# Package: sedonadb (via r-universe)

January 16, 2026

**Title** Bindings for Apache SedonaDB

**Version** 0.2.0

**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** 7.3.2

**SystemRequirements** Cargo (Rust's package manager), rustc

**Depends** R (>= 4.1.0)

**Suggests** adbcdrivermanager, rlang, sf, testthat (>= 3.0.0), withr, wk

**Imports** geoarrow, nanoarrow

**Config/testthat/edition** 3

**Config/build/bootstrap** TRUE

**Config/pak/sysreqs** libzstd-dev libclang-dev

**Repository** https://r-multiverse-staging.r-universe.dev

**Date/Publication** 2025-12-01 15:30:33 UTC

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

**RemoteRef** apache-sedona-db-0.2.0

**RemoteSha** 69f89a9a0cced55a3792eb6dfe1f76e9dd01033c

**RemoteSubdir** r/sedonadb

## Contents

---

`as_sedonadb_dataframe` *Convert an object to a DataFrame*

---

## Description

Convert an object to a DataFrame

## Usage

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

## Arguments

| | |
|---|---|
| `x` | An object to convert |
| `...` | Extra arguments passed to/from methods |
| `schema` | The requested schema |

## Value

A sedonadb_dataframe

## Examples

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

## 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 |
| `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:<br>• "homebrew": Look for PROJ installed by Homebrew. This is the easiest option on MacOS.<br>• "system": Look for PROJ in the platform library load path (e.g., after installing system proj on Linux).<br>• "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_count *Count rows in a DataFrame*

---

### Description

Count rows in a DataFrame

### Usage

```
sd_count(.data)
```

### Arguments

| | |
|---|---|
| .data | A sedonadb_dataframe |

### 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_view(table_ref)
```

### Arguments

| | |
|---|---|
| `table_ref` | The name of the view reference |

### 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_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 |
| 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)
```

### Arguments

| | |
|---|---|
| path | One or more paths or URIs to Parquet files |

### 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_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)
```

### Arguments

| | |
|---|---|
| `udf` | An object of class 'datafusion_scalar_udf' |

### Value

NULL, invisibly

## sd_sql *Create a DataFrame from SQL*

### Description

The query will only be executed when requested.

### Usage

```
sd_sql(sql)
```

### Arguments

| | |
|---|---|
| `sql` | A SQL string to execute |

### Value

A sedonadb_dataframe

### Examples

```
sd_sql("SELECT ST_Point(0, 1) as geom") |> sd_preview()
```

## 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)
```

### Arguments

| | |
|---|---|
| .data | A sedonadb_dataframe |
| table_ref | The name of the view reference |
| overwrite | Use TRUE to overwrite a view with the same name (if it exists) |

### Value

.data, invisibly

### Examples

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

## 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,
  partition_by = character(0),
  sort_by = character(0),
  single_file_output = NULL,
  geoparquet_version = "1.0",
  overwrite_bbox_columns = FALSE
)
```

## Arguments

| | |
|---|---|
| .data | A sedonadb_dataframe |
| path | A filename or directory to which parquet file(s) should be written |
| 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 |

## Value

The input, invisibly

## Examples

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

```
sd_sql("SELECT ST_SetSRID(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