Package ‘duckdf’

November 13, 2026

Type Package

Title SQL for Dataframes, but with Ducks

Version 0.2.1

Description An SQL interface for R dataframes built on the DuckDB embedded analytical database. Dataframes are registered directly with an in-process DuckDB instance and queried with standard SQL, so analyses can join, filter, aggregate and reshape R objects with SQL while results are returned as dataframes, without copying data to an external database server. The package wraps the 'duckdb' R package with a DBI-compliant connection and provides a small, checked surface for the common register-query-unnregister workflow. The underlying engine and its vectorised execution model are described in Raasveldt and Mühleisen (2019) <doi:10.1145/3299869.3320212>.

License AGPL-3

Encoding UTF-8

Depends R (>= 4.2.0)

Imports DBI (>= 1.1.3), checkmate, duckdb (>= 1.5.6), jsonlite

Suggests bench, data.table, dbplyr, dplyr, duckplyr, knitr, rmarkdown, sqldf, testthat (>= 3.0.0), tibble, withr

URL https://github.com/phillc73/duckdf

BugReports https://github.com/phillc73/duckdf/issues

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Phill Clarke [aut, cre]

Maintainer Phill Clarke <phill@mimosadev.org>

Repository CRAN

Date/Publication 2026-11-13 17:10:14 UTC

Contents

cleanup 2
duckdf 3
duckdf_views 4
gander 5
ingest 6
persist 7

Index **9**

`cleanup` *Clean up your duckdbs*

Description

`cleanup` removes all traces of the named duckdb database from disk and closes the shared in-memory connection.

Usage

```
cleanup(db_name)
```

Arguments

`db_name`	String. The specific name of the duckdb database to be removed. Required.

Value

Invisibly, the character vector of files removed. Missing files are not errors.

Examples

```
## Not run:

# Query the mtcars dataframe and write the whole table to an on-disk duckdb
duckdf("SELECT mpg, cyl FROM mtcars WHERE disp >= 200",
       persist = TRUE)

# Remove all traces of the on-disk duckdb database called mtcars
duckdf::cleanup("mtcars")


## End(Not run)
```

duckdf *Query your dataframes with SQL*

Description

duckdf runs a SQL query against R dataframes registered as zero-copy DuckDB views. Like SQL? Like dataframes? Like ducks?

Usage

```
duckdf(
  query,
  persist = FALSE,
  df_name = NULL,
  df = NULL,
  params = NULL,
  db_name = NULL,
  as.data.table = FALSE
)
```

Arguments

query	String. A single SQL statement (multi-statement queries are rejected whenever the json extension is available; in its absence the degraded path executes them last-wins). Required.
persist	Logical. If TRUE, delegate to persist() for an on-disk database (see that function).
df_name	Character vector of variable names holding dataframes - registered verbatim, the extractor is not invoked. This is also the escape hatch: multi-statement queries and DDL (CREATE TABLE ...) pass through raw. A table created this way stays on the shared connection, is not listed by duckdf_views() (registered views only) and is cleared only by cleanup().
df	Named list of dataframes - registered under their list names; those names are the extraction candidate scope. Preferred form inside functions: no caller-frame dependency. When the json extension is unavailable, the degraded path passes DDL and multi-statement queries through as df_name does.
params	Parameter list forwarded to DBI::dbGetQuery() untouched; use DuckDB ?/$1 placeholders in the query.
db_name	String. On-disk database name for the persist path. The extractor is not invoked when df_name is given, so multi-statement queries and DDL also pass through here.
as.data.table	Logical. Return a data.table instead of a plain data.frame.

Details

Bare queries (no df/df_name) use DuckDB's json extension to extract table names; machines where the extension cannot be autoloaded should pass df_name or df (df runs a once-per-session availability probe; neither form needs the extension for its work). Offline, the degraded path executes multi-statement queries last-wins and passes DDL through, mirroring df_name's documented passthrough semantics, and scope errors surface as engine errors at execution.

Value

The query result: data.frame by default, data.table when as.data.table = TRUE.

Examples

```
## Not run:

# Query the mtcars dataframe
duckdf("SELECT mpg, cyl FROM mtcars WHERE disp >= 200")

# Parameterised, function-safe form
cyl_query <- function(x) {
  duckdf("SELECT mpg FROM mtcars WHERE cyl = $1", df = list(mtcars = mtcars),
         params = list(x))
}

# Read a csv without registering anything - DuckDB reads it natively
duckdf("SELECT * FROM read_csv_auto('myfile.csv')")


## End(Not run)
```

---

duckdf_views *List your registered views*

Description

duckdf_views lists the views currently registered on the shared duckdf connection.

Usage

```
duckdf_views()
```

Value

A data.frame with one row per registered view: the view name and the class of the backing R object. Empty (not an error) when nothing is registered. Read-only: no unregistration happens here.

Examples

```
## Not run:

# See what duckdf is currently quacking about
duckdf::duckdf_views()


## End(Not run)
```

---

gander *Gander at duckdbs*

Description

gander returns some brief information about an on-disk duckdb database.

Usage

```
gander(db_name, show_types = FALSE)
```

Arguments

db_name	String. The specific name of the duckdb database to have a gander at. Required.
show_types	Logical. Add column type information to each element. Default FALSE.

Value

A named list, one element per table in the database, each holding the table's first five rows. Element names are the table names. With show_types = TRUE each element gains a "types" attribute naming the columns and their types.

Examples

```
## Not run:

# Query the mtcars dataframe and write the whole table to an on-disk duckdb
duckdf("SELECT mpg, cyl FROM mtcars WHERE disp >= 200",
       persist = TRUE)

# Gander at the tables in the on-disk duckdb database for mtcars,
# showing column types
duckdf::gander("mtcars", show_types = TRUE)


## End(Not run)
```

ingest *Ingest data*

Description

ingest loads a CSV file via duckdb_read_csv and returns either a new data.frame, data.table, tibble or an on-disk duckdb.

Usage

```
ingest(name, files, persist = TRUE, object_type = "data.frame", ...)
```

Arguments

name	String. The name of the new data.frame, data.table, tibble or on-disk duckdb. Required.
files	String. The name of the CSV file to ingest. File must be located in the current working directory. Required.
persist	Logical. If TRUE the CSV lands as table name in a new on-disk database file name; the duckdb_read_csv result is returned invisibly (a count; side-effect function). If FALSE the ingested data is returned as an object of object_type. Calling again with the same name on an existing database re-reads the CSV into the existing table: the rows add up, they do not replace - remove the database first (cleanup()) for a fresh ingest. Note that duckdb_read_csv itself leaves an empty file named duckdb in the working directory; that is duckdb-r behaviour.
object_type	String. The type of object to return when persist = FALSE: "data.frame" (default), "data.table" or "tibble".
...	All duckdb_read_csv options, passed through unchanged (header, na.strings, nrow.check, delim, quote, col.names, lower.case.names, sep, transaction).

Value

With persist = TRUE: the duckdb_read_csv result, invisibly. With persist = FALSE: the ingested data as an object of object_type. Callers capture: mtcars2 <- ingest("mtcars2", "mtcars.csv", persist = FALSE).

Examples

```
## Not run:

# Ingest the filename.csv file as an on-disk duckdb
# database named `descriptive_name`
duckdf::ingest(name = "descriptive_name", files = "filename.csv")

# Ingest the filename.csv file as a data.table, returned for capture
mtcars2 <- duckdf::ingest(name = "mtcars2", files = "filename.csv",
                          persist = FALSE, object_type = "data.table")
```

```
## End(Not run)
```

`persist` *Write persistent duckdb to disk*

Description

`persist` executes a query against an on-disk duckdb database in the current working directory, creating it (and writing the queried dataframes as tables) when it does not exist.

Usage

```
persist(
  query,
  db_name = NULL,
  df = NULL,
  df_name = NULL,
  params = NULL,
  as.data.table = FALSE
)
```

Arguments

`query`	String. A single SQL statement (multi-statement queries are rejected). The name of the on-disk database will be the same as the first named dataframe in this query string, unless specified differently in `db_name` parameter. Required.
`db_name`	String. Specifies the name of the on disk duckdb to read from or write to. If an on-disk duckdb matching this name is not found in the current working directory, one will be created (writing the df/df_name tables); if it exists, df and df_name are an error - the query runs against the data already on disk.
`df`	Named list of dataframes to write as tables when creating the database; those names are the extraction candidate scope.
`df_name`	Character vector of variable names in the calling scope to write as tables - registered verbatim, the extractor is not invoked, so multi-statement queries and DDL also pass through here.
`params`	Parameter list forwarded to `DBI::dbGetQuery()` untouched.
`as.data.table`	Logical. Return a data.table instead of a plain data.frame.

Details

Bare queries (no `df`/`df_name`) use DuckDB's json extension to extract table names; machines where the extension cannot be autoloaded should pass `df_name` or `df` together with `db_name`, which never touch the extractor.

Value

The results of the SQL statement: data.frame by default, data.table when `as.data.table = TRUE`. All tables are written before any query executes; the connection is closed on every exit path.

Examples

```
## Not run:

# Query the mtcars dataframe and write the whole table to an on-disk duckdb
duckdf::persist("SELECT mpg, cyl FROM mtcars WHERE disp >= 200")

# Same, with a different database name
duckdf::persist("SELECT mpg, cyl FROM mtcars WHERE disp >= 200",
                db_name = "mtcars_duckdb")


## End(Not run)
```

Index

cleanup, 2

duckdf, 3
duckdf_views, 4

gander, 5

ingest, 6

persist, 7