Package 'DBIsqldf'

October 12, 2022

Version 0.9.9-2

Date 2022-09-07

Title Manipulate R Data Frames Using SQL

Author Neal Fultz <nfultz@gmail.com> and G. Grothendieck <ggrothendieck@gmail.com>

Maintainer Neal Fultz <nfultz@gmail.com>

Description Helper function sqldf() transparently initializes a database, imports data frames into it, and executes a query or other statement. This supports all DBI drivers to some degree, and parameterized queries.

ByteCompile true

Imports DBI

Suggests RSQLite

License GPL-2

BugReports https://github.com/nfultz/DBIsqldf/issues

URL https://github.com/nfultz/DBIsqldf

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-09-09 08:13:04 UTC

R topics documented:

DBIsqldf-package 2
sqldf 2

Index **5**

DBIsqldf-package *DBIsqldf package overview*

Description

Provides an easy way to perform SQL selects on R data frames.

Details

The package contains a single function `sqldf()` whose help file contains more information and examples.

This package is a fork from the original sqldf-package. The primary differences are:

- DBI interface only. This increases compatibility with database drivers, but at the expense of database-specific convenience features.
- DBI native parameterized query support.
- More conservative defaults for writing tables - data frames are imported as temporary tables and should be committed explicitly is they need to be persisted.
- No legacy (transitive) dependencies.

sqldf *SQL select on data frames*

Description

sqldf is an R package for running SQL statements on R data frames, optimized for convenience. The user simply specifies an SQL statement in R using data frame names in place of table names and a database with appropriate table layouts/schema is automatically created, the data frames are automatically loaded into the database, the specified SQL statement is performed, the result is read back into R and the database is deleted all automatically behind the scenes making the database's existence transparent to the user who only specifies the SQL statement. Surprisingly this can at times be even faster than the corresponding pure R calculation (although the purpose of the project is convenience and not speed).

Usage

```
sqldf(statement, ..., conn, tx)
```

Arguments

`statement`	A SQL statement. Use ? for parameterized queries.
`...`	parameters to be interpolated into `statement`.
`conn`	optional, a DBIConnection object.
`tx`	optional, one of `'none'` (default), `'commit'`, or `'rollback'`.

Details

All DBI drivers are supported, although features may be limited by driver support. RSQLite and duckdb are recommended.

DBIsqldf is free software published under the GNU General Public License that can be downloaded from CRAN.

Options:

sqldf.driver: A DBIDriver (or cloneable DBIConnection) object for a transient database connection when conn is omitted, or an expression or function that yields one. The default is quote(RSQLite::SQLite)

FAQ:

1. How does DBIsqldf handle classes and factors?
   It doesn't. The exact behaviour will depend on the database you choose to use. This includes dates.
2. Why does DBIsqldf seem to mangle certain variable names?
   If a data frame contains variables with names that are not valid in SQL, the behavior is left up to the database driver you choose to use. Notably, periods can cause issues - you may need to double quote the affected variables in your SQL statement. Also please be aware that SQL is **not case sensitive**, while R is.
3. Why does sqldf("select var(x) from DF") not work?
   Functions are provided by the database you choose to use, not by R.
4. Why am I having problems with update?
   Be sure to set tx to commit to persist any changes you have made to the database.
5. Why do certain calculations come out as integer rather than double?
   Calculations performed within the database will follow the database's rules.

Value

A data frame of results if statement was a query, otherwise nothing.

Examples

```
if(is.null(getOption("sqldf.driver")) && !requireNamespace("RSQLite", quietly=TRUE)) {
 message("Please configure `options(sqldf.driver)` or install RSQLite to run below examples.")
} else {

data(iris, envir=environment())

# head
sqldf("select * from iris limit 5")

# Filter
sqldf("select * from iris where species = 'virginica' limit 5 ")

# Parameterized query
sqldf("select * from iris where species = ? limit 5", "versicolor")
```

```
# Aggregate, quoting
sqldf('select species, avg("Petal.Width") from iris group by 1 limit 5')

# Compare with aggregate(Petal.Width~Species, iris, FUN = var)
# CTE, back join
sqldf('
  with tbl_width as (
      select species, avg("Petal.Width") as xbar, count(1) as n
      from iris
      group by 1
  )
  select Species,
  1.0/(n-1) * sum(("Petal.Width" - xbar)*("Petal.Width" - xbar)) as var
  from iris
  join tbl_width using (species)
  group by 1
')

}

if(!requireNamespace("RSQLite", quietly=TRUE)) {
  message("Below examples require RSQLite specifically.")
} else {

# Manually managing the DB connection, and writing to a database
conn <- DBI::dbConnect(RSQLite::SQLite(), tempfile())

data(mtcars, envir=environment())

sqldf("create table usacars as select * from mtcars where am = 1", conn=conn, tx='commit')

#Now compare
DBI::dbListTables(conn)

# vs persisted tables
sqldf("SELECT type, name FROM sqlite_schema", conn=conn)

# hang up and reconnect
DBI::dbDisconnect(conn)
conn <- DBI::dbConnect(conn)

DBI::dbListTables(conn)

sqldf("SELECT cyl, count(1) as n from usacars group by 1", conn=conn)

DBI::dbDisconnect(conn)

}
```

Index

DBIConnection, *2*, *3*
DBIDriver, *3*
DBIsqldf-package, 2
duckdb, *3*

RSQLite, *3*

sqldf, 2
sqldf(), *2*