point geometry
dbSpatial(conn = duckdb_conn,
           name = "test_points",
           value = test_data,
           x_colName = "x",
           y_colName = "y",
           overwrite = TRUE)

# read csv
dbSpatial(conn = duckdb_conn,
           name = "test_points",
           value = test_file,
           x_colName = "x",
           y_colName = "y",
           overwrite = TRUE)
unlink(test_file)
DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}

```

---

dbSpatial\_options      *dbSpatial Package Global Options*

---

## Description

Global options to control package behavior.

## Details

Use `options()` to set the below options.

## Options

- `dbSpatial.max_print`: integer. Max characters for WKT in show method (default 30).
- `dbSpatial.max_mem_convert`: numeric. Max bytes for implicit coercion (default 8GB).
- `dbSpatial.verbose`: logical. Print info messages during coercion (default TRUE).

## Examples

```

options(dbSpatial.max_print = 50)
options(dbSpatial.max_mem_convert = 16 * 1024^3)
options(dbSpatial.verbose = FALSE)

```

---

head,dbSpatial-method *head method for dbSpatial*

---

**Description**

head method for dbSpatial

**Usage**

```
## S4 method for signature 'dbSpatial'  
head(x, n = 6L, ...)
```

**Arguments**

x	A dbSpatial object.
n	Number of rows to return.
...	Additional arguments.

**Value**

A dbSpatial object containing the first n rows of x.

**See Also**

Other dbData: [tail,dbSpatial-method](#)

---

loadSpatial *Install and/or load DuckDB spatial extension*

---

**Description**

Install and/or load DuckDB spatial extension

**Usage**

```
loadSpatial(conn)
```

**Arguments**

conn	duckdb connection
------	-------------------

**Value**

No return value, called for side effects. Installs and loads the DuckDB spatial extension for conn.

**Examples**

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  duckdb_conn = DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  loadSpatial(conn = duckdb_conn)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}

```

---

show,dbSpatial-method *show method for dbSpatial*

---

**Description**

Show method for dbSpatial

**Usage**

```

## S4 method for signature 'dbSpatial'
show(object)

```

**Arguments**

object            A dbSpatial object.

**See Also**

Other dbSpatial: [as\\_dbSpatial\(\)](#), [dbSpatial](#), [st\\_as\\_sf.dbSpatial\(\)](#), [vect,dbSpatial-method](#)

---

st\_area            *Get area of geometries*

---

**Description**

Returns the area of the geometry column.

**Arguments**

x                    [dbSpatial](#) object  
geomName            character string. The geometry column name. Default: "geom".  
...                    additional arguments passed to methods

**Value**

[dbSpatial](#) object (lazy tibble with area column)

**See Also**

Other measurements: [st\\_length\(\)](#), [st\\_perimeter\(\)](#)

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  square <- sf::st_sf(
    id = 1,
    geom = sf::st_sfc(
      sf::st_polygon(list(rbind(
        c(0, 0), c(1, 0), c(1, 1), c(0, 1), c(0, 0)
      )))
    )
  )
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(square, conn = duckdb_conn, name = "square",
    overwrite = TRUE)
  st_area(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

---

 st\_as\_geojson

*Convert to GeoJSON*


---

**Description**

Returns the GeoJSON representation of the geometry.

**Usage**

```
st_as_geojson(x, ...)

## S3 method for class 'dbSpatial'
st_as_geojson(x, ...)
```

**Arguments**

x                    [dbSpatial](#) object  
 ...                  additional arguments

**Value**

[dbSpatial](#) object with GeoJSON column

**Functions**

- `st_as_geojson(dbSpatial)`: Method for `dbSpatial` objects

**See Also**

Other constructors: [st\\_as\\_text\(\)](#)

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  point <- sf::st_sf(id = 1, geom = sf::st_sfc(sf::st_point(c(1, 2))))
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(point, conn = duckdb_conn, name = "point",
                   overwrite = TRUE)
  st_as_geojson(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

---

st\_as\_sf.dbSpatial      *Convert dbSpatial objects to sf objects*

---

**Description**

S3 method implementation for converting dbSpatial objects to sf objects.

**Usage**

```
## S3 method for class 'dbSpatial'
st_as_sf(x, geomName = "geom", select = tidyselect::everything(), ...)
```

**Arguments**

x	A <a href="#">dbSpatial</a> object to convert
geomName	character string. The geometry column name in the <a href="#">dbSpatial</a> object. Default: "geom".
select	Columns to retain in output (default: all columns)
...	Additional arguments passed to <a href="#">sf::st_read</a>

**Details**

This method handles conversion of [dbSpatial](#) to [sf::sf](#) objects using:

- Dynamic column selection via tidyselect semantics
- Automatic geometry column preservation
- SQL-level column subsetting for efficiency

**Value**

An [sf::sf](#) object

**See Also**

Other dbSpatial: [as\\_dbSpatial\(\)](#), [dbSpatial](#), [show,dbSpatial-method](#), [vect,dbSpatial-method](#)

---

st\_as\_text

*Convert to WKT*

---

**Description**

Returns the Well-Known Text (WKT) representation of the geometry.

**Arguments**

x                    [dbSpatial](#) object  
...                    additional arguments

**Value**

[dbSpatial](#) object with WKT column

**See Also**

Other constructors: [st\\_as\\_geojson\(\)](#)

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {  
  point <- sf::st_sf(id = 1, geom = sf::st_sfc(sf::st_point(c(1, 2))))  
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")  
  DBI::dbExecute(duckdb_conn, "SET threads = 1")  
  x <- as_dbSpatial(point, conn = duckdb_conn, name = "point",  
                    overwrite = TRUE)  
  st_as_text(x)  
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)  
}
```

---

st_bbox	<i>Compute bounding box for dbSpatial object</i>
---------	--

---

**Description**

Returns the bounding box of the geometry column.

**Usage**

```
## S3 method for class 'dbSpatial'  
st_bbox(obj, ...)
```

**Arguments**

obj	<a href="#">dbSpatial</a> object
...	additional arguments passed to methods

**Value**

bbox object (named numeric vector with xmin, ymin, xmax, ymax)

**See Also**

Other geom\_summary: [st\\_xmax\(\)](#), [st\\_ymax\(\)](#)

---

st_buffer	<i>Compute buffer around geometry</i>
-----------	---------------------------------------

---

**Description**

Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance.

**Arguments**

x	<a href="#">dbSpatial</a> object
dist	numeric distance
...	additional arguments

**Value**

[dbSpatial](#) object with buffered geometry

**See Also**

Other geometry\_ops: [st\\_centroid\(\)](#), [st\\_simplify\(\)](#)

**Examples**

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  point <- sf::st_sf(id = 1, geom = sf::st_sfc(sf::st_point(c(0, 0))))
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(point, conn = duckdb_conn, name = "point",
                   overwrite = TRUE)
  st_buffer(x, dist = 1)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}

```

---

st\_centroid

*Compute centroid of geometry*


---

**Description**

Returns the geometric center of a geometry.

**Arguments**

x                    [dbSpatial](#) object  
...                    additional arguments

**Value**

[dbSpatial](#) object with centroid geometry

**See Also**

Other geometry\_ops: [st\\_buffer\(\)](#), [st\\_simplify\(\)](#)

**Examples**

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  square <- sf::st_sf(
    id = 1,
    geom = sf::st_sfc(
      sf::st_polygon(list(rbind(
        c(0, 0), c(1, 0), c(1, 1), c(0, 1), c(0, 0)
      )))
    )
  )
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(square, conn = duckdb_conn, name = "square",
                   overwrite = TRUE)
  st_centroid(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}

```

}

---

st_geometrytype	<i>Get geometry types (DuckDB-native)</i>
-----------------	---

---

**Description**

Returns the geometry type for each row in a [dbSpatial](#) object using DuckDB's spatial function `ST_GeometryType()`.

**Usage**

```
st_geometrytype(dbSpatial, geomName = "geom", collect = FALSE, n = NULL, ...)
```

**Arguments**

dbSpatial	A <a href="#">dbSpatial</a> object.
geomName	Geometry column name. Default: "geom".
collect	Logical (default = FALSE). If TRUE, collect results into a character vector.
n	Optional integer. If provided alongside <code>collect = TRUE</code> , limits the query to the first n rows before collecting.
...	Additional arguments (ignored).

**Value**

If `collect = FALSE` (default), a lazy tibble with a single column `geom_type`. If `collect = TRUE`, a character vector of geometry type(s).

---

st_join.dbSpatial	<i>Spatial join for dbSpatial objects</i>
-------------------	---

---

**Description**

Performs a spatial join between two `dbSpatial` objects using a specified spatial predicate function.

**Usage**

```
## S3 method for class 'dbSpatial'
st_join(x, y, join = st_intersects, suffix = c(".x", ".y"), ...)
```

**Arguments**

x	A <code>dbSpatial</code> object.
y	A <code>dbSpatial</code> object.
join	Spatial predicate function to use for the join. Default is <code>st_intersects</code> . Supported predicates: <code>st_intersects</code> , <code>st_contains</code> , <code>st_within</code> , <code>st_covers</code> , <code>st_covered_by</code> , <code>st_crosses</code> , <code>st_disjoint</code> , <code>st_equals</code> , <code>st_touches</code> , <code>st_overlaps</code> .
suffix	Character vector of length 2. Suffixes to add to duplicate column names from x and y. Default is <code>c(".x", ".y")</code> .
...	Additional arguments passed to the internal join function.

**Details**

This function follows the `sf` pattern where spatial predicates are passed as the `join` argument.

**Value**

A `dbSpatial` object containing the spatial join result.

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  con <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(con, "SET threads = 1")

  df1 <- data.frame(id = 1:3, x = c(0, 10, 20), y = c(0, 10, 20))
  pts1 <- dbSpatial(conn = con, name = "pts1", value = df1,
    x_colName = "x", y_colName = "y", overwrite = TRUE)

  df2 <- data.frame(id = 4:6, x = c(0, 15, 25), y = c(0, 15, 25))
  pts2 <- dbSpatial(conn = con, name = "pts2", value = df2,
    x_colName = "x", y_colName = "y", overwrite = TRUE)

  # Spatial join using intersection
  result <- sf::st_join(pts1, pts2, join = sf::st_intersects)

  # Spatial join using within predicate
  result <- sf::st_join(pts1, pts2, join = sf::st_within)

  DBI::dbDisconnect(con, shutdown = TRUE)
}
```

---

st_length	<i>Get length of geometries</i>
-----------	---------------------------------

---

### Description

Returns the length of the geometry column.

### Arguments

`x` [dbSpatial](#) object

`geomName` character string. The geometry column name. Default: "geom".

... additional arguments passed to methods

### Value

[dbSpatial](#) object (lazy tibble with length column)

### See Also

Other measurements: [st\\_area\(\)](#), [st\\_perimeter\(\)](#)

### Examples

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  line <- sf::st_sf(
    id = 1,
    geom = sf::st_sfc(
      sf::st_linestring(rbind(c(0, 0), c(1, 1), c(2, 1)))
    )
  )
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(line, conn = duckdb_conn, name = "line",
    overwrite = TRUE)
  st_length(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

---

st_npoints	<i>Get number of points</i>
------------	-----------------------------

---

## Description

Returns the number of points in a geometry.

## Usage

```
st_npoints(x, ...)  
  
## S4 method for signature 'dbSpatial'  
st_npoints(x, ...)
```

## Arguments

x	<a href="#">dbSpatial</a> object
...	additional arguments

## Value

[dbSpatial](#) object with npoints column

## Methods (by class)

- `st_npoints(dbSpatial)`: Method for `dbSpatial` objects

## See Also

Other accessors: [st\\_x\(\)](#), [st\\_y\(\)](#)

## Examples

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {  
  line <- sf::st_sf(  
    id = 1,  
    geom = sf::st_sfc(sf::st_linestring(rbind(c(0, 0), c(1, 1), c(2, 1))))  
  )  
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")  
  DBI::dbExecute(duckdb_conn, "SET threads = 1")  
  x <- as_dbSpatial(line, conn = duckdb_conn, name = "line",  
                    overwrite = TRUE)  
  st_npoints(x)  
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)  
}
```

---

st_perimeter	<i>Get perimeter of geometries</i>
--------------	------------------------------------

---

### Description

Returns the perimeter of the geometry column.

### Arguments

`x` [dbSpatial](#) object

`geomName` character string. The geometry column name. Default: "geom".

... additional arguments passed to methods

### Value

[dbSpatial](#) object (lazy tibble with perimeter column)

### See Also

Other measurements: [st\\_area\(\)](#), [st\\_length\(\)](#)

### Examples

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  square <- sf::st_sf(
    id = 1,
    geom = sf::st_sfc(
      sf::st_polygon(list(rbind(
        c(0, 0), c(1, 0), c(1, 1), c(0, 1), c(0, 0)
      )))
    )
  )
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(square, conn = duckdb_conn, name = "square",
    overwrite = TRUE)
  st_perimeter(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

---

st_simplify	<i>Simplify geometry</i>
-------------	--------------------------

---

**Description**

Returns a simplified version of the given geometry using the Douglas-Peucker algorithm.

**Arguments**

x	<a href="#">dbSpatial</a> object
dTolerance	numeric tolerance
...	additional arguments

**Value**

[dbSpatial](#) object with simplified geometry

**See Also**

Other geometry\_ops: [st\\_buffer\(\)](#), [st\\_centroid\(\)](#)

**Examples**

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  line <- sf::st_sf(
    id = 1,
    geom = sf::st_sfc(
      sf::st_linestring(rbind(
        c(0, 0), c(0.5, 0.2), c(1, 0), c(1.5, 0.1), c(2, 0)
      ))
    )
  )
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(line, conn = duckdb_conn, name = "line",
    overwrite = TRUE)
  st_simplify(x, dTolerance = 0.1)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

---

st_translate	<i>Translate x, y coordinates by delta x, delta y for point geometries</i>
--------------	--

---

### Description

This function translates point geometries by the specified delta x and delta y values.

### Usage

```
st_translate(dbSpatial, geomName = "geom", dx, dy, ...)

## S4 method for signature 'dbSpatial'
st_translate(dbSpatial, geomName = "geom", dx, dy, ...)
```

### Arguments

dbSpatial	<a href="#">dbSpatial</a> object.
geomName	character string. The geometry column name in the <a href="#">dbSpatial</a> object. Default: "geom".
dx	numeric.value to shift x coordinates by
dy	numeric. value to shift y coordinates by
...	additional arguments passed to methods

### Value

[dbSpatial](#) object

### Functions

- `st_translate(dbSpatial)`: Method for `dbSpatial` object

### Examples

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  con = DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(con, "SET threads = 1")

  coordinates <- data.frame(x = c(100, 200, 300), y = c(500, 600, 700))
  attributes <- data.frame(id = 1:3, name = c("A", "B", "C"))

  # Combine the coordinates and attributes
  dummy_data <- cbind(coordinates, attributes)

  points <- dbSpatial(conn = con,
                     name = "points",
                     value = dummy_data,
                     overwrite = TRUE,
                     x_colName = "x",
```

```

        y_colName = "y")

  points

  points_translated <- st_translate(dbSpatial = points, dx = 100, dy = -20)

  points_translated
  DBI::dbDisconnect(con, shutdown = TRUE)
}

```

---

st\_x

*Get X coordinate*


---

### Description

Returns the X coordinate of a point.

### Usage

```

st_x(x, ...)

## S4 method for signature 'dbSpatial'
st_x(x, ...)

```

### Arguments

x                    [dbSpatial](#) object  
...                    additional arguments

### Value

[dbSpatial](#) object with X coordinate column

### Methods (by class)

- `st_x(dbSpatial)`: Method for `dbSpatial` objects

### See Also

Other accessors: [st\\_npoints\(\)](#), [st\\_y\(\)](#)

### Examples

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  point <- sf::st_sf(id = 1, geom = sf::st_sfc(sf::st_point(c(1, 2))))
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(point, conn = duckdb_conn, name = "point",
                    overwrite = TRUE)
}

```

```

  st_x(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}

```

---

st\_xmax

*Get maximum x coordinate*


---

### Description

This function returns the maximum x coordinate in each geometry in the specified `dbSpatial` object.

### Usage

```

st_xmax(dbSpatial, geomName = "geom", ...)

## S4 method for signature 'dbSpatial'
st_xmax(dbSpatial, geomName = "geom", ...)

```

### Arguments

<code>dbSpatial</code>	<code>dbSpatial</code> object.
<code>geomName</code>	character string. The geometry column name in the <code>dbSpatial</code> object. Default: "geom".
<code>...</code>	additional arguments passed to methods

### Value

numerical column vector in database

### Functions

- `st_xmax(dbSpatial)`: Method for `dbSpatial` object

### See Also

Other `geom_summary`: `st_bbox()`, `st_ymax()`

### Examples

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  # Create a data.frame with x and y coordinates and attributes
  coordinates <- data.frame(x = c(100, 200, 300), y = c(500, 600, 700))
  attributes <- data.frame(id = 1:3, name = c("A", "B", "C"))

  # Combine the coordinates and attributes
  dummy_data <- cbind(coordinates, attributes)
}

```

```

# Create a duckdb connection
con = DBI::dbConnect(duckdb::duckdb(), ":memory:")
DBI::dbExecute(con, "SET threads = 1")

# Create a duckdb table with spatial points
db_points = dbSpatial(conn = con,
                      value = dummy_data,
                      x_colName = "x",
                      y_colName = "y",
                      name = "foo",
                      overwrite = TRUE)

st_xmax(dbSpatial = db_points)
DBI::dbDisconnect(con, shutdown = TRUE)
}

```

---

st\_y

*Get Y coordinate*


---

## Description

Returns the Y coordinate of a point.

## Usage

```
st_y(x, ...)
```

```
## S4 method for signature 'dbSpatial'
```

```
st_y(x, ...)
```

## Arguments

x	<a href="#">dbSpatial</a> object
...	additional arguments

## Value

[dbSpatial](#) object with Y coordinate column

## Methods (by class)

- `st_y(dbSpatial)`: Method for `dbSpatial` objects

## See Also

Other accessors: [st\\_npoints\(\)](#), [st\\_x\(\)](#)

**Examples**

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  point <- sf::st_sf(id = 1, geom = sf::st_sfc(sf::st_point(c(1, 2))))
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  x <- as_dbSpatial(point, conn = duckdb_conn, name = "point",
                    overwrite = TRUE)

  st_y(x)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}

```

---

`st_ymax`*Get maximum y coordinate*

---

**Description**

This function returns the maximum y coordinate of the geometries in the specified `dbSpatial` object.

**Usage**

```

st_ymax(dbSpatial, geomName = "geom", ...)

## S4 method for signature 'dbSpatial'
st_ymax(dbSpatial, geomName = "geom", ...)

```

**Arguments**

`dbSpatial`      `dbSpatial` object.

`geomName`      character string. The geometry column name in the `dbSpatial` object. Default: "geom".

`...`            additional arguments passed to methods

**Value**

numerical column vector in database

**Functions**

- `st_ymax(dbSpatial)`: Method for `dbSpatial` object

**See Also**

Other `geom_summary`: `st_bbox()`, `st_xmax()`

**Examples**

```

if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  # Create a data.frame with x and y coordinates and attributes
  coordinates <- data.frame(x = c(100, 200, 300), y = c(500, 600, 700))
  attributes <- data.frame(id = 1:3, name = c("A", "B", "C"))

  # Combine the coordinates and attributes
  dummy_data <- cbind(coordinates, attributes)

  # Create a duckdb connection
  con = DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(con, "SET threads = 1")

  # Create a duckdb table with spatial points
  db_points = dbSpatial(conn = con,
                        value = dummy_data,
                        x_colName = "x",
                        y_colName = "y",
                        name = "foo",
                        overwrite = TRUE)

  st_ymax(dbSpatial = db_points)
  DBI::dbDisconnect(con, shutdown = TRUE)
}

```

---

tail,dbSpatial-method *tail method for dbSpatial*

---

**Description**

tail method for dbSpatial

**Usage**

```

## S4 method for signature 'dbSpatial'
tail(x, n = 6L, ...)

```

**Arguments**

x	A dbSpatial object.
n	Number of rows to return.
...	Additional arguments.

**Value**

A dbSpatial object containing the last n rows of x.

**See Also**

Other dbData: [head,dbSpatial-method](#)

---

vect,dbSpatial-method *Create SpatVector objects*

---

## Description

Create SpatVector objects

## Usage

```
## S4 method for signature 'dbSpatial'
vect(x, select = tidyselect::everything(), ...)
```

## Arguments

x	A <a href="#">dbSpatial</a> object to convert into a <a href="#">terra::SpatVector</a> object
select	Columns to retain in output (default: all columns)
...	Additional arguments passed to <a href="#">sf::st_as_sf</a>

## Value

A [terra::SpatVector](#) containing the selected columns and geometry materialized from x.

## See Also

Other dbSpatial: [as\\_dbSpatial\(\)](#), [dbSpatial](#), [show,dbSpatial-method](#), [st\\_as\\_sf.dbSpatial\(\)](#)

## Examples

```
if (interactive() && requireNamespace("duckdb", quietly = TRUE)) {
  point_data <- data.frame(x = c(100, 200), y = c(500, 600), id = 1:2)
  point_vect <- terra::vect(point_data, geom = c("x", "y"))
  duckdb_conn <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
  DBI::dbExecute(duckdb_conn, "SET threads = 1")
  dbs <- as_dbSpatial(point_vect, conn = duckdb_conn, name = "point_vect",
                     overwrite = TRUE)

  terra::vect(dbs)
  DBI::dbDisconnect(duckdb_conn, shutdown = TRUE)
}
```

# Index

- \* **accessors**
  - st\_npoints, 17
  - st\_x, 21
  - st\_y, 23
- \* **constructors**
  - st\_as\_geojson, 9
  - st\_as\_text, 11
- \* **dbData**
  - head, dbSpatial-method, 7
  - tail, dbSpatial-method, 25
- \* **dbSpatial**
  - as\_dbSpatial, 3
  - dbSpatial, 4
  - show, dbSpatial-method, 8
  - st\_as\_sf.dbSpatial, 10
  - vect, dbSpatial-method, 26
- \* **duckdb-ext**
  - loadSpatial, 7
- \* **geom\_construction**
  - st\_translate, 20
- \* **geom\_scalar**
  - st\_geometrytype, 14
- \* **geom\_summary**
  - st\_bbox, 12
  - st\_xmax, 22
  - st\_ymax, 24
- \* **geometry\_ops**
  - st\_buffer, 12
  - st\_centroid, 13
  - st\_simplify, 19
- \* **measurements**
  - st\_area, 8
  - st\_length, 16
  - st\_perimeter, 18
- \* **options**
  - dbSpatial\_options, 6
- \$, dbSpatial-method, 3
- as\_dbSpatial, 3, 5, 8, 11, 26
- dbSpatial, 3, 4, 4, 5, 8–24, 26
- dbSpatial-class (dbSpatial), 4
- dbSpatial-options (dbSpatial\_options), 6
- dbSpatial\_options, 6
- head, dbSpatial-method, 7
- loadSpatial, 7
- sf::sf, 10
- sf::st\_as\_sf, 26
- sf::st\_read, 10
- show, dbSpatial-method, 8
- st\_area, 8, 16, 18
- st\_as\_geojson, 9, 11
- st\_as\_sf.dbSpatial, 4, 5, 8, 10, 26
- st\_as\_text, 10, 11
- st\_bbox, 12, 22, 24
- st\_buffer, 12, 13, 19
- st\_centroid, 12, 13, 19
- st\_geometrytype, 14
- st\_join.dbSpatial, 14
- st\_length, 9, 16, 18
- st\_npoints, 17, 21, 23
- st\_npoints, dbSpatial-method
  - (st\_npoints), 17
- st\_perimeter, 9, 16, 18
- st\_simplify, 12, 13, 19
- st\_translate, 20
- st\_translate, dbSpatial-method
  - (st\_translate), 20
- st\_x, 17, 21, 23
- st\_x, dbSpatial-method (st\_x), 21
- st\_xmax, 12, 22, 24
- st\_xmax, dbSpatial-method (st\_xmax), 22
- st\_y, 17, 21, 23
- st\_y, dbSpatial-method (st\_y), 23
- st\_ymax, 12, 22, 24
- st\_ymax, dbSpatial-method (st\_ymax), 24
- tail, dbSpatial-method, 25

terra::SpatVector, [26](#)

vect, dbSpatial-method, [26](#)