Package 'sqldf'

January 30, 2026

Version 0.4-12

Date 2026-01-29

Title Manipulate R Data Frames Using SQL

Description The sqldf() function is typically passed a single argument which is an SQL select statement where the table names are ordinary R data frame names. sqldf() transparently sets up a database, imports the data frames into that database, performs the SQL select or other statement and returns the result using a heuristic to determine which class to assign to each column of the returned data frame. The sqldf() or read.csv.sql() functions can also be used to read filtered files into R even if the original files are larger than R itself can handle. 'RSQLite', 'RH2', 'RMySQL' and 'RPostgreSQL' backends are supported.

ByteCompile true

Depends R (>= 3.1.0), gsubfn (>= 0.6), proto, RSQLite

Suggests RH2, RMySQL, RPostgreSQL, svUnit, tcltk, MASS

Imports DBI, chron

License GPL-2

BugReports https://github.com/ggrothendieck/sqldf/issues

URL https://github.com/ggrothendieck/sqldf

NeedsCompilation no

Author G. Grothendieck [aut, cre]

Maintainer G. Grothendieck <ggrothendieck@gmail.com>

Repository CRAN

Date/Publication 2026-01-30 06:10:09 UTC

Contents

sqldf-package *sqldf 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.

References

The sqldf help page contains the primary documentation. The sqldf github home page https://github.com/ggrothendieck/sqldf contains links to SQLite pages that may be helpful in formulating queries.

read.csv.sql *Read File Filtered by SQL*

Description

Read a file into R filtering it with an sql statement. Only the filtered portion is processed by R so that files larger than R can otherwise handle can be accommodated.

Usage

```
read.csv.sql(file, sql = "select * from file", header = TRUE, sep = ",",
row.names, eol, skip, filter, nrows, field.types,
colClasses, dbname = tempfile(), drv = "SQLite", ...)
read.csv2.sql(file, sql = "select * from file", header = TRUE, sep = ";",
row.names, eol, skip, filter, nrows, field.types,
colClasses, dbname = tempfile(), drv = "SQLite", ...)
```

Arguments

file	A file path or a URL (beginning with http:// or ftp://). If the filter argument is used and no file is to be input to the filter then file can be omitted, NULL, NA or "".
sql	character string holding an SQL statement. The table representing the file should be referred to as file.
header	As in read.csv.
sep	As in read.csv.

row.names	As in read.csv.
eol	Character which ends line.
skip	Skip indicated number of lines in input file.
filter	If specified, this should be a shell/batch command that the input file is piped through. For read.csv2.sql it is by default the following on non-Windows systems: tr , .. This translates all commas in the file to dots. On Windows similar functionalty is provided but to do that using a vbscript file that is included with sqldf to emulate the tr command.
nrows	Number of rows used to determine column types. It defaults to 50. Using -1 causes it to use all rows for determining column types. This argument is rarely needed.
field.types	A list whose names are the column names and whose contents are the SQLite types (not the R class names) of the columns. Specifying these types improves how fast it takes. Unless speed is very important this argument is not normally used.
colClasses	As in read.csv.
dbname	As in sqldf except that the default is tempfile(). Specifying NULL will put the database in memory which may improve speed but will limit the size of the database by the available memory.
drv	This argument is ignored. Currently the only database SQLite supported by read.csv.sql and read.csv2.sql is SQLite. Note that the H2 database has a builtin SQL function, CSVREAD, which can be used in place of read.csv.sql.
...	Passed to sqldf.

Details

Reads the indicated file into an sql database creating the database if it does not already exist. Then it applies the sql statement returning the result as a data frame. If the database did not exist prior to this statement it is removed.

Note that it uses facilities of SQLite to read the file which are intended for speed and therefore not as flexible as in R. For example, it does not recognize quoted fields as special but will regard the quotes as part of the field. See the sqldf help for more information.

read.csv2.sql is like read.csv.sql except the default sep is ";" and the default filter translates all commas in the file to decimal points (i.e. to dots).

On Windows, if the filter argument is used and if Rtools is detected in the registry then the Rtools bin directory is added to the search path facilitating use of those tools without explicitly setting any the path.

Value

If the sql statement is a select statement then a data frame is returned.

Examples

```
## Not run:
# might need to specify eol= too depending on your system
write.csv(iris, "iris.csv", quote = FALSE, row.names = FALSE)
iris2 <- read.csv.sql("iris.csv",
sql = "select * from file where Species = 'setosa' ")


## End(Not run)
```

---

sqldf *SQL select on data frames*

Description

SQL select on data frames

Usage

```
sqldf(x, stringsAsFactors = FALSE,
   row.names = FALSE, envir = parent.frame(),
   method = getOption("sqldf.method"),
   file.format = list(), dbname, drv = getOption("sqldf.driver"),
   user, password = "", host = "localhost", port,
   dll = getOption("sqldf.dll"), connection = getOption("sqldf.connection"),
   verbose = isTRUE(getOption("sqldf.verbose")))
```

Arguments

x	Character string representing an SQL select statement or character vector whose components each represent a successive SQL statement to be executed. The select statement syntax must conform to the particular database being used. If x is missing then it establishes a connection which subsequent sqldf statements access. In that case the database is not destroyed until the next sqldf statement with no x.
stringsAsFactors	If TRUE then those columns output from the database as "character" are converted to "factor" if the heuristic is unable to determine the class.
row.names	For TRUE the tables in the data base are given a row_names column filled with the row names of the corresponding data frames. Note that in SQLite a special rowid (or equivalently oid or _rowid_) is available in any case.
envir	The environment where the data frames representing the tables are to be found.
method	This argument is a list of two functions, keywords or character vectors. If the second component of the list is NULL (the default) then the first component of the list can be specified without wrapping it in a list. The first component specifies

a transformation of the data frame output from the database and the second specifies a transformation to each data frame that is passed to the data base just before it is read into the database The second component is less frequently used. If the first component is NULL or not specified that it defaults to "auto". If the second component is NULL or not specified then no transformation is performed on the input.

The allowable keywords for the first components are (1) "auto" which is the default and automatically assigns the class of each column using the heuristic described later, (2) "auto.factor" which is the same as "auto" but does not assign "factor" and "ordered" classes, (3) "raw" or NULL which means use whatever classes are returned by the database with no automatic processing and (4) "name__class" which means that columns names that end in __class with two underscores where class is an R class (such as Date) are converted to that class and the __class portion is removed from the column name. For example, sqldf("select a as x__Date from DF", method = "name__class") would cause column a to be coerced to class Date and have the column name x. The first component of method can also be a character vector of classes to assign to the returned data.frame. The example just given could alternately be implemented using sqldf("select a as x from DF", method = "Date") Note that when Date is used in this way it assumes the database contains the number of days since January 1, 1970. If the date is in the format yyyy-mm-dd then use Date2 as the class.

file.format A list whose components are passed to sqliteImportFile. Components may include sep, header, row.names, skip, eol and filter. Except for filter they are passed to sqliteImportFile and have the same default values as in sqliteImportFile (except for eol which defaults to the end of line character(s) for the operating system in use – note that if the file being read does not have the line endings for the platform being used then eol will have to be specified. In particular, certain UNIX-like tools on Windows may produce files with UNIX line endings in which case eol="\n" should be specified). filter may optionally contain a batch/shell command through which the input file is piped prior to reading it in. Alternately filter may be a list whose first component is a batch/shell command containing names which correspond to the names of the subsequent list components. These subsequent components should each be a character vector which sqldf will read into a temporary file. The name of the temporary file will be replaced into the command. For example, filter = list("gawk -f prog", prog = '{ print gensub(/,/, ".", "g") }') . command line quoting which may vary among shells and Windows. Note that if the filter produces files with UNIX line endings on Windows then eol must be specified, as discussed above. file.format may be set to NULL in order not to search for input file objects at all. The file.format can also be specified as an attribute in each file object itself in which case such specification overrides any given through the argument list. There is further discussion of file.format below.

dbname Name of the database. For SQLite and h2 data bases this defaults to ":memory:" which results in an embedded database. For MySQL this defaults to getOption("RMysql.dbname") and if that is not specified then "test" is used For RPostgreSQL this defaults to getOption("sqldf.RPostgreSQL.dbname") and if that is not specified then

"test" is used.

drv "SQLite", "MySQL", "h2", "PostgreSQL" or "pgSQL" or any of those names prefaced with "R". If not specified then the "dbDriver" option is checked and if that is not set then sqldf checks whether RPostgreSQL, RMySQL or RH2 is loaded in that order and the driver corresponding to the first one found is used. If none are loaded then "SQLite" is used. dbname=NULL causes the default to be used.

user user name. Not needed for embedded databases. For RPostgreSQL the default is taken from option sqldf.RPostgreSQL.user and if that is not specified either then "postgres" is used.

password password. Not needed for embedded databases. For RPostgreSQL the default is taken from option sqldf.RPostgreSQL.password and if that is not specified then "postgres" is used.

host host. Default of "localhost" is normally sufficient. For RPostgreSQL the default is taken from option sqldf.RPostgreSQL.host and if that is not specified then "test" is used.

port port. For RPostgreSQL the default is taken from the option sqldf.RPostgreSQL.port and if that is not specified then 5432 is used.

dll Name of an SQLite loadable extension to automatically load. If found on PATH then it is automatically loaded and the SQLite functions it in will be accessible.

connection If this is NULL then a connection is created; otherwise the indicated connection is used. The default is the value of the option sqldf.connection. If neither connection nor sqldf.connection are specified a connection is automatically generated on-the-fly and closed on exit of the call to sqldf. If this argument is not NULL then the specified connection is left open on termination of the sqldf call. Usually this argument is left unspecified. It can be used to make repeated calls to a database without reloading it.

verbose If TRUE then verboe output shown. Anything else suppresses verbose output. Can be set globally using option "sqldf.verbose".

Details

The typical action of sqldf is to

create a database in memory

read in the data frames and files used in the select statement. This is done by scanning the select statement to see which words in the select statement are of class "data.frame" or "file" in the parent frame, or the specified environment if envir is used, and for each object found by reading it into the database if it is a data frame. Note that this heuristic usually reads in the wanted data frames and files but on occasion may harmlessly reads in extra ones too.

run the select statement getting the result as a data frame

assign the classes of the returned data frame's columns if method = "auto". This is done by checking all the column names in the read-in data frames and if any are the same as a column output from the data base then that column is coerced to the class of the column whose name matched. If the class of the column is "factor" or "ordered" or if the column is not matched

then the column is returned as is. If method = "auto.factor" then processing is similar except that "factor" and "ordered" classes and their levels will be assigned as well. The "auto.factor" heuristic is less reliable than the "auto" heuristic. If method = "raw" then the classes are returned as is from the database.

cleanup If the database was created by sqldf then it is deleted; otherwise, all tables that were created are dropped in order to leave the database in the same state that it was before. The database connection is terminated.

sqldf supports the following R options for RPostgreSQL: "sqldf.RPostgreSQL.dbname", "sqldf.RPostgreSQL.user", "sqldf.RPostgreSQL.password", "sqldf.RPostgreSQL.host" and "sqldf.RPostgreSQL.port" which have defaults "test", "postgres", "postgres", "localhost" and 5432, respectively. It also supports "sqldf.RPostgreSQL.other" which is a list of named parameters. These may include dbname, user, password, host and port. Individually these take precdence over otherwise specified connection arguments.

Warning. Although sqldf is usually used with on-the-fly databases which it automatically sets up and destroys if you wish to use it with existing databases be sure to back up your database prior to using it since incorrect operation could destroy the entire database.

Value

The result of the specified select statement is output as a data frame. If a vector of sql statements is given as x then the result of the last one is returned. If the x and connection arguments are missing then it returns a new connection and also places this connection in the option sqldf.connection.

Note

If row.names = TRUE is used then any NATURAL JOIN will make use of it which may not be what was intended.

3/2 and 3.0/2 are the same in R but in SQLite the first one causes integer arithmetic to be used whereas the second using floating point. Thus both evaluate to 1.5 in R but they evaluate to 1 and 1.5 respectively in SQLite.

The dbWriteTable/sqliteImportFile routines that sqldf uses to transfer files to the data base are intended for speed and they are not as flexible as read.table. Also they have slightly different defaults. (If more flexible input is needed use the slower read.table to read the data into a data frame instead of reading directly from a file.) The default for sep is sep = ",". If the first row of the file has one fewer entry than subsequent ones then it is assumed that header <- row.names <- TRUE and otherwise that header <- row.names <- FALSE. The header can be forced to header <- TRUE by specifying file.format = list(header = TRUE) as an argument to sqldf. sep and row.names are other file.format subarguments. Also, one limitation with .csv files is that quotes are not regarded as special within files so a comma within a data field such as "Smith, James" would be regarded as a field delimiter and the quotes would be entered as part of the data which probably is not what is intended.

Typically the SQL result will have the same data as the analogous non-database R code manipulations using data frames but may differ in row names and other attributes. In the examples below we use identical in those cases where the two results are the same in all respects or set the row names to NULL if they would have otherwise differed only in row names or use all.equal if the data portion is the same but attributes aside from row names differ.

On MySQL the database must pre-exist. Create a `c:\my.ini` or `%MYSQL_HOME%\my.ini` file on Windows or a `/etc/my.cnf` file on UNIX to contain information about the database. This file may specify the username, password and port. The password can be omitted if one has not been set. If using a standard port setup then the `port` can be omitted as well. The database is taken from the `dbname` argument of the `sqldf` command or if not set from `getOption("sqldf.dbname")` or if that option is not set it is assumed to be `"test"`. Note that MySQL does not use the `user`, `password`, `host`, `port` arguments of sqldf.

If `getOption("sqldf.dll")` is specified then the named dll will be loaded as an SQLite loadable extension. This is in addition to the extension functions included with RSQLite.

References

The sqldf home page https://github.com/ggrothendieck/sqldf contains more examples as well as links to SQLite pages that may be helpful in formulating queries. It also containers pointers to using sqldf with H2 and PostgreSQL.

Examples

```
#
# These examples show how to run a variety of data frame manipulations
# in R without SQL and then again with SQL
#

# head
a1r <- head(warpbreaks)
a1s <- sqldf("select * from warpbreaks limit 6")
identical(a1r, a1s)

# subset

a2r <- subset(CO2, grepl("^Qn", Plant))
a2s <- sqldf("select * from CO2 where Plant like 'Qn%'")
all.equal(as.data.frame(a2r), a2s)

data(farms, package = "MASS")
a3r <- subset(farms, Manag %in% c("BF", "HF"))
a3s <- sqldf("select * from farms where Manag in ('BF', 'HF')")
row.names(a3r) <- NULL
identical(a3r, a3s)

a4r <- subset(warpbreaks, breaks >= 20 & breaks <= 30)
a4s <- sqldf("select * from warpbreaks where breaks between 20 and 30",
   row.names = TRUE)
identical(a4r, a4s)

a5r <- subset(farms, Mois == 'M1')
a5s <- sqldf("select * from farms where Mois = 'M1'", row.names = TRUE)
identical(a5r, a5s)

a6r <- subset(farms, Mois == 'M2')
a6s <- sqldf("select * from farms where Mois = 'M2'", row.names = TRUE)
```

```
identical(a6r, a6s)

# rbind
a7r <- rbind(a5r, a6r)
a7s <- sqldf("select * from a5s union all select * from a6s")

# sqldf drops the unused levels of Mois but rbind does not; however,
# all data is the same and the other columns are identical
row.names(a7r) <- NULL
identical(a7r[-1], a7s[-1])

# aggregate - avg conc and uptake by Plant and Type
a8r <- aggregate(iris[1:2], iris[5], mean)
a8s <- sqldf('select Species, avg("Sepal.Length") `Sepal.Length`,
   avg("Sepal.Width") `Sepal.Width` from iris group by Species')
all.equal(a8r, a8s)

# by - avg conc and total uptake by Plant and Type
a9r <- do.call(rbind, by(iris, iris[5], function(x) with(x,
data.frame(Species = Species[1],
mean.Sepal.Length = mean(Sepal.Length),
mean.Sepal.Width = mean(Sepal.Width),
mean.Sepal.ratio = mean(Sepal.Length/Sepal.Width)))))
row.names(a9r) <- NULL
a9s <- sqldf('select Species, avg("Sepal.Length") `mean.Sepal.Length`,
avg("Sepal.Width") `mean.Sepal.Width`,
avg("Sepal.Length"/"Sepal.Width") `mean.Sepal.ratio` from iris
group by Species')
all.equal(a9r, a9s)

# head - top 3 breaks
a10r <- head(warpbreaks[order(warpbreaks$breaks, decreasing = TRUE), ], 3)
a10s <- sqldf("select * from warpbreaks order by breaks desc limit 3")
row.names(a10r) <- NULL
identical(a10r, a10s)

# head - bottom 3 breaks
a11r <- head(warpbreaks[order(warpbreaks$breaks), ], 3)
a11s <- sqldf("select * from warpbreaks order by breaks limit 3")
# attributes(a11r) <- attributes(a11s) <- NULL
row.names(a11r) <- NULL
identical(a11r, a11s)

# ave - rows for which v exceeds its group average where g is group
DF <- data.frame(g = rep(1:2, each = 5), t = rep(1:5, 2), v = 1:10)
a12r <- subset(DF, v > ave(v, g, FUN = mean))
Gavg <- sqldf("select g, avg(v) as avg_v from DF group by g")
a12s <- sqldf("select DF.g, t, v from DF, Gavg where DF.g = Gavg.g and v > avg_v")
row.names(a12r) <- NULL
identical(a12r, a12s)

# same but reduce the two select statements to one using a subquery
a13s <- sqldf("select g, t, v
```

```
from DF d1, (select g as g2, avg(v) as avg_v from DF group by g)
where d1.g = g2 and v > avg_v")
identical(a12r, a13s)

# same but shorten using natural join
a14s <- sqldf("select g, t, v
from DF
natural join (select g, avg(v) as avg_v from DF group by g)
where v > avg_v")
identical(a12r, a14s)

# table
a15r <- table(warpbreaks$tension, warpbreaks$wool)
a15s <- sqldf("select sum(wool = 'A'), sum(wool = 'B')
   from warpbreaks group by tension")
all.equal(as.data.frame.matrix(a15r), a15s, check.attributes = FALSE)

# reshape
t.names <- paste("t", unique(as.character(DF$t)), sep = "_")
a16r <- reshape(DF, direction = "wide", timevar = "t", idvar = "g", varying = list(t.names))
a16s <- sqldf("select
g,
sum((t == 1) * v) t_1,
sum((t == 2) * v) t_2,
sum((t == 3) * v) t_3,
sum((t == 4) * v) t_4,
sum((t == 5) * v) t_5
from DF group by g")
all.equal(a16r, a16s, check.attributes = FALSE)

# order
a17r <- Formaldehyde[order(Formaldehyde$optden, decreasing = TRUE), ]
a17s <- sqldf("select * from Formaldehyde order by optden desc")
row.names(a17r) <- NULL
identical(a17r, a17s)

# centered moving average of length 7
set.seed(1)
DF <- data.frame(x = rnorm(15, 1:15))
s18 <- sqldf("select a.x x, avg(b.x) movavgx from DF a, DF b
   where a.row_names - b.row_names between -3 and 3
   group by a.row_names having count(*) = 7
   order by a.row_names+0",
 row.names = TRUE)
r18 <- data.frame(x = DF[4:12,], movavgx = rowMeans(embed(DF$x, 7)))
row.names(r18) <- NULL
all.equal(r18, s18)

# merge.  a19r and a19s are same except row order and row names
A <- data.frame(a1 = c(1, 2, 1), a2 = c(2, 3, 3), a3 = c(3, 1, 2))
B <- data.frame(b1 = 1:2, b2 = 2:1)
a19s <- sqldf("select * from A, B")
a19r <- merge(A, B)
```

```
Sort <- function(DF) DF[do.call(order, DF),]
all.equal(Sort(a19s), Sort(a19r), check.attributes = FALSE)

# within Date, of the highest quality records list the one closest
# to noon.  Note use of two sql statements in one call to sqldf.

Lines <- "DeployID Date.Time LocationQuality Latitude Longitude
STM05-1 2005/02/28 17:35 Good -35.562 177.158
STM05-1 2005/02/28 19:44 Good -35.487 177.129
STM05-1 2005/02/28 23:01 Unknown -35.399 177.064
STM05-1 2005/03/01 07:28 Unknown -34.978 177.268
STM05-1 2005/03/01 18:06 Poor -34.799 177.027
STM05-1 2005/03/01 18:47 Poor -34.85 177.059
STM05-2 2005/02/28 12:49 Good -35.928 177.328
STM05-2 2005/02/28 21:23 Poor -35.926 177.314
"

DF <- read.table(textConnection(Lines), skip = 1,  as.is = TRUE,
 col.names = c("Id", "Date", "Time", "Quality", "Lat", "Long"))

sqldf(c("create temp table DFo as select * from DF order by
  Date DESC, Quality DESC,
  abs(substr(Time, 1, 2) + substr(Time, 4, 2) /60 - 12) DESC",
  "select * from DFo group by Date"))

## Not run:

# test of file connections with sqldf

# create test .csv file of just 3 records
write.table(head(iris, 3), "iris3.dat", sep = ",", quote = FALSE)

# look at contents of iris3.dat
readLines("iris3.dat")

# set up file connection
iris3 <- file("iris3.dat")
sqldf('select * from iris3 where "Sepal.Width" > 3')

# using a non-default separator
# file.format can be an attribute of file object or an arg passed to sqldf
write.table(head(iris, 3), "iris3.dat", sep = ";", quote = FALSE)
iris3 <- file("iris3.dat")
sqldf('select * from iris3 where "Sepal.Width" > 3', file.format = list(sep = ";"))

# same but pass file.format through attribute of file object
attr(iris3, "file.format") <- list(sep = ";")
sqldf('select * from iris3 where "Sepal.Width" > 3')

# copy file straight to disk without going through R
# and then retrieve portion into R
sqldf('select * from iris3 where "Sepal.Width" > 3', dbname = tempfile())
```

```
### same as previous example except it allows multiple queries against
### the database.  We use iris3 from before.  This time we use an
### in memory SQLite database.

sqldf() # open a connection
sqldf('select * from iris3 where "Sepal.Width" > 3')

# At this point we have an iris3 variable in both
# the R workspace and in the SQLite database so we need to
# explicitly let it know we want the version in the database.
# If we were not to do that it would try to use the R version
# by default and fail since sqldf would prevent it from
# overwriting the version already in the database to protect
# the user from inadvertent errors.
sqldf('select * from main.iris3 where "Sepal.Width" > 4')
sqldf('select * from main.iris3 where "Sepal_Width" < 4')
sqldf() # close connection

### another way to do this is a mix of sqldf and RSQLite statements
### In that case we need to fetch the connection for use with RSQLite
### and do not have to specifically refer to main since RSQLite can
### only access the database.

con <- sqldf()
# this iris3 refers to the R variable and file
sqldf('select * from iris3 where "Sepal.Width" > 3')
sqldf("select count(*) from iris3")
# these iris3 refer to the database table
dbGetQuery(con, 'select * from iris3 where "Sepal.Width" > 4')
dbGetQuery(con, 'select * from iris3 where "Sepal.Width" < 4')
sqldf()


## End(Not run)
```

Index