

Package ‘duckdb’

June 23, 2025

Title DBI Package for the DuckDB Database Management System

Version 1.3.1

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 (>= 4.1.0)

Imports methods, utils

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

Config/build/compilation-database false

Config/build/never-clean true

Config/comment/compilation-database Generate manually with pkgload::generate_db() for faster pkgload::load_all()

Config/gha/extra-packages arrow=?ignore-before-r=4.2.0

Encoding UTF-8

RoxygenNote 7.3.2.9000

SystemRequirements xz (for building from source)

Biarch true

NeedsCompilation yes

Author Hannes Mühleisen [aut] (ORCID: <<https://orcid.org/0000-0001-8552-0029>>),
Mark Raasveldt [aut] (ORCID: <<https://orcid.org/0000-0001-5005-6844>>),
Kirill Müller [cre] (ORCID: <<https://orcid.org/0000-0002-1416-3412>>),
Stichting DuckDB Foundation [cph],
Apache Software Foundation [cph],
PostgreSQL Global Development Group [cph],
The Regents of the University of California [cph],

Cameron Desrochers [cph],
Victor Zverovich [cph],
RAD Game Tools [cph],
Valve Software [cph],
Rich Geldreich [cph],
Tenacious Software LLC [cph],
The RE2 Authors [cph],
Google Inc. [cph],
Facebook Inc. [cph],
Steven G. Johnson [cph],
Jiahao Chen [cph],
Tony Kelman [cph],
Jonas Fonseca [cph],
Lukas Fittl [cph],
Salvatore Sanfilippo [cph],
Art.sy, Inc. [cph],
Oran Agra [cph],
Redis Labs, Inc. [cph],
Melissa O'Neill [cph],
PCG Project contributors [cph]

Maintainer Kirill Müller <kirill@cynkra.com>

Repository CRAN

Date/Publication 2025-06-23 12:10:07 UTC

Contents

backend-duckdb	2
duckdb	4
duckdb_explain-class	6
duckdb_read_csv	6
duckdb_register	8
duckdb_register_arrow	9

Index	11
--------------	-----------

backend-duckdb	<i>DuckDB SQL backend for dbplyr</i>
----------------	--------------------------------------

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 `dbplyr::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

<code>src</code>	A duckdb connection object
<code>path</code>	Path to existing Parquet, CSV or JSON file
<code>...</code>	Any parameters to be forwarded
<code>cache</code>	Enable object cache for Parquet files
<code>query</code>	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_fun %>%
  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(),
  ...,
  environment_scan = FALSE
)

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",
  array = "none"
)
```

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

Arguments

<code>dbdir</code>	Location for database files. Should be a path to an existing directory in the file system. With the default (or <code>""</code>), all data is kept in RAM.
<code>read_only</code>	Set to <code>TRUE</code> 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 <code>drv</code> object or a <code>drv</code> object pointing to the same path) will silently ignore this flag.
<code>bigint</code>	How 64-bit integers should be returned. There are two options: <code>"numeric"</code> and <code>"integer64"</code> . If <code>"numeric"</code> is selected, bigint integers will be treated as double/numeric. If <code>"integer64"</code> is selected, bigint integers will be set to bit64 encoding.
<code>config</code>	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.
<code>...</code>	Reserved for future extensions, must be empty.
<code>environment_scan</code>	Set to <code>TRUE</code> to treat data frames from the calling environment as tables. If a database table with the same name exists, it takes precedence. The default of this setting may change in a future version.
<code>drv</code>	Object returned by <code>duckdb()</code>
<code>debug</code>	Print additional debug information, such as queries.
<code>timezone_out</code>	The time zone returned to R, defaults to <code>"UTC"</code> , which is currently the only timezone supported by duckdb. If you want to display datetime values in the local timezone, set to <code>Sys.timezone()</code> or <code>""</code> .
<code>tz_out_convert</code>	How to convert timestamp columns to the timezone specified in <code>timezone_out</code> . There are two options: <code>"with"</code> , and <code>"force"</code> . If <code>"with"</code> is chosen, the timestamp will be returned as it would appear in the specified time zone. If <code>"force"</code> is chosen, the timestamp will have the same clock time as the timestamp in the database, but with the new time zone.
<code>array</code>	How arrays should be returned. There are two options: <code>"none"</code> and <code>"matrix"</code> . If <code>"none"</code> is selected, arrays are not returned. Instead an error is generated. If <code>"matrix"</code> is selected, arrays are returned as a column matrix. Each array is one row in the matrix.
<code>conn</code>	A <code>duckdb_connection</code> object
<code>shutdown</code>	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	<i>DuckDB EXPLAIN query tree</i>
----------------------	----------------------------------

Description

DuckDB EXPLAIN query tree

duckdb_read_csv	<i>Reads a CSV file into DuckDB</i>
-----------------	-------------------------------------

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

<code>conn</code>	A DuckDB connection, created by <code>dbConnect()</code> .
<code>name</code>	The name for the virtual table that is registered or unregistered
<code>files</code>	One or more CSV file names, should all have the same structure though
<code>...</code>	Reserved for future extensions, must be empty.
<code>header</code>	Whether or not the CSV files have a separate header in the first line
<code>na.strings</code>	Which strings in the CSV files should be considered to be NULL
<code>nrow.check</code>	How many rows should be read from the CSV file to figure out data types
<code>delim</code>	Which field separator should be used
<code>quote</code>	Which quote character is used for columns in the CSV file
<code>col.names</code>	Override the detected or generated column names
<code>col.types</code>	Character vector of column types in the same order as <code>col.names</code> , 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.
<code>lower.case.names</code>	Transform column names to lower case
<code>sep</code>	Alias for <code>delim</code> for compatibility
<code>transaction</code>	Should a transaction be used for the entire operation
<code>temporary</code>	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

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

<code>conn</code>	A DuckDB connection, created by <code>dbConnect()</code> .
<code>name</code>	The name for the virtual table that is registered or unregistered
<code>arrow_scannable</code>	A scannable Arrow-object
<code>use_async</code>	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

`adbcdrivermanager::adbc_driver()`, 4

`backend-duckdb`, 2

`dbConnect`, `duckdb_driver`-method
(`duckdb`), 4

`dbConnect__duckdb_driver` (`duckdb`), 4

`dbDisconnect`, `duckdb_connection`-method
(`duckdb`), 4

`dbDisconnect__duckdb_connection`
(`duckdb`), 4

`dplyr::tbl()`, 2, 3

`duckdb`, 4

`duckdb_adbc` (`duckdb`), 4

`duckdb_connection`, 6

`duckdb_driver`, 5

`duckdb_explain` (`duckdb_explain-class`), 6

`duckdb_explain-class`, 6

`duckdb_list_arrow`
(`duckdb_register_arrow`), 9

`duckdb_read_csv`, 6

`duckdb_register`, 8

`duckdb_register_arrow`, 9

`duckdb_shutdown` (`duckdb`), 4

`duckdb_unregister` (`duckdb_register`), 8

`duckdb_unregister_arrow`
(`duckdb_register_arrow`), 9

`print.duckdb_explain`
(`duckdb_explain-class`), 6

`simulate_duckdb` (`backend-duckdb`), 2

`Sys.timezone()`, 5

`tbl_file` (`backend-duckdb`), 2

`tbl_function` (`backend-duckdb`), 2

`tbl_query` (`backend-duckdb`), 2