Package 'duckdbfs'

October 19, 2023

Title High Performance Remote File System Access Using 'duckdb'

Version 0.0.3

Description Provides friendly wrappers for creating 'duckdb'-backed connections to tabular datasets ('csv', parquet, etc) on local or remote file systems. This mimics the behaviour of ``open_dataset'' in the 'arrow' package, but in addition to 'S3' file system also generalizes to any list of 'http' URLs.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

URL https://github.com/cboettig/duckdbfs, https://cboettig.github.io/duckdbfs/

BugReports https://github.com/cboettig/duckdbfs/issues

Imports DBI, dbplyr, dplyr, duckdb (>= 0.8.1)

Suggests curl, sf, jsonlite, spelling, miniocclient, testthat (>= 3.0.0)

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Author Carl Boettiger [aut, cre] (<https://orcid.org/0000-0002-1642-628X>)

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN

Date/Publication 2023-10-19 19:40:02 UTC

R topics documented:

`cached_connection` *create a cachable duckdb connection*

Description

This function is primarily intended for internal use by other `duckdbfs` functions. However, it can be called directly by the user whenever it is desirable to have direct access to the connection object.

Usage

```
cached_connection(dbdir = ":memory:", read_only = FALSE)
```

Arguments

`dbdir`	Location for database files. Should be a path to an existing directory in the file system. With the default, all data is kept in RAM
`read_only`	Set to `TRUE` for read-only operation

Details

When first called (by a user or internal function), this function both creates a duckdb connection and places that connection into a cache (`duckdbfs_conn` option). On subsequent calls, this function returns the cached connection, rather than recreating a fresh connection.

This frees the user from the responsibility of managing a connection object, because functions needing access to the connection can use this to create or access the existing connection. At the close of the global environment, this function's finalizer should gracefully shutdown the connection before removing the cache.

Value

a `duckdb::duckdb()` connection object

Examples

```
con <- cached_connection()
close_connection(con)
```

`close_connection` *close connection*

Description

close connection

Usage

```
close_connection(conn = cached_connection())
```

Arguments

conn	a duckdb connection (leave blank) Closes the invisible cached connection to duckdb

Details

Shuts down connection before gc removes it. Then clear cached reference to avoid using a stale connection This avoids complaint about connection being garbage collected.

Value

returns nothing.

Examples

```
close_connection()
```

`duckdb_s3_config` *Configure S3 settings for database connection*

Description

This function is used to configure S3 settings for a database connection. It allows you to set various S3-related parameters such as access key, secret access key, endpoint, region, session token, uploader settings, URL compatibility mode, URL style, and SSL usage.

Usage

```
duckdb_s3_config(
  conn = cached_connection(),
  s3_access_key_id = NULL,
  s3_secret_access_key = NULL,
  s3_endpoint = NULL,
  s3_region = NULL,
  s3_session_token = NULL,
  s3_uploader_max_filesize = NULL,
  s3_uploader_max_parts_per_file = NULL,
  s3_uploader_thread_limit = NULL,
  s3_url_compatibility_mode = NULL,
  s3_url_style = NULL,
  s3_use_ssl = NULL,
  anonymous = NULL
)
```

Arguments

conn	A database connection object created using the cache_connection function (default: cache_connection()).
s3_access_key_id	The S3 access key ID (default: NULL).
s3_secret_access_key	The S3 secret access key (default: NULL).
s3_endpoint	The S3 endpoint (default: NULL).
s3_region	The S3 region (default: NULL).
s3_session_token	The S3 session token (default: NULL).
s3_uploader_max_filesize	The maximum filesize for S3 uploader (between 50GB and 5TB, default 800GB).
s3_uploader_max_parts_per_file	The maximum number of parts per file for S3 uploader (between 1 and 10000, default 10000).
s3_uploader_thread_limit	The thread limit for S3 uploader (default: 50).
s3_url_compatibility_mode	Disable Globs and Query Parameters on S3 URLs (default: 0, allows globs/queries).
s3_url_style	The style of S3 URLs to use. Default is "vhost" unless s3_endpoint is set, which makes default "path" (i.e. MINIO systems).
s3_use_ssl	Enable or disable SSL for S3 connections (default: 1 (TRUE)).
anonymous	request anonymous access (sets s3_access_key_id and s3_secret_access_key to "", allowing anonymous access to public buckets).

Details

see https://duckdb.org/docs/sql/configuration.html

Value

Returns silently (NULL) if successful.

Examples

```
# Configure S3 settings
duckdb_s3_config(
            s3_access_key_id = "YOUR_ACCESS_KEY_ID",
            s3_secret_access_key = "YOUR_SECRET_ACCESS_KEY",
            s3_endpoint = "YOUR_S3_ENDPOINT",
            s3_region = "YOUR_S3_REGION",
            s3_uploader_max_filesize = "800GB",
            s3_uploader_max_parts_per_file = 100,
            s3_uploader_thread_limit = 8,
            s3_url_compatibility_mode = FALSE,
            s3_url_style = "vhost",
            s3_use_ssl = TRUE,
            anonymous = TRUE)
```

---

load_spatial *load the duckdb geospatial data plugin*

Description

load the duckdb geospatial data plugin

Usage

```
load_spatial(conn = cached_connection())
```

Arguments

conn	A database connection object created using the cache_connection function (default: cache_connection()).

Value

loads the extension and returns status invisibly.

References

https://duckdb.org/docs/extensions/spatial.html

`open_dataset` *Open a dataset from a variety of sources*

Description

This function opens a dataset from a variety of sources, including Parquet, CSV, etc, using either local file system paths, URLs, or S3 bucket URI notation.

Usage

```
open_dataset(
  sources,
  schema = NULL,
  hive_style = TRUE,
  unify_schemas = FALSE,
  format = c("parquet", "csv", "tsv", "text"),
  conn = cached_connection(),
  tblname = tmp_tbl_name(),
  mode = "VIEW",
  filename = FALSE,
  recursive = TRUE,
  ...
)
```

Arguments

`sources`	A character vector of paths to the dataset files.
`schema`	The schema for the dataset. If NULL, the schema will be inferred from the dataset files.
`hive_style`	A logical value indicating whether to the dataset uses Hive-style partitioning.
`unify_schemas`	A logical value indicating whether to unify the schemas of the dataset files (union_by_name). If TRUE, will execute a UNION by column name across all files (NOTE: this can add considerably to the initial execution time)
`format`	The format of the dataset files. One of `"parquet"`, `"csv"`, `"tsv"`, or `"text"`.
`conn`	A connection to a database.
`tblname`	The name of the table to create in the database.
`mode`	The mode to create the table in. One of `"VIEW"` or `"TABLE"`. Creating a `VIEW`, the default, will execute more quickly because it does not create a local copy of the dataset. `TABLE` will create a local copy in duckdb's native format, downloading the full dataset if necessary. When using `TABLE` mode with large data, please be sure to use a conn connections with disk-based storage, e.g. by calling `cached_connection()`, e.g. `cached_connection("storage_path")`, otherwise the full data must fit into RAM. Using `TABLE` assumes familiarity with R's DBI-based interface.
`filename`	A logical value indicating whether to include the filename in the table name.

recursive	should we assume recursive path? default TRUE. Set to FALSE if trying to open a single, un-partitioned file.
...	optional additional arguments passed to duckdb_s3_config(). Note these apply after those set by the URI notation and thus may be used to override or provide settings not supported in that format.

Value

A lazy dplyr::tbl object representing the opened dataset backed by a duckdb SQL connection. Most dplyr (and some tidyr) verbs can be used directly on this object, as they can be translated into SQL commands automatically via dbplyr. Generic R commands require using dplyr::collect() on the table, which forces evaluation and reading the resulting data into memory.

Examples

```
# A remote, hive-partitioned Parquet dataset
base <- paste0("https://github.com/duckdb/duckdb/raw/main/",
               "data/parquet-testing/hive-partitioning/union_by_name/")
f1 <- paste0(base, "x=1/f1.parquet")
f2 <- paste0(base, "x=1/f2.parquet")
f3 <- paste0(base, "x=2/f2.parquet")

open_dataset(c(f1,f2,f3), unify_schemas = TRUE)

# Access an S3 database specifying an independently-hosted (MINIO) endpoint
efi <- open_dataset("s3://neon4cast-scores/parquet/aquatics",
                    s3_access_key_id="",
                    s3_endpoint="data.ecoforecast.org")
```

---

to_sf *Convert output to sf object*

Description

Convert output to sf object

Usage

```
to_sf(x, conn = cached_connection())
```

Arguments

x	a remote duckdb tbl (from open_dataset) or dplyr-pipeline thereof.

conn — the connection object from the tbl. Takes a duckdb table (from `open_dataset`) or a dataset or dplyr pipline and returns an sf object. **Important**: the table must have a `geometry` column, which you will almost always have to create first.

Note: `to_sf()` triggers collection into R. This function is suitable to use at the end of a dplyr pipeline that will subset the data. Using this function on a large dataset without filtering first may exceed available memory.

Value

an `sf` class object (in memory).

Examples

```
library(dplyr)
csv_file <- system.file("extdata/spatial-test.csv", package="duckdbfs")

# Note that we almost always must first create a `geometry` column, e.g.
# from lat/long columns using the `st_point` method.
sf <-
  open_dataset(csv_file, format = "csv") |>
  mutate(geometry = ST_Point(longitude, latitude)) |>
  to_sf()

# We can use the full space of spatial operations, including spatial
# and normal dplyr filters.  All operations are translated into a
# spatial SQL query by `to_sf`:
open_dataset(csv_file, format = "csv") |>
  mutate(geometry = ST_Point(longitude, latitude)) |>
  mutate(dist = ST_Distance(geometry, ST_Point(0,0))) |>
  filter(site %in% c("a", "b", "e")) |>
  to_sf()
```

`write_dataset` *write_dataset*

Description

write_dataset

Usage

```
write_dataset(
  dataset,
  path,
  conn = cached_connection(),
```

```
  format = c("parquet", "csv"),
  partitioning = dplyr::group_vars(dataset),
  overwrite = TRUE,
  ...
)
```

Arguments

dataset	a remote tbl object from open_dataset, or an in-memory data.frame.
path	a local file path or S3 path with write credentials
conn	duckdbfs database connection
format	export format
partitioning	names of columns to use as partition variables
overwrite	allow overwriting of existing files?
...	additional arguments to duckdb_s3_config()

Value

Returns the path, invisibly.

Examples

```
write_dataset(mtcars, tempfile())


write_dataset(mtcars, tempdir())
```

Index