Package 'tidyquery'

February 6, 2021

Type Package

Title Query 'R' Data Frames with 'SQL'

Version 0.2.2

Maintainer Ian Cook <ianmcook@gmail.com>

Description Use 'SQL' 'SELECT' statements to query 'R' data frames.

URL https://github.com/ianmcook/tidyquery

BugReports https://github.com/ianmcook/tidyquery/issues

NeedsCompilation no

License Apache License 2.0

Encoding UTF-8

RoxygenNote 7.1.1

Imports dplyr (>= 0.7.4), lubridate (>= 1.6.0), queryparser (>= 0.3.1), rlang (>= 0.2.0), stringr (>= 1.0.0), utils

Suggests covr (>= 3.2.0), DBI (>= 0.7), dbplyr (>= 1.2.1), dtplyr (>= 1.0.0), nycflights13, RSQLite (>= 2.1.0), testthat (>= 2.1.0)

Collate 'compat.R' 'query.R' 'join.R' 'quote.R' 'remove.R' 'replace.R' 'show_dplyr.R' 'unscope.R'

Config/testthat/edition 3

Author Ian Cook [aut, cre], Cloudera [cph]

Repository CRAN

Date/Publication 2021-02-06 07:30:04 UTC

R topics documented:

query *Query an R data frame with SQL*

Description

query takes a SQL SELECT statement and uses it to query an R data frame

Usage

```
query(data, sql)
```

Arguments

data	a data frame or data frame-like object (optional)
sql	a character string containing a SQL SELECT statement

Details

If the data argument is not specified, then the FROM clause of the SQL statement determines which data frame to query.

The names of data frames and columns are case-sensitive (like in R). Keywords and function names are not case-sensitive (like in SQL).

In addition to R data frames and tibbles (tbl_df objects), this function can query dtplyr_step objects created by **dtplyr**, a **data.table** backend for **dbplyr**. It is also possible to use this function together with **dbplyr** to query remote database tables (tbl_sql objects), but this depends on which database and which backend package (if any) you are using, so results may vary.

This function is subject to the current limitations of the **queryparser** package.

Value

An object of the same class as data.

Examples

```
library(dplyr)

iris %>% query("SELECT Species, AVG(Petal.Length) GROUP BY Species")

query("SELECT Species, AVG(Petal.Length) FROM iris GROUP BY Species")

iris %>%
  filter(Petal.Length > 4) %>%
  query("SELECT Species, MAX(Sepal.Length) AS max_sep_len
           GROUP BY Species") %>%
  arrange(desc(max_sep_len))

library(nycflights13)
```

```
query <- "SELECT origin, dest,
    COUNT(flight) AS num_flts,
    round(AVG(distance)) AS dist,
    round(AVG(arr_delay)) AS avg_delay
  FROM flights
  WHERE distance BETWEEN 200 AND 300
    AND air_time IS NOT NULL
  GROUP BY origin, dest
  HAVING num_flts > 5000
  ORDER BY num_flts DESC, avg_delay DESC
  LIMIT 100;"

query(query)
```

`show_dplyr` *Show dplyr code equivalent to a SQL query*

Description

`show_dplyr` takes a SQL `SELECT` statement and prints equivalent dplyr code

Usage

```
show_dplyr(data, sql)
```

Arguments

`data`	a data frame or data frame-like object (optional)
`sql`	a character string containing a SQL `SELECT` statement

Details

For more details, see `query`. Instead of running the dplyr code like `query` does, `show_dplyr` prints the dplyr code.

In function calls in the printed code, long lists of arguments may be truncated and appended with `....`

See Also

`query`

Examples

```
library(dplyr)
library(nycflights13)

query <- "SELECT origin, dest,
    COUNT(flight) AS num_flts,
    round(AVG(distance)) AS dist,
    round(AVG(arr_delay)) AS avg_delay
  FROM flights
  WHERE distance BETWEEN 200 AND 300
    AND air_time IS NOT NULL
  GROUP BY origin, dest
  HAVING num_flts > 5000
  ORDER BY num_flts DESC, avg_delay DESC
  LIMIT 100;"

show_dplyr(query)
```

Index