Package 'SQLDataFrame'

September 30, 2020

Title Representation of SQL database in DataFrame metaphor

Version 1.2.0

Description SQLDataFrame is developed to lazily represent and efficiently analyze SQL-based tables in _R_. SQLDataFrame supports common and familiar 'DataFrame' operations such as '[' subsetting, rbind, cbind, etc.. The internal implementation is based on the widely adopted dplyr grammar and SQL commands. In-memory datasets or plain text files (.txt, .csv, etc.) could also be easily converted into SQLDataFrames objects (which generates a new database on-disk).

biocViews Infrastructure, DataRepresentation

URL https://github.com/Bioconductor/SQLDataFrame

BugReports https://github.com/Bioconductor/SQLDataFrame/issues

Depends R (>= 3.6), dplyr (>= 0.8.0.1), dbplyr (>= 1.4.0), S4Vectors

Imports DBI, lazyeval, methods, tools, stats, BiocGenerics, RSQLite, tibble

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.0

Suggests RMySQL, bigrquery, testthat, knitr, rmarkdown, DelayedArray

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/SQLDataFrame

git_branch RELEASE_3_11

git_last_commit c10c571

git_last_commit_date 2020-04-27

Date/Publication 2020-09-29

Author Qian Liu [aut, cre] (<https://orcid.org/0000-0003-1456-5099>), Martin Morgan [aut]

Maintainer Qian Liu <qliu7@buffalo.edu>

R topics documented:

`left_join` *join* `SQLDataFrame` *together*

Description

*_join functions for `SQLDataFrame` objects. Will preserve the duplicate rows for the input argument 'x'.

Usage

```
## S3 method for class 'SQLDataFrame'
left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'SQLDataFrame'
inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'SQLDataFrame'
semi_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)

## S3 method for class 'SQLDataFrame'
anti_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)
```

Arguments

`x`	`SQLDataFrame` objects to join.
`y`	`SQLDataFrame` objects to join.
`by`	A character vector of variables to join by. If 'NULL', the default, '*_join()' will do a natural join, using all variables with common names across the two tables. See `?dplyr::join` for details.
`copy`	Only kept for S3 generic/method consistency. Used as "copy = FALSE" internally and not modifiable.
`suffix`	A character vector of length 2 specify the suffixes to be added if there are non-joined duplicate variables in 'x' and 'y'. Default values are ".x" and ".y".See `?dplyr::join` for details.
`...`	Other arguments passed on to `*_join` methods. `localConn` can be passed here for one MySQL connection with write permission. Will be used only when `join`-ing two SQLDataFrame objects from different MySQL connections (to different MySQL databases), and neither has write permission. The situation is rare and should be avoided. See Details.

Details

The *_join functions support aggregation of SQLDataFrame objects from same or different connection (e.g., cross databases), either with or without write permission.

SQLite database tables are supported by SQLDataFrame package, in the same/cross-database aggregation, and saving.

For MySQL databases, There are different situations:

When the input SQLDataFrame objects connects to same remote MySQL database without write permission (e.g., ensembl), the functions work like dbplyr with the lazy operations and a DataFrame interface. Note that the aggregated SQLDataFrame can not be saved using saveSQLDataFrame.

When the input SQLDataFrame objects connects to different MySQL databases, and neither has write permission, the *_join functions are supported but will be quite time consuming. To avoid this situation, a more efficient way is to save the database table in local MySQL server using saveSQLDataFrame, and then call the *_join functions again.

More frequent situation will be the *_join operation on two SQLDataFrame objects, of which at least one has write permission. Then the cross-database aggregation through SQLDataFrame package will be supported by generating federated table from the non-writable connection in the writable connection. Look for MySQL database manual for more details.

Value

A SQLDataFrame object.

Examples

```
test.db1 <- system.file("extdata/test.db", package = "SQLDataFrame")
test.db2 <- system.file("extdata/test1.db", package = "SQLDataFrame")
con1 <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db1)
con2 <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db2)
obj1 <- SQLDataFrame(conn = con1,
                     dbtable = "state",
                     dbkey = c("region", "population"))
obj2 <- SQLDataFrame(conn = con2,
                     dbtable = "state1",
                     dbkey = c("region", "population"))

obj1_sub <- obj1[1:10, 1:2]
obj2_sub <- obj2[8:15, 2:3]

left_join(obj1_sub, obj2_sub)
inner_join(obj1_sub, obj2_sub)
semi_join(obj1_sub, obj2_sub)
anti_join(obj1_sub, obj2_sub)
```

---

makeSQLDataFrame *Construct SQLDataFrame from file.*

Description

Given a file name, makeSQLDataFrame will write the file contents into SQL database, and open the database table as SQLDataFrame.

Usage

```
makeSQLDataFrame(
  filename,
  dbtable = NULL,
  dbkey = character(),
  conn,
  host,
  user,
  dbname = NULL,
  password = NULL,
  type = c("SQLite", "MySQL"),
  overwrite = FALSE,
  sep = ",",
  index = FALSE,
  ...
)
```

Arguments

filename	A data.frame or DataFrame object, or a character string of the filepath to the text file that to be saved as SQL database table. For filepath, the data columns should not be quoted on disk.
dbtable	A character string for the to be saved database table name. If not provided, will use the name of the input data.frame or DataFrame object, or the basename(filename) without extension if filename is a character string.
dbkey	A character vector of column name(s) that could uniquely identify each row of the filename. Must be provided in order to construct a SQLDataFrame.
conn	a valid DBIConnection from SQLite or MySQL. If provided, arguments of 'user', 'host', 'dbname', 'password' will be ignored.
host	host name for SQL database.
user	user name for SQL database.
dbname	database name for SQL connection. For SQLite connection, it uses a tempfile(fileext = ".db") if not provided.
password	password for SQL database connection.
type	The SQL database type, supports "SQLite" and "MySQL".
overwrite	Whether to overwrite the dbtable if already exists. Default is FALSE.
sep	a character string to separate the terms. Not 'NA_character_'. Default is ,.
index	Whether to create an index table. Default is FALSE.
...	additional arguments to be passed.

Details

The provided file must has one or more columns to unique identify each row (no duplicate rows allowed). The file must be rectangular without rownames. (if rownames are needed, save it as a column.)

Value

A SQLDataFrame object.

Examples

```
mtc <- tibble::rownames_to_column(mtcars)

## data.frame input
obj <- makeSQLDataFrame(mtc, dbkey = "rowname")
obj

## character input
filename <- file.path(tempdir(), "mtc.csv")
write.csv(mtc, file= filename, row.names = FALSE, quote = FALSE)
obj <- makeSQLDataFrame(filename, dbkey = "rowname")
obj

## save as MySQL database
## Not run:
localConn <- DBI::dbConnect(dbDriver("MySQL"),
                            host = "",
                            user = "",
                            password = "",
                            dbname = "")
makeSQLDataFrame(filename, dbtable = "mtcMysql", dbkey = "rowname", conn = localConn)

## End(Not run)
```

---

rbind *rbind of* SQLDataFrame *objects*

Description

Performs rbind on SQLDataFrame objects.

Usage

```
## S4 method for signature 'SQLDataFrame'
rbind(..., deparse.level = 1)
```

Arguments

...	One or more SQLDataFrame objects. These can be given as named arguments.
deparse.level	See '?base::cbind' for a description of this argument.

Details

rbind supports aggregation of SQLDataFrame objects. For representation of SQLite tables, same or different connections are supported. For representation of MySQL tables, at least one SQLDataFrame must have write permission.

Value

A SQLDataFrame object.

Examples

```
test.db1 <- system.file("extdata/test.db", package = "SQLDataFrame")
test.db2 <- system.file("extdata/test1.db", package = "SQLDataFrame")
con1 <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db1)
con2 <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db2)
obj1 <- SQLDataFrame(conn = con1,
                     dbtable = "state",
                     dbkey = c("region", "population"))
obj2 <- SQLDataFrame(conn = con2,
                     dbtable = "state1",
                     dbkey = c("region", "population"))
obj1_sub <- obj1[1:10, 2:3]
obj2_sub <- obj2[8:15,2:3]

## rbind
res_rbind <- rbind(obj1_sub, obj2_sub)
res_rbind
dim(res_rbind)
```

`saveSQLDataFrame` *Save SQLDataFrame object as a new database table.*

Description

The function to save `SQLDataFrame` object as a database table with a supplied path to database. It also returns a `SQLDataFrame` object constructed from the user-supplied `dbname`, `dbtable`, and `dbkey`.

Usage

```
saveSQLDataFrame(
  x,
  dbname = tempfile(fileext = ".db"),
  dbtable = deparse(substitute(x)),
  localConn = connSQLDataFrame(x),
  overwrite = FALSE,
  index = TRUE,
  ...
)
```

Arguments

`x`	The `SQLDataFrame` object to be saved.
`dbname`	A character string of the file path of to be saved SQLite database file. Will only be used when the input SQLDataFrame represents a SQLite database table. Default to save in a temporary file.
`dbtable`	A character string for the to be saved database table name. Default is the name of the input `SQLDataFrame`.
`localConn`	A MySQL connection with write permission. Will be used only when the input SQLDataFrame objects connects to MySQL connections without write permission. A new MySQL table will be written in the the database this argument provides.

overwrite	Whether to overwrite the dbtable if already exists. Only applies to the saving of SQLDataFrame objects representing SQLite database tables. Default is FALSE.
index	Whether to create the database index. Default is TRUE.
...	other parameters passed to methods.

Details

For SQLDataFrame from union or rbind, if representation of MySQL tables, the data will be sorted by key columns, and saved as MySQL table in the connection with write permission.

Value

A SQLDataFrame object.

Examples

```
test.db <- system.file("extdata/test.db", package = "SQLDataFrame")
conn <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db)
obj <- SQLDataFrame(conn = conn, dbtable = "state", dbkey = "state")
obj1 <- obj[1:10, 2:3]
obj1 <- saveSQLDataFrame(obj1, dbtable = "obj_subset")
connSQLDataFrame(obj1)
dbtable(obj1)
```

SQLDataFrame *SQLDataFrame class*

Description

SQLDataFrame constructor, slot getters, show method and coercion methods to DataFrame and data.frame objects.

Usage

```
SQLDataFrame(
  conn,
  host,
  user,
  dbname,
  password = NULL,
  billing = character(0),
  type = c("SQLite", "MySQL", "BigQuery"),
  dbtable = character(0),
  dbkey = character(0),
  col.names = NULL
)

## S4 method for signature 'SQLDataFrame'
tblData(x)

## S4 method for signature 'SQLDataFrame'
dbtable(x)
```

```
## S4 method for signature 'SQLDataFrame'
dbkey(x)

## S4 replacement method for signature 'SQLDataFrame'
dbkey(x) <- value

## S4 method for signature 'SQLDataFrame'
dbconcatKey(x)

## S4 method for signature 'SQLDataFrame'
dbnrows(x)

## S4 method for signature 'SQLDataFrame'
ROWNAMES(x)

## S4 method for signature 'SQLDataFrame'
show(object)

## S4 method for signature 'SQLDataFrame'
as.data.frame(x, row.names = NULL, optional = NULL, ...)
```

Arguments

conn	a valid DBIConnection from SQLite, MySQL or BigQuery. If provided, arguments of 'user', 'host', 'dbname', 'password' will be ignored.
host	host name for MySQL database or project name for BigQuery.
user	user name for SQL database.
dbname	database name for SQL connection.
password	password for SQL database connection.
billing	the Google Cloud project name with authorized billing information.
type	The SQL database type, supports "SQLite", "MySQL" and "BigQuery".
dbtable	A character string for the table name in that database. If not provided and there is only one table available, it will be read in by default.
dbkey	A character vector for the name of key columns that could uniquely identify each row of the database table. Will be ignored for BigQueryConnection.
col.names	A character vector specifying the column names you want to read into the SQLDataFrame.
x	An SQLDataFrame object
value	The column name to be used as dbkey(x)
object	An SQLDataFrame object.
row.names	NULL or a character vector giving the row names for the data frame. Only including this argument for the as.data.frame generic. Does not apply to SQLDataFrame. See base::as.data.frame for details.
optional	logical. If TRUE, setting row names and converting column names is optional. Only including this argument for the as.data.frame generic. Does not apply to SQLDataFrame. See base::as.data.frame for details.
...	additional arguments to be passed.

Value

A SQLDataFrame object.

Examples

```
## SQLDataFrame construction
dbname <- system.file("extdata/test.db", package = "SQLDataFrame")
conn <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = dbname)
obj <- SQLDataFrame(conn = conn, dbtable = "state",
                    dbkey = "state")
obj1 <- SQLDataFrame(conn = conn, dbtable = "state",
                     dbkey = c("region", "population"))
obj1

### construction from database credentials
obj2 <- SQLDataFrame(dbname = dbname, type = "SQLite",
                     dbtable = "state", dbkey = "state")
all.equal(obj, obj2)  ## [1] TRUE

## slot accessors
connSQLDataFrame(obj)
dbtable(obj)
dbkey(obj)
dbkey(obj1)

dbconcatKey(obj)
dbconcatKey(obj1)

## slot accessors (for internal use only)
tblData(obj)
dbnrows(obj)

## ROWNAMES
ROWNAMES(obj[sample(10, 5), ])
ROWNAMES(obj1[sample(10, 5), ])

## coercion
as.data.frame(obj)
as(obj, "DataFrame")


## dbkey replacement
dbkey(obj) <- c("region", "population")
obj

## construction from MySQL
## Not run:
mysqlConn <- DBI::dbConnect(dbDriver("MySQL"),
                            host = "",
                            user = "",
                            password = "",  ## required if need further
                                            ## aggregation operations such as
                                            ## join, union, rbind, etc.
                            dbname = "")
sdf <- SQLDataFrame(conn = mysqlConn, dbtable = "", dbkey = "")
```

```
## Or pass credentials directly into constructor
objensb <- SQLDataFrame(user = "genome",
                        host = "genome-mysql.soe.ucsc.edu",
                        dbname = "xenTro9",
                        type = "MySQL",
                        dbtable = "xenoRefGene",
                        dbkey = c("name", "txStart"))


## End(Not run)

## construction from BigQuery
## Not run:
con <- DBI::dbConnect(bigquery(),  ## equivalent dbDriver("bigquery")
                      project = "bigquery-public-data",
                      dataset = "human_variant_annotation",
                      billing = "")  ## your project name that linked to
                                     ## Google Cloud with billing information.
sdf <- SQLDataFrame(conn = con, dbtable = "ncbi_clinvar_hg38_20180701")

## Or pass credentials directly into constructor
sdf1 <- SQLDataFrame(host = "bigquery-public-data",
                     dbname = "human_variant_annotation",
                     billing = "",
                     type = "BigQuery",
                     dbtable = "ncbi_clinvar_hg38_20180701")


## End(Not run)
```

`SQLDataFrame-methods` *SQLDataFrame methods*

Description

`head,tail`: Retrieve the first / last n rows of the `SQLDataFrame` object. See `?S4Vectors::head` for more details.

`dim,dimnames,length,names`: Retrieve the dimension, dimension names, number of columns and colnames of SQLDataFrame object.

`[i,j]` supports subsetting by `i` (for row) and `j` (for column) and respects 'drop=FALSE'.

Use `select()` function to select certain columns.

Use `filter()` to choose rows/cases where conditions are true.

`mutate()` adds new columns and preserves existing ones; It also preserves the number of rows of the input. New variables overwrite existing variables of the same name.

`connSQLDataFrame` returns the connection of a SQLDataFrame object.

Usage

```
## S4 method for signature 'SQLDataFrame'
head(x, n = 6L)
```

```
## S4 method for signature 'SQLDataFrame'
tail(x, n = 6L)

## S4 method for signature 'SQLDataFrame'
dim(x)

## S4 method for signature 'SQLDataFrame'
dimnames(x)

## S4 method for signature 'SQLDataFrame'
length(x)

## S4 method for signature 'SQLDataFrame'
names(x)

## S4 method for signature 'SQLDataFrame,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'SQLDataFrame,SQLDataFrame'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'SQLDataFrame,list'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'SQLDataFrame'
x[[i, j, ...]]

## S4 method for signature 'SQLDataFrame'
x$name

## S3 method for class 'SQLDataFrame'
select(.data, ...)

## S3 method for class 'SQLDataFrame'
filter(.data, ...)

## S3 method for class 'SQLDataFrame'
mutate(.data, ...)

connSQLDataFrame(x)
```

Arguments

x	An SQLDataFrame object.
n	Number of rows.
i	Row subscript. Could be numeric / character / logical values, a named list of key values, and SQLDataFrame, data.frame, tibble objects.
j	Column subscript.
...	additional arguments to be passed. • select(): One or more unquoted expressions separated by commas. You can treat variable names like they are positions, so you can use expressions

like 'x:y' to select ranges of variables. Positive values select variables; negative values drop variables. See ?dplyr::select for more details.

- `filter()`: Logical predicates defined in terms of the variables in '.data'. Multiple conditions are combined with '&'. Only rows where the condition evaluates to 'TRUE' are kept. See ?dplyr::filter for more details.
- `mutate()`: Name-value pairs of expressions, each with length 1 or the same length as the number of rows in the group (if using 'group_by()') or in the entire input (if not using groups). The name of each argument will be the name of a new variable, and the value will be its corresponding value. New variables overwrite existing variables of the same name. NOTE that the new value could only be of length 1 or the operation of existing columns. If a new vector of values are given, error will return. This is due to the internal method of 'mutate.tbl_lazy' not being able to take new arbitrary values.

`drop` Whether to drop with reduced dimension. Default is TRUE.

`name` column name to be extracted by $.

`.data` A SQLDataFrame object.

Value

`head,tail`: An SQLDataFrame object with certain rows.

`dim`: interger vector

`dimnames`: A list of character vectors.

`length`: An integer

`names`: A character vector

`[i,j]`: A SQLDataFrame object or vector with realized column values (with single column subsetting and default `drop=TRUE`.)

`select`: always returns a SQLDataFrame object no matter how may columns are selected. If only key column(s) is(are) selected, it will return a SQLDataFrame object with 0 col (only key columns are shown).

`filter`: A SQLDataFrame object with subset rows of the input SQLDataFrame object matching conditions.

`mutate`: A SQLDataFrame object.

Examples

```
##################
## basic methods
##################

test.db <- system.file("extdata/test.db", package = "SQLDataFrame")
conn <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db)
obj <- SQLDataFrame(conn = conn, dbtable = "state", dbkey = "state")
dim(obj)
dimnames(obj)
length(obj)
names(obj)

obj1 <- SQLDataFrame(conn = conn, dbtable = "state",
                     dbkey = c("region", "population"))
```

```
###############
## subsetting
###############

obj[1]
obj["region"]
obj$region
obj[]
obj[,]
obj[NULL, ]
obj[, NULL]

## by numeric / logical / character vectors
obj[1:5, 2:3]
obj[c(TRUE, FALSE), c(TRUE, FALSE)]
obj[c("Alabama", "South Dakota"), ]
obj1[c("South:3615.0", "West:3559.0"), ]
### Remeber to add `.0` trailing for numeric values. If not sure,
### check `ROWNAMES()`.

## by SQLDataFrame
obj_sub <- obj[sample(10), ]
obj[obj_sub, ]

## by a named list of key column values (or equivalently data.frame /
## tibble)
obj[data.frame(state = c("Colorado", "Arizona")), ]
obj[tibble::tibble(state = c("Colorado", "Arizona")), ]
obj[list(state = c("Colorado", "Arizona")), ]
obj1[list(region = c("South", "West"),
          population = c("3615.0", "365.0")), ]
### remember to add the '.0' trailing for numeric values. If not sure,
### check `ROWNAMES()`.

## Subsetting with key columns

obj["state"] ## list style subsetting, return a SQLDataFrame object with col = 0.
obj[c("state", "division")] ## list style subsetting, return a SQLDataFrame object with col = 1.
obj[, "state"] ## realize specific key column value.
obj[, c("state", "division")] ## col = 1, but do not realize.


###################
## select, filter, mutate
###################
library(dplyr)
obj %>% select(division) ## equivalent to obj["division"], or obj[, "division", drop = FALSE]
obj %>% select(region:size)

obj %>% filter(region == "West" & size == "medium")
obj1 %>% filter(region == "West" & population > 10000)

obj %>% mutate(p1 = population / 10)
obj %>% mutate(s1 = size)

obj %>% select(region, size, population) %>%
    filter(population > 10000) %>%
```

```
    mutate(pK = population/1000)
obj1 %>% select(region, size, population) %>%
    filter(population > 10000) %>%
    mutate(pK = population/1000)

###################
## connection info
###################

connSQLDataFrame(obj)
```

---

union *Union of* SQLDataFrame *objects*

Description

Performs union operations on SQLDataFrame objects.

Usage

```
## S4 method for signature 'SQLDataFrame,SQLDataFrame'
union(x, y, localConn, ...)
```

Arguments

x	A SQLDataFrame object.
y	A SQLDataFrame object.
localConn	A MySQL connection with write permission. Will be used only when unioning two SQLDataFrame objects from different MySQL connections (to different MySQL databases), and neither has write permission. The situation is rare and operation is expensive. See Details for suggestions.
...	Other arguments passed on to methods.

Details

The union function supports aggregation of SQLDataFrame objects from same or different connection (e.g., cross databases), either with or without write permission.

SQLite database tables are supported by SQLDataFrame package, in the same/cross-database aggregation, and saving.

For MySQL databases, There are different situations:

When the input SQLDataFrame objects connects to same remote MySQL database without write permission (e.g., ensembl), the functions work like dbplyr with the lazy operations and a DataFrame interface. Note that the unioned SQLDataFrame can not be saved using saveSQLDataFrame.

When the input SQLDataFrame objects connects to different MySQL databases, and neither has write permission, the union function is supported but will be quite expensive. To avoid this situation, a more efficient way is to save the database table in local MySQL server (with write permission) using saveSQLDataFrame, and then call the union function again.

More frequent situation will be the union operation on two SQLDataFrame objects, of which at least one has write permission. Then the cross-database aggregation through SQLDataFrame package will be supported by generating federated table from the non-writable connection in the writable connection. Look for MySQL database manual for more details.

NOTE also, that the union operation on SQLDataFrame objects will perform differently on database table from SQLite or MySQL. For SQLite tables, the union will sort the data using the key columns, and write the sorted data into a provided SQLite dbname when saveSQLDataFrame was called. For MySQL tables, union preserves the orders, but will be sorted with key columns and saved into a user provided MySQL database (with write permission) when saveSQLdataFrame was called.

Value

A SQLDataFrame object.

Examples

```
test.db1 <- system.file("extdata/test.db", package = "SQLDataFrame")
test.db2 <- system.file("extdata/test1.db", package = "SQLDataFrame")
con1 <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db1)
con2 <- DBI::dbConnect(DBI::dbDriver("SQLite"), dbname = test.db2)
obj1 <- SQLDataFrame(conn = con1,
                     dbtable = "state",
                     dbkey = c("region", "population"))
obj2 <- SQLDataFrame(conn = con2,
                     dbtable = "state1",
                     dbkey = c("region", "population"))
obj1_sub <- obj1[1:10, 2:3]
obj2_sub <- obj2[8:15,2:3]

## union
res_union <- union(obj1_sub, obj2_sub)  ## sorted
dim(res_union)
```

Index