Package: sedonadb (via r-universe)

June 7, 2026

Title Bindings for Apache SedonaDB

Version 0.3.0.9000

Description Provides bindings for Apache SedonaDB, a lightweight query engine optimized for spatial workflows.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

SystemRequirements Cargo (Rust's package manager), rustc

Depends R (>= 4.1.0)

Suggests adbcdrivermanager, dplyr, lintr, rlang, tibble, sf, testthat (>= 3.0.0), tidyselect, withr, wk

Imports geoarrow, nanoarrow

Config/testthat/edition 3

Config/build/bootstrap TRUE

Config/Needs/check lintr

Config/pak/sysreqs libzstd-dev libclang-dev

Repository https://apache.r-universe.dev

Date/Publication 2026-06-07 05:21:13 UTC

RemoteUrl https://github.com/apache/sedona-db

RemoteRef HEAD

RemoteSha 15b128bff667725748b2f23d907112ca57ad756b

RemoteSubdir r/sedonadb

Contents

.fns 2
.tables 3
as_sedonadb_dataframe 3

as_sedonadb_literal . . . 4
sd_arrange . . . 4
sd_compute . . . 5
sd_configure_proj . . . 6
sd_connect . . . 7
sd_count . . . 8
sd_drop_view . . . 8
sd_expr_column . . . 9
sd_filter . . . 11
sd_group_by . . . 11
sd_join . . . 12
sd_join_by . . . 14
sd_join_select . . . 15
sd_join_select_default . . . 16
sd_preview . . . 17
sd_read_parquet . . . 17
sd_read_sf . . . 18
sd_register_udf . . . 19
sd_select . . . 20
sd_sql . . . 20
sd_summarise . . . 21
sd_to_view . . . 22
sd_transmute . . . 22
sd_with_params . . . 23
sd_write_parquet . . . 24
sedonadb_adbc . . . 25

Index **27**

`.fns` *SedonaDB Functions*

Description

This object is an escape hatch for calling SedonaDB/DataFusion functions directly for translations that are not yet registered or are otherwise misbehaving.

Usage

```
.fns
```

Format

An object of class `sedonadb_fns` of length 0.

.tables *SedonaDB Table Qualifiers*

Description

This object is an escape hatch for referring to the right or left side of a join when constructing `sd_join_by()` or `sd_join_select()` expressions.

Usage

```
.tables
```

Format

An object of class `sedonadb_tables` of length 2.

as_sedonadb_dataframe *Convert an object to a DataFrame*

Description

Convert an object to a DataFrame

Usage

```
as_sedonadb_dataframe(x, ..., schema = NULL, ctx = NULL)
```

Arguments

`x`	An object to convert
`...`	Extra arguments passed to/from methods
`schema`	The requested schema
`ctx`	A SedonaDB context. This should always be passed to inner calls to SedonaDB functions; NULL implies the global context.

Value

A sedonadb_dataframe

Examples

```
as_sedonadb_dataframe(data.frame(x = 1:3))
```

as_sedonadb_literal *S3 Generic to create a SedonaDB literal expression*

Description

This generic provides the opportunity for objects to register a mechanism to be understood as literals in the context of a SedonaDB expression. Users constructing expressions directly should use `sd_expr_literal()`.

Usage

```
as_sedonadb_literal(x, ..., type = NULL, factory = NULL)
```

Arguments

`x`	An object to convert to a SedonaDB literal
`...`	Passed to/from methods
`type`	An optional data type to request for the output
`factory`	An `sd_expr_factory()` that should be passed to any other calls to `as_sedonadb_literal()` if needed

Value

An object of class SedonaDBExpr

Examples

```
as_sedonadb_literal("abcd")
```

sd_arrange *Order rows of a SedonaDB data frame using column values*

Description

Order rows of a SedonaDB data frame using column values

Usage

```
sd_arrange(.data, ...)
```

Arguments

`.data`	A sedonadb_dataframe or an object that can be coerced to one.
`...`	Unnamed expressions for arrange expressions. These are evaluated in the same way as `dplyr::arrange()` except does not support extra dplyr features such as `across()`, `.by_group`, or `.locale`.

Value

An object of class sedonadb_dataframe

Examples

```
data.frame(x = c(10:1, NA)) |> sd_arrange(x)
data.frame(x = c(1:10, NA)) |> sd_arrange(desc(x))
```

---

sd_compute *Collect a DataFrame into memory*

Description

Use `sd_compute()` to collect and return the result as a DataFrame; use `sd_collect()` to collect and return the result as an R data.frame.

Usage

```
sd_compute(.data)

sd_collect(.data, ptype = NULL)
```

Arguments

`.data`	A sedonadb_dataframe or an object that can be coerced to one.
`ptype`	The target R object. See nanoarrow::convert_array_stream.

Value

`sd_compute()` returns a sedonadb_dataframe; `sd_collect()` returns a data.frame (or subclass according to `ptype`).

Examples

```
sd_sql("SELECT 1 as one") |> sd_compute()
sd_sql("SELECT 1 as one") |> sd_collect()
```

`sd_configure_proj` *Configure PROJ*

Description

Performs a runtime configuration of PROJ, which can be used in place of a build-time linked version of PROJ or to add in support if PROJ was not linked at build time.

Usage

```
sd_configure_proj(
  preset = NULL,
  shared_library = NULL,
  database_path = NULL,
  search_path = NULL
)
```

Arguments

`preset`	One of: • `"homebrew"`: Look for PROJ installed by Homebrew. This is the easiest option on MacOS. • `"system"`: Look for PROJ in the platform library load path (e.g., after installing system proj on Linux). • `"auto"`: Try all presets in the order listed above, issuing a warning if none can be configured.
`shared_library`	An absolute or relative path to a shared library valid for the platform.
`database_path`	A path to proj.db
`search_path`	A path to the data files required by PROJ for some transforms.

Value

NULL, invisibly

Examples

```
sd_configure_proj("auto")
```

sd_connect *Create a SedonaDB context*

Description

Runtime options configure the execution environment. Use `global = TRUE` to configure the global context or use the returned object as a scoped context. A scoped context is recommended for programmatic usage as it prevents named views from interfering with each other.

Usage

```
sd_connect(
  ...,
  global = FALSE,
  memory_limit = NULL,
  temp_dir = NULL,
  memory_pool_type = NULL,
  unspillable_reserve_ratio = NULL
)
```

Arguments

`...`	Reserved for future options
`global`	Use TRUE to set options on the global context.
`memory_limit`	Maximum memory for query execution, as a human-readable string (e.g., `"4gb"`, `"512m"`) or `NULL` for unbounded (the default).
`temp_dir`	Directory for temporary/spill files, or `NULL` to use the DataFusion default.
`memory_pool_type`	Memory pool type: `"greedy"` (default) or `"fair"`. Only takes effect when `memory_limit` is set.
`unspillable_reserve_ratio`	Fraction of memory (0–1) reserved for unspillable consumers. Only applies when `memory_pool_type` is `"fair"`. Defaults to 0.2 when not explicitly set.

Value

The constructed context, invisibly.

Examples

```
sd_connect(memory_limit = "100mb", memory_pool_type = "fair")
```

sd_count *Count rows in a DataFrame*

Description

Count rows in a DataFrame

Usage

```
sd_count(.data)
```

Arguments

.data	A sedonadb_dataframe or an object that can be coerced to one.

Value

The number of rows after executing the query

Examples

```
sd_sql("SELECT 1 as one") |> sd_count()
```

sd_drop_view *Create or Drop a named view*

Description

Remove a view created with sd_to_view() from the context.

Usage

```
sd_drop_view(table_ref)

sd_ctx_drop_view(ctx, table_ref)

sd_view(table_ref)

sd_ctx_view(ctx, table_ref)
```

Arguments

table_ref	The name of the view reference
ctx	A SedonaDB context.

Value

The context, invisibly

Examples

```
sd_sql("SELECT 1 as one") |> sd_to_view("foofy")
sd_view("foofy")
sd_drop_view("foofy")
try(sd_view("foofy"))
```

---

sd_expr_column *Create SedonaDB logical expressions*

Description

Create SedonaDB logical expressions

Usage

```
sd_expr_column(column_name, qualifier = NULL, factory = sd_expr_factory())

sd_expr_literal(x, type = NULL, factory = sd_expr_factory())

sd_expr_binary(op, lhs, rhs, factory = sd_expr_factory())

sd_expr_negative(expr, factory = sd_expr_factory())

sd_expr_any_function(
  function_name,
  args,
  ...,
  na.rm = NULL,
  factory = sd_expr_factory()
)

sd_expr_scalar_function(function_name, args, factory = sd_expr_factory())

sd_expr_aggregate_function(
  function_name,
  args,
  ...,
  na.rm = FALSE,
  distinct = FALSE,
  factory = sd_expr_factory()
)
```

```
sd_expr_cast(expr, type, factory = sd_expr_factory())

sd_expr_alias(expr, alias, factory = sd_expr_factory())

sd_expr_parse_binary(expr)

as_sd_expr(x, factory = sd_expr_factory())

is_sd_expr(x)

sd_expr_factory(ctx = NULL)
```

Arguments

column_name	A column name
qualifier	An optional qualifier (e.g., table reference) that may be used to disambiguate a specific reference
factory	A sd_expr_factory(). This factory wraps a SedonaDB context and is used to resolve scalar functions and/or retrieve options.
x	An object to convert to a SedonaDB literal (constant).
type	A destination type into which expr should be cast.
op	Operator name for a binary expression. In general these follow R function names (e.g., >, <, +, -).
lhs, rhs	Arguments to a binary expression
expr	A SedonaDBExpr or object coercible to one with as_sd_expr().
function_name	The name of the function to call. This name is resolved from the context associated with factory.
args	A list of SedonaDBExpr or object coercible to one with as_sd_expr().
...	Reserved for future use
na.rm	For aggregate expressions, should nulls be ignored? The R idiom is to respect null; however, the SQL idiom is to drop them. The default value follows the R idiom (na.rm = FALSE).
distinct	For aggregate expressions, use only distinct values.
alias	An alias to apply to expr.
ctx	A SedonaDB context or NULL to use the default context.

Value

An object of class SedonaDBExpr

Examples

```
sd_expr_column("foofy")
sd_expr_literal(1L)
sd_expr_scalar_function("abs", list(1L))
```

```
sd_expr_cast(1L, nanoarrow::na_int64())
sd_expr_alias(1L, "foofy")
```

`sd_filter` *Keep rows of a SedonaDB DataFrame that match a condition*

Description

Keep rows of a SedonaDB DataFrame that match a condition

Usage

```
sd_filter(.data, ...)
```

Arguments

`.data`	A sedonadb_dataframe or an object that can be coerced to one.
`...`	Unnamed expressions for filter conditions. These are evaluated in the same way as `dplyr::filter()` except does not support extra dplyr features such as `across()` or `.by`.

Value

An object of class sedonadb_dataframe

Examples

```
data.frame(x = 1:10) |> sd_filter(x > 5)
```

`sd_group_by` *Group SedonaDB DataFrames by one or more expressions*

Description

Note that unlike `dplyr::group_by()`, these groups are dropped after any transformations.

Usage

```
sd_group_by(.data, ...)

sd_ungroup(.data)
```

Arguments

.data	A sedonadb_dataframe or an object that can be coerced to one.
...	Named expressions whose unique combination will be used as groups to potentially compute a future aggregate expression. These are evaluated in the same way as dplyr::group_by() except .add nor .drop are supported.

Value

An object of class sedonadb_dataframe

Examples

```
data.frame(letter = c(rep("a", 3), rep("b", 4), rep("c", 3)), x = 1:10) |>
  sd_group_by(letter) |>
  sd_summarise(x = sum(x))
```

---

sd_join *Join two SedonaDB DataFrames*

Description

Perform a join operation between two dataframes. Use sd_join_by() to specify join conditions using x$column and y$column syntax to reference columns from the left and right tables respectively.

Usage

```
sd_join(
  x,
  y,
  by = NULL,
  join_type = "inner",
  select = sd_join_select_default(),
  keep = NULL
)

sd_left_join(x, y, by = NULL, select = sd_join_select_default(), keep = NULL)

sd_right_join(x, y, by = NULL, select = sd_join_select_default(), keep = NULL)

sd_inner_join(x, y, by = NULL, select = sd_join_select_default(), keep = NULL)

sd_full_join(x, y, by = NULL, select = sd_join_select_default(), keep = NULL)

sd_semi_join(x, y, by = NULL)
```

```
sd_anti_join(x, y, by = NULL)

sd_cross_join(x, y, select = sd_join_select_default())
```

Arguments

x	The left dataframe
y	The right dataframe
by	Join specification. One of: • A sedonadb_join_by object from sd_join_by() • A sedonadb_join_by object from sd_join_intersects(), sd_join_contains(), sd_join_within(), sd_join_covers(), sd_join_coveredby(), sd_join_touches(), sd_join_crosses(), sd_join_overlaps(), or sd_join_equals(). • A character vector of column names to join on in both tables • A named character vector mapping left-table column names to right-table column names, e.g. c(x_val = "y_val") • NULL for a natural join on columns with matching names
join_type	The type of join to perform. One of "inner", "left", "right", "full", "leftsemi", "rightsemi", "leftanti", "rightanti", "leftmark", or "rightmark".
select	Post-join column selection. One of • NULL for no modification, which may result in duplicate (unqualified) column names. The column may still be referred to with a qualifier in advanced usage using sd_expr_column(). • sd_join_select_default() for dplyr-like behaviour (equi-join keys or spatial join keys removed for inner/left joins, intersecting names suffixed) • sd_join_select() for a custom selection
keep	Use TRUE to keep all key columns in an equijoin or spatial join. This is only applied when using sd_join_select_default() for left/inner joins and right equijoins (full joins and right spatial joins always keep spatial join keys).

Value

An object of class sedonadb_dataframe

Examples

```
df1 <- data.frame(x = letters[1:10], y = 1:10)
df2 <- data.frame(y = 10:1, z = LETTERS[1:10])
df1 |> sd_join(df2)
```

sd_join_by *Specify join conditions*

Description

Use sd_join_by() to specify join conditions for sd_join() using expressions that reference columns from both tables. Table references are specified using x$column and y$column syntax to disambiguate columns from the left and right tables, and the special helper x$geom() and y$geom() may be used for tables with exactly one geometry column. Spatial joins can use spatial predicates in the join by expression (e.g., sd_join_by(st_intersects(x$geom(), y$geom()))) or the shorthand sd_join_intersects().

Usage

```
sd_join_by(...)

sd_join_intersects()

sd_join_contains()

sd_join_within()

sd_join_covers()

sd_join_coveredby()

sd_join_touches()

sd_join_crosses()

sd_join_overlaps()

sd_join_equals()

sd_join_dwithin(distance)
```

Arguments

...	Expressions specifying join conditions. These should be comparison expressions (e.g., x$id == y$id, x$value > y$threshold) or spatial predicate expressions (e.g., st_intersects(x$geometry, y$geometry)). Multiple conditions are combined with AND. Like dplyr's join_by(), single columns are parsed as an equijoin condition (e.g., id becomes x$id == y$id).
distance	For a within-distance join, the distance threshold.

Details

For programmatic usage, the `.tables` pronoun may be used to unambiguously refer to a table qualifier (similar to the `.data` pronoun which may be used in single-table SedonaDB verbs).

Value

An object of class `sedonadb_join_by` containing the unevaluated join condition expressions.

Examples

```
# Equality join on id column
sd_join_by(x$id == y$id)

# Can use just the column name as a shorthand
sd_join_by(id)

# Multiple conditions (combined with AND)
sd_join_by(x$id == y$id, x$date >= y$start_date)

# Inequality join
sd_join_by(x$value > y$threshold)

# Spatial joins
sd_join_intersects()
sd_join_dwithin(100)
```

`sd_join_select` *Specify custom post-join column selection*

Description

Use `sd_join_select()` to specify which columns to include in the join result and optionally rename them. Columns may be referenced using `x$column` and `y$column` syntax to disambiguate columns from the left and right tables, or by bare column name when the name exists on only one side of the join.

Usage

```
sd_join_select(...)
```

Arguments

`...`	Named expressions specifying output columns. Each expression may reference a column using `x$column` or `y$column` syntax, or use a bare column name when it is unambiguous. If the same column name exists on both sides of the join, it must be qualified with `x$` or `y$`. The name of the argument becomes the output column name. Unnamed arguments use the original column name (without table prefix).

Value

An object of class `sedonadb_join_select` containing the unevaluated column selection expressions.

Examples

```
# Select and rename columns
sd_join_select(id = x$id, left_value = x$value, right_value = y$value)

# Unnamed arguments keep original column name
sd_join_select(x$id, x$name, y$value)
```

---

sd_join_select_default

Specify default post-join column selection

Description

Use `sd_join_select_default()` to specify that the join result should remove duplicate equijoin key columns (keeping the x-side version) and apply suffixes to any remaining overlapping column names.

Usage

```
sd_join_select_default(suffix = c(".x", ".y"))
```

Arguments

`suffix`	A character vector of length 2 specifying suffixes to add to overlapping column names from the left (x) and right (y) tables.

Value

An object of class `sedonadb_join_select_default` specifying the default column selection behavior.

Examples

```
# Default suffixes
sd_join_select_default()

# Custom suffixes
sd_join_select_default(suffix = c("_left", "_right"))
```

sd_preview *Preview and print the results of running a query*

Description

This is used to implement `print()` for the sedonadb_dataframe or can be used to explicitly preview if `options(sedonadb.interactive = FALSE)`.

Usage

```
sd_preview(.data, n = NULL, ascii = NULL, width = NULL)
```

Arguments

`.data`	A sedonadb_dataframe or an object that can be coerced to one.
`n`	The number of rows to preview. Use `Inf` to preview all rows. Defaults to `getOption("pillar.print_max")`.
`ascii`	Use `TRUE` to force ASCII table formatting or `FALSE` to force unicode formatting. By default, use a heuristic to determine if the output is unicode-friendly or the value of `getOption("cli.unicode")`.
`width`	The character width of the output. Defaults to `getOption("width")`.

Value

.data, invisibly

Examples

```
sd_sql("SELECT 1 as one") |> sd_preview()
```

sd_read_parquet *Create a DataFrame from one or more Parquet files*

Description

The query will only be executed when requested.

Usage

```
sd_read_parquet(path)

sd_ctx_read_parquet(ctx, path)
```

Arguments

path	One or more paths or URIs to Parquet files
ctx	A SedonaDB context.

Value

A sedonadb_dataframe

Examples

```
path <- system.file("files/natural-earth_cities_geo.parquet", package = "sedonadb")
sd_read_parquet(path) |> head(5) |> sd_preview()
```

---

sd_read_sf *Read GDAL/OGR via the sf package*

Description

Uses the ArrowArrayStream interface to GDAL exposed via the sf package to read GDAL/OGR-based data sources.

Usage

```
sd_read_sf(
  dsn,
  layer = NULL,
  ...,
  query = NA,
  options = NULL,
  drivers = NULL,
  filter = NULL,
  fid_column_name = NULL,
  lazy = FALSE
)

sd_ctx_read_sf(
  ctx,
  dsn,
  layer = NULL,
  ...,
  query = NA,
  options = NULL,
  drivers = NULL,
  filter = NULL,
  fid_column_name = NULL,
  lazy = FALSE
)
```

Arguments

dsn, layer	Description of datasource and layer. See sf::read_sf() for details.
...	Currently unused and must be empty
query	A SQL query to pass on to GDAL/OGR.
options	A character vector with layer open options in the form "KEY=VALUE".
drivers	A list of drivers to try if the dsn cannot be guessed.
filter	A spatial object that may be used to filter while reading. In the future SedonaDB will automatically calculate this value based on the query. May be any spatial object that can be converted to WKT via wk::as_wkt(). This filter's CRS must match that of the data.
fid_column_name	An optional name for the feature id (FID) column.
lazy	Use TRUE to stream the data from the source rather than collect first. This can be faster for large data sources but can also be confusing because the data may only be scanned exactly once.
ctx	A SedonaDB context created using sd_connect().

Value

A SedonaDB DataFrame.

Examples

```
nc_gpkg <- system.file("gpkg/nc.gpkg", package = "sf")
sd_read_sf(nc_gpkg)
```

---

sd_register_udf *Register a user-defined function*

Description

Several types of user-defined functions can be registered into a session context. Currently, the only implemented variety is an external pointer to a Rust FFI_ScalarUDF, an example of which is available from the DataFusion Python documentation.

Usage

```
sd_register_udf(udf)

sd_ctx_register_udf(ctx, udf)
```

Arguments

udf	An object of class 'datafusion_scalar_udf'
ctx	A SedonaDB context.

Value

NULL, invisibly

`sd_select` *Keep or drop columns of a SedonaDB DataFrame*

Description

Keep or drop columns of a SedonaDB DataFrame

Usage

```
sd_select(.data, ...)
```

Arguments

`.data`	A sedonadb_dataframe or an object that can be coerced to one.
`...`	One or more bare names. Evaluated like `dplyr::select()`.

Value

An object of class sedonadb_dataframe

Examples

```
data.frame(x = 1:10, y = letters[1:10]) |> sd_select(x)
```

`sd_sql` *Create a DataFrame from SQL*

Description

The query will only be executed when requested.

Usage

```
sd_sql(sql, ..., params = NULL)

sd_ctx_sql(ctx, sql, ..., params = NULL)
```

Arguments

`sql`	A SQL string to execute
`...`	These dots are for future extensions and currently must be empty.
`params`	A list of parameters to fill placeholders in the query.
`ctx`	A SedonaDB context.

Value

A sedonadb_dataframe

Examples

```
sd_sql("SELECT ST_Point(0, 1) as geom")
sd_sql("SELECT ST_Point($1, $2) as geom", params = list(1, 2))
sd_sql("SELECT ST_Point($x, $y) as geom", params = list(x = 1, y = 2))
```

---

sd_summarise *Aggregate SedonaDB DataFrames to a single row per group*

Description

Aggregate SedonaDB DataFrames to a single row per group

Usage

```
sd_summarise(.data, ..., .env = parent.frame())

sd_summarize(.data, ..., .env = parent.frame())
```

Arguments

.data	A sedonadb_dataframe or an object that can be coerced to one.
...	Aggregate expressions. These are evaluated in the same way as dplyr::summarise() except the outer expression must be an aggregate expression (e.g., sum(x) + 1 is not currently possible).
.env	The calling environment for programmatic usage

Value

An object of class sedonadb_dataframe

Examples

```
data.frame(x = c(10:1, NA)) |> sd_summarise(x = sum(x, na.rm = TRUE))
```

sd_to_view *Register a DataFrame as a named view*

Description

This is useful for creating a view that can be referenced in a SQL statement. Use sd_drop_view() to remove it.

Usage

```
sd_to_view(.data, table_ref, overwrite = FALSE, ctx = NULL)
```

Arguments

.data	A sedonadb_dataframe or an object that can be coerced to one.
table_ref	The name of the view reference
overwrite	Use TRUE to overwrite a view with the same name (if it exists)
ctx	A SedonaDB context.

Value

.data, invisibly

Examples

```
sd_sql("SELECT 1 as one") |> sd_to_view("foofy")
sd_sql("SELECT * FROM foofy")
```

sd_transmute *Create, modify, and delete columns of a SedonaDB DataFrame*

Description

Create, modify, and delete columns of a SedonaDB DataFrame

Usage

```
sd_transmute(.data, ...)
```

Arguments

.data	A sedonadb_dataframe or an object that can be coerced to one.
...	Named expressions for new columns to create. These are evaluated in the same way as dplyr::transmute() except does not support extra dplyr features such as across() or .by.

Value

An object of class sedonadb_dataframe

Examples

```
data.frame(x = 1:10) |>
  sd_transmute(y = x + 1L)
```

`sd_with_params` *Fill in placeholders*

Description

This is a slightly more verbose form of `sd_sql()` with `params` that is useful if a data frame is to be repeatedly queried.

Usage

```
sd_with_params(.data, ...)
```

Arguments

`.data`	A sedonadb_dataframe or an object that can be coerced to one.
`...`	Named or unnamed parameters that will be coerced to literals with `as_sedonadb_literal()`.

Value

A sedonadb_dataframe with the provided parameters filled into the query

Examples

```
sd_sql("SELECT ST_Point($1, $2) as pt") |>
  sd_with_params(11, 12)
sd_sql("SELECT ST_Point($x, $y) as pt") |>
  sd_with_params(x = 11, y = 12)
```

sd_write_parquet *Write DataFrame to (Geo)Parquet files*

Description

Write this DataFrame to one or more (Geo)Parquet files. For input that contains geometry columns, GeoParquet metadata is written such that suitable readers can recreate Geometry/Geography types when reading the output and potentially read fewer row groups when only a subset of the file is needed for a given query.

Usage

```
sd_write_parquet(
  .data,
  path,
  options = NULL,
  partition_by = character(0),
  sort_by = character(0),
  single_file_output = NULL,
  geoparquet_version = "1.0",
  overwrite_bbox_columns = FALSE,
  max_row_group_size = NULL,
  compression = NULL
)
```

Arguments

.data	A sedonadb_dataframe or an object that can be coerced to one.
path	A filename or directory to which parquet file(s) should be written
options	A named list of key/value options to be used when constructing a parquet writer. Common options are exposed as other arguments to `sd_write_parquet()`; however, this argument allows setting any DataFusion Parquet writer option. If an option is specified here and by another argument to this function, the value specified as an explicit argument takes precedence.
partition_by	A character vector of column names to partition by. If non-empty, applies hive-style partitioning to the output
sort_by	A character vector of column names to sort by. Currently only ascending sort is supported
single_file_output	Use TRUE or FALSE to force writing a single Parquet file vs. writing one file per partition to a directory. By default, a single file is written if `partition_by` is unspecified and `path` ends with `.parquet`
geoparquet_version	GeoParquet metadata version to write if output contains one or more geometry columns. The default ("1.0") is the most widely supported and will result in

geometry columns being recognized in many readers; however, only includes statistics at the file level. Use "1.1" to compute an additional bounding box column for every geometry column in the output: some readers can use these columns to prune row groups when files contain an effective spatial ordering. The extra columns will appear just before their geometry column and will be named "[geom_col_name]_bbox" for all geometry columns except "geometry", whose bounding box column name is just "bbox"

`overwrite_bbox_columns`
: Use TRUE to overwrite any bounding box columns that already exist in the input. This is useful in a read -> modify -> write scenario to ensure these columns are up-to-date. If FALSE (the default), an error will be raised if a bbox column already exists

`max_row_group_size`
: Target maximum number of rows in each row group. Defaults to the global configuration value (1M rows).

`compression`
: Sets the Parquet compression codec. Valid values are: uncompressed, snappy, gzip(level), brotli(level), lz4, zstd(level), and lz4_raw. Defaults to the global configuration value (zstd(3)).

Value

The input, invisibly

Examples

```
tmp_parquet <- tempfile(fileext = ".parquet")

sd_sql("SELECT ST_Point(1, 2, 4326) as geom") |>
  sd_write_parquet(tmp_parquet)

sd_read_parquet(tmp_parquet)
unlink(tmp_parquet)
```

---

sedonadb_adbc *SedonaDB ADBC Driver*

Description

SedonaDB ADBC Driver

Usage

```
sedonadb_adbc()
```

Value

An `adbcdrivermanager::adbc_driver()` of class 'sedonadb_driver_sedonadb'

Examples

```
library(adbcdrivermanager)

con <- sedonadb_adbc() |>
  adbc_database_init() |>
  adbc_connection_init()
con |>
  read_adbc("SELECT ST_Point(0, 1) as geometry") |>
  as.data.frame()
```

Index

∗ **datasets**
    .fns, 2
    .tables, 3
.fns, 2
.tables, 3

adbcdrivermanager::adbc_driver(), *25*
as_sd_expr *(*sd_expr_column*)*, 9
as_sd_expr(), *10*
as_sedonadb_dataframe, 3
as_sedonadb_literal, 4
as_sedonadb_literal(), *23*

dplyr::arrange(), *4*
dplyr::filter(), *11*
dplyr::group_by(), *11, 12*
dplyr::select(), *20*
dplyr::summarise(), *21*
dplyr::transmute(), *22*

is_sd_expr *(*sd_expr_column*)*, 9

nanoarrow::convert_array_stream, *5*

sd_anti_join *(*sd_join*)*, 12
sd_arrange, 4
sd_collect *(*sd_compute*)*, 5
sd_compute, 5
sd_configure_proj, 6
sd_connect, 7
sd_connect(), *19*
sd_count, 8
sd_cross_join *(*sd_join*)*, 12
sd_ctx_drop_view *(*sd_drop_view*)*, 8
sd_ctx_read_parquet *(*sd_read_parquet*)*, 17
sd_ctx_read_sf *(*sd_read_sf*)*, 18
sd_ctx_register_udf *(*sd_register_udf*)*, 19
sd_ctx_sql *(*sd_sql*)*, 20
sd_ctx_view *(*sd_drop_view*)*, 8
sd_drop_view, 8
sd_drop_view(), *22*
sd_expr_aggregate_function *(*sd_expr_column*)*, 9
sd_expr_alias *(*sd_expr_column*)*, 9
sd_expr_any_function *(*sd_expr_column*)*, 9
sd_expr_binary *(*sd_expr_column*)*, 9
sd_expr_cast *(*sd_expr_column*)*, 9
sd_expr_column, 9
sd_expr_column(), *13*
sd_expr_factory *(*sd_expr_column*)*, 9
sd_expr_factory(), *10*
sd_expr_literal *(*sd_expr_column*)*, 9
sd_expr_literal(), *4*
sd_expr_negative *(*sd_expr_column*)*, 9
sd_expr_parse_binary *(*sd_expr_column*)*, 9
sd_expr_scalar_function *(*sd_expr_column*)*, 9
sd_filter, 11
sd_full_join *(*sd_join*)*, 12
sd_group_by, 11
sd_inner_join *(*sd_join*)*, 12
sd_join, 12
sd_join(), *14*
sd_join_by, 14
sd_join_by(), *3*, *12, 13*
sd_join_contains *(*sd_join_by*)*, 14
sd_join_contains(), *13*
sd_join_coveredby *(*sd_join_by*)*, 14
sd_join_coveredby(), *13*
sd_join_covers *(*sd_join_by*)*, 14
sd_join_covers(), *13*
sd_join_crosses *(*sd_join_by*)*, 14
sd_join_crosses(), *13*
sd_join_dwithin *(*sd_join_by*)*, 14
sd_join_equals *(*sd_join_by*)*, 14
sd_join_equals(), *13*
sd_join_intersects *(*sd_join_by*)*, 14
sd_join_intersects(), *13*

sd_join_overlaps *(*sd_join_by*)*, 14
sd_join_overlaps(), *13*
sd_join_select, 15
sd_join_select(), *3*, *13*
sd_join_select_default, 16
sd_join_select_default(), *13*
sd_join_touches *(*sd_join_by*)*, 14
sd_join_touches(), *13*
sd_join_within *(*sd_join_by*)*, 14
sd_join_within(), *13*
sd_left_join *(*sd_join*)*, 12
sd_preview, 17
sd_read_parquet, 17
sd_read_sf, 18
sd_register_udf, 19
sd_right_join *(*sd_join*)*, 12
sd_select, 20
sd_semi_join *(*sd_join*)*, 12
sd_sql, 20
sd_sql(), *23*
sd_summarise, 21
sd_summarize *(*sd_summarise*)*, 21
sd_to_view, 22
sd_to_view(), *8*
sd_transmute, 22
sd_ungroup *(*sd_group_by*)*, 11
sd_view *(*sd_drop_view*)*, 8
sd_with_params, 23
sd_write_parquet, 24
sedonadb_adbc, 25
sf::read_sf(), *19*

wk::as_wkt(), *19*