Package: duckdb (via r-universe)

November 22, 2024

Title DBI Package for the DuckDB Database Management System

Version 1.1.2

Description The DuckDB project is an embedded analytical data management system with support for the Structured Query Language (SQL). This package includes all of DuckDB and an R Database Interface (DBI) connector.

License MIT + file LICENSE

URL https://r.duckdb.org/, https://github.com/duckdb/duckdb-r

BugReports https://github.com/duckdb/duckdb-r/issues

Depends DBI, R (>= 3.6.0)

Imports methods, utils

Suggests adbcdrivermanager, arrow (>= 13.0.0), bit64, callr, clock, DBItest, dbplyr, dplyr, rlang, testthat, tibble, vctrs, withr

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2.9000

Config/build/compilation-database true

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

RemoteUrl https://github.com/duckdb/duckdb-r

RemoteRef v1.1.2

RemoteSha 71c71a6b824bd0f2c0fca43c5dfa53645b17bfb7

Contents

---

backend-duckdb *DuckDB SQL backend for dbplyr*

Description

This is a SQL backend for dbplyr tailored to take into account DuckDB's possibilities. This mainly follows the backend for PostgreSQL, but contains more mapped functions.

`tbl_file()` is an experimental variant of `dplyr::tbl()` to directly access files on disk. It is safer than `dplyr::tbl()` because there is no risk of misinterpreting the request, and paths with special characters are supported.

`tbl_function()` is an experimental variant of `dplyr::tbl()` to create a lazy table from a table-generating function, useful for reading nonstandard CSV files or other data sources. It is safer than `dplyr::tbl()` because there is no risk of misinterpreting the query. See `https://duckdb.org/docs/data/overview` for details on data importing functions.

As an alternative, use `dplyr::tbl(src, dplyr::sql("SELECT ... FROM ..."))` for custom SQL queries.

`tbl_query()` is deprecated in favor of `tbl_function()`.

Use `simulate_duckdb()` with `lazy_frame()` to see simulated SQL without opening a DuckDB connection.

Usage

```
tbl_file(src, path, ..., cache = FALSE)

tbl_function(src, query, ..., cache = FALSE)

tbl_query(src, query, ...)

simulate_duckdb(...)
```

Arguments

`src`	A duckdb connection object
`path`	Path to existing Parquet, CSV or JSON file
`...`	Any parameters to be forwarded
`cache`	Enable object cache for Parquet files
`query`	SQL code, omitting the `FROM` clause

Examples

```
library(dplyr, warn.conflicts = FALSE)
con <- DBI::dbConnect(duckdb(), path = ":memory:")

db <- copy_to(con, data.frame(a = 1:3, b = letters[2:4]))

db %>%
  filter(a > 1) %>%
  select(b)

path <- tempfile(fileext = ".csv")
write.csv(data.frame(a = 1:3, b = letters[2:4]))

db_csv <- tbl_file(con, path)
db_csv %>%
  summarize(sum_a = sum(a))

db_csv_fun <- tbl_function(con, paste0("read_csv_auto('", path, "')"))
db_csv %>%
  count()

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

---

duckdb *Connect to a DuckDB database instance*

Description

`duckdb()` creates or reuses a database instance.

`duckdb_shutdown()` shuts down a database instance.

Return an `adbcdrivermanager::adbc_driver()` for use with Arrow Database Connectivity via the adbcdrivermanager package.

`dbConnect()` connects to a database instance.

`dbDisconnect()` closes a DuckDB database connection. The associated DuckDB database instance is shut down automatically, it is no longer necessary to set `shutdown = TRUE` or to call `duckdb_shutdown()`.

Usage

```
duckdb(
  dbdir = DBDIR_MEMORY,
  read_only = FALSE,
  bigint = "numeric",
  config = list()
)
```

```
duckdb_shutdown(drv)

duckdb_adbc()

## S4 method for signature 'duckdb_driver'
dbConnect(
  drv,
  dbdir = DBDIR_MEMORY,
  ...,
  debug = getOption("duckdb.debug", FALSE),
  read_only = FALSE,
  timezone_out = "UTC",
  tz_out_convert = c("with", "force"),
  config = list(),
  bigint = "numeric"
)

## S4 method for signature 'duckdb_connection'
dbDisconnect(conn, ..., shutdown = TRUE)
```

Arguments

dbdir	Location for database files. Should be a path to an existing directory in the file system. With the default (or `""`), all data is kept in RAM.
read_only	Set to `TRUE` for read-only operation. For file-based databases, this is only applied when the database file is opened for the first time. Subsequent connections (via the same `drv` object or a `drv` object pointing to the same path) will silently ignore this flag.
bigint	How 64-bit integers should be returned. There are two options: `"numeric"` and `"integer64"`. If `"numeric"` is selected, bigint integers will be treated as double/numeric. If `"integer64"` is selected, bigint integers will be set to bit64 encoding.
config	Named list with DuckDB configuration flags, see https://duckdb.org/docs/configuration/overview#configuration-reference for the possible options. These flags are only applied when the database object is instantiated. Subsequent connections will silently ignore these flags.
drv	Object returned by `duckdb()`
...	Ignored
debug	Print additional debug information such as queries
timezone_out	The time zone returned to R, defaults to `"UTC"`, which is currently the only timezone supported by duckdb. If you want to display datetime values in the local timezone, set to `Sys.timezone()` or `""`.
tz_out_convert	How to convert timestamp columns to the timezone specified in `timezone_out`. There are two options: `"with"`, and `"force"`. If `"with"` is chosen, the timestamp will be returned as it would appear in the specified time zone. If `"force"` is chosen, the timestamp will have the same clock time as the timestamp in the database, but with the new time zone.

conn	A duckdb_connection object
shutdown	Unused. The database instance is shut down automatically.

Value

duckdb() returns an object of class duckdb_driver.

dbDisconnect() and duckdb_shutdown() are called for their side effect.

An object of class "adbc_driver"

dbConnect() returns an object of class duckdb_connection.

Examples

```
library(adbcdrivermanager)
with_adbc(db <- adbc_database_init(duckdb_adbc()), {
  as.data.frame(read_adbc(db, "SELECT 1 as one;"))
})

drv <- duckdb()
con <- dbConnect(drv)

dbGetQuery(con, "SELECT 'Hello, world!'")

dbDisconnect(con)
duckdb_shutdown(drv)

# Shorter:
con <- dbConnect(duckdb())
dbGetQuery(con, "SELECT 'Hello, world!'")
dbDisconnect(con, shutdown = TRUE)
```

---

duckdb_explain-class *DuckDB EXPLAIN query tree*

Description

DuckDB EXPLAIN query tree

duckdb_get_substrait *Get the Substrait plan for a SQL query Transforms a SQL query into a raw vector containing the serialized Substrait query blob*

Description

Get the Substrait plan for a SQL query Transforms a SQL query into a raw vector containing the serialized Substrait query blob

Usage

```
duckdb_get_substrait(conn, query, enable_optimizer = TRUE)
```

Arguments

conn	A DuckDB connection, created by `dbConnect()`.
query	The query string in SQL
enable_optimizer	Optional parameter to enable/disable query-optimizer. By default optimizer is enabled.

Value

A raw vector containing the substrait protobuf blob

duckdb_get_substrait_json *Get the Substrait plan for a SQL query in the JSON format Transforms a SQL query into a vector containing the serialized Substrait query JSON*

Description

Get the Substrait plan for a SQL query in the JSON format Transforms a SQL query into a vector containing the serialized Substrait query JSON

Usage

```
duckdb_get_substrait_json(conn, query, enable_optimizer = TRUE)
```

Arguments

conn	A DuckDB connection, created by `dbConnect()`.
query	The query string in SQL
enable_optimizer	Optional parameter to enable/disable query-optimizer. By default optimizer is enabled.

Value

A vector containing the substrait protobuf JSON

`duckdb_prepare_substrait`

Query DuckDB using Substrait Method for interpreting a Substrait BLOB plan as a DuckDB Query Plan It interprets and executes the query.

Description

Query DuckDB using Substrait Method for interpreting a Substrait BLOB plan as a DuckDB Query Plan It interprets and executes the query.

Usage

```
duckdb_prepare_substrait(conn, query, arrow = FALSE)
```

Arguments

`conn`	A DuckDB connection, created by `dbConnect()`.
`query`	The Protobuf-encoded Substrait Query Plan. Qack!
`arrow`	Whether the result should be in Arrow format

Value

A DuckDB Query Result

`duckdb_prepare_substrait_json`

Query DuckDB using Substrait Method for interpreting a Substrait JSON plan as a DuckDB Query Plan It interprets and executes the query.

Description

Query DuckDB using Substrait Method for interpreting a Substrait JSON plan as a DuckDB Query Plan It interprets and executes the query.

Usage

```
duckdb_prepare_substrait_json(conn, json, arrow = FALSE)
```

Arguments

conn	A DuckDB connection, created by dbConnect().
json	The Json Query Plan. Qack!
arrow	Whether the result should be in Arrow format

Value

A DuckDB Query Result

---

duckdb_read_csv *Reads a CSV file into DuckDB*

Description

Directly reads a CSV file into DuckDB, tries to detect and create the correct schema for it. This usually is much faster than reading the data into R and writing it to DuckDB.

Usage

```
duckdb_read_csv(
  conn,
  name,
  files,
  ...,
  header = TRUE,
  na.strings = "",
  nrow.check = 500,
  delim = ",",
  quote = "\"",
  col.names = NULL,
  col.types = NULL,
  lower.case.names = FALSE,
  sep = delim,
  transaction = TRUE,
  temporary = FALSE
)
```

Arguments

conn	A DuckDB connection, created by dbConnect().
name	The name for the virtual table that is registered or unregistered
files	One or more CSV file names, should all have the same structure though
...	Reserved for future extensions, must be empty.
header	Whether or not the CSV files have a separate header in the first line
na.strings	Which strings in the CSV files should be considered to be NULL

`nrow.check`	How many rows should be read from the CSV file to figure out data types
`delim`	Which field separator should be used
`quote`	Which quote character is used for columns in the CSV file
`col.names`	Override the detected or generated column names
`col.types`	Character vector of column types in the same order as col.names, or a named character vector where names are column names and types pairs. Valid types are DuckDB data types, e.g. VARCHAR, DOUBLE, DATE, BIGINT, BOOLEAN, etc.
`lower.case.names`	Transform column names to lower case
`sep`	Alias for delim for compatibility
`transaction`	Should a transaction be used for the entire operation
`temporary`	Set to `TRUE` to create a temporary table

Details

If the table already exists in the database, the csv is appended to it. Otherwise the table is created.

Value

The number of rows in the resulted table, invisibly.

Examples

```
con <- dbConnect(duckdb())

data <- data.frame(a = 1:3, b = letters[1:3])
path <- tempfile(fileext = ".csv")

write.csv(data, path, row.names = FALSE)

duckdb_read_csv(con, "data", path)
dbReadTable(con, "data")

dbDisconnect(con)


# Providing data types for columns
path <- tempfile(fileext = ".csv")
write.csv(iris, path, row.names = FALSE)

con <- dbConnect(duckdb())
duckdb_read_csv(con, "iris", path,
  col.types = c(
    Sepal.Length = "DOUBLE",
    Sepal.Width = "DOUBLE",
    Petal.Length = "DOUBLE",
    Petal.Width = "DOUBLE",
    Species = "VARCHAR"
```

```
  )
)
dbReadTable(con, "iris")
dbDisconnect(con)
```

`duckdb_register` *Register a data frame as a virtual table*

Description

`duckdb_register()` registers a data frame as a virtual table (view) in a DuckDB connection. No data is copied.

Usage

```
duckdb_register(conn, name, df, overwrite = FALSE, experimental = FALSE)

duckdb_unregister(conn, name)
```

Arguments

`conn`	A DuckDB connection, created by `dbConnect()`.
`name`	The name for the virtual table that is registered or unregistered
`df`	A `data.frame` with the data for the virtual table
`overwrite`	Should an existing registration be overwritten?
`experimental`	Enable experimental optimizations

Details

`duckdb_unregister()` unregisters a previously registered data frame.

Value

These functions are called for their side effect.

Examples

```
con <- dbConnect(duckdb())

data <- data.frame(a = 1:3, b = letters[1:3])

duckdb_register(con, "data", data)
dbReadTable(con, "data")

duckdb_unregister(con, "data")

dbDisconnect(con)
```

duckdb_register_arrow *Register an Arrow data source as a virtual table*

Description

duckdb_register_arrow() registers an Arrow data source as a virtual table (view) in a DuckDB connection. No data is copied.

Usage

```
duckdb_register_arrow(conn, name, arrow_scannable, use_async = NULL)

duckdb_unregister_arrow(conn, name)

duckdb_list_arrow(conn)
```

Arguments

conn	A DuckDB connection, created by dbConnect().
name	The name for the virtual table that is registered or unregistered
arrow_scannable	A scannable Arrow-object
use_async	Switched to the asynchronous scanner. (deprecated)

Details

duckdb_unregister_arrow() unregisters a previously registered data frame.

Value

These functions are called for their side effect.

Index