Package 'dtplyr'

February 25, 2019

Title Data Table Back-End for 'dplyr'

Version 0.0.3

Description This implements the data table back-end for 'dplyr' so that you can seamlessly use data table and 'dplyr' together.

License GPL (>= 2)

Imports dplyr (>= 0.5.0), data.table (>= 1.9.6), lazyeval, rlang

Suggests Lahman, nycflights13, testthat, covr

LazyData true

RoxygenNote 6.1.1

URL https://github.com/hadley/dtplyr

BugReports https://github.com/hadley/dtplyr/issues

NeedsCompilation no

Author Lionel Henry [cre],
Hadley Wickham [aut],
RStudio [cph]

Maintainer Lionel Henry <lionel@rstudio.com>

Repository CRAN

Date/Publication 2019-02-25 19:40:03 UTC

R topics documented:

dtplyr-package *dtplyr: Data Table Back-End for 'dplyr'*

Description

This implements the data table back-end for 'dplyr' so that you can seamlessly use data table and 'dplyr' together.

Author(s)

Maintainer: Lionel Henry <lionel@rstudio.com>

Authors:

- Hadley Wickham <hadley@rstudio.com>

Other contributors:

- RStudio [copyright holder]

See Also

Useful links:

- https://github.com/hadley/dtplyr
- Report bugs at https://github.com/hadley/dtplyr/issues

grouped_dt *A grouped data table.*

Description

The easiest way to create a grouped data table is to call the group_by method on a data table or tbl: this will take care of capturing the unevalated expressions for you.

Usage

```
grouped_dt(data, vars, copy = TRUE)

is.grouped_dt(x)
```

Arguments

data	a tbl or data frame.
vars	a list of quoted variables.
copy	If TRUE, will make copy of input.
x	an object to check

Examples

```
library(dplyr, warn.conflicts = FALSE)
if (require("nycflights13")) {
flights_dt <- tbl_dt(flights)
group_size(group_by(flights_dt, year, month, day))
group_size(group_by(flights_dt, dest))

monthly <- group_by(flights_dt, month)
summarise(monthly, n = n(), delay = mean(arr_delay))
}
```

`join.tbl_dt` *Join data table tbls.*

Description

See `join` for a description of the general purpose of the functions.

Usage

```
inner_join.data.table(x, y, by = NULL, copy = FALSE, suffix = c(".x",
  ".y"), ...)

left_join.data.table(x, y, by = NULL, copy = FALSE, suffix = c(".x",
  ".y"), ...)

right_join.data.table(x, y, by = NULL, copy = FALSE, suffix = c(".x",
  ".y"), ...)

full_join.data.table(x, y, by = NULL, copy = FALSE, suffix = c(".x",
  ".y"), ...)

semi_join.data.table(x, y, by = NULL, copy = FALSE, ...)

anti_join.data.table(x, y, by = NULL, copy = FALSE, ...)
```

Arguments

`x, y`	tbls 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. A message lists the variables so that you can check they're right (to suppress the message, simply explicitly list the variables that you want to join). To join by different variables on x and y use a named vector. For example, `by = c("a" = "b")` will match `x.a` to `y.b`.

copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.
suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
...	Included for compatibility with generic; otherwise ignored.

Examples

```
library(dplyr, warn.conflicts = FALSE)

if (require("Lahman")) {
batting_dt <- tbl_dt(Batting)
person_dt <- tbl_dt(Master)

# Inner join: match batting and person data
inner_join(batting_dt, person_dt)

# Left join: keep batting data even if person missing
left_join(batting_dt, person_dt)

# Semi-join: find batting data for top 4 teams, 2010:2012
grid <- expand.grid(
  teamID = c("WAS", "ATL", "PHI", "NYA"),
  yearID = 2010:2012)
top4 <- semi_join(batting_dt, grid, copy = TRUE)

# Anti-join: find batting data with out player data
anti_join(batting_dt, person_dt)
}
```

---

src_dt *A local data table source.*

Description

This is mainly useful for testing, since makes it possible to refer to local and remote tables using exactly the same syntax.

Usage

```
src_dt(pkg = NULL, env = NULL)
```

Arguments

pkg, env	Either the name of a package or an environment object in which to look for objects.

tbl_dt *Create a data table tbl.*

Description

A data table tbl wraps a local data table.

Usage

```
tbl_dt(data, copy = TRUE)
```

Arguments

data	a data table
copy	If the input is a data.table, copy it?

Examples

```
ds <- tbl_dt(mtcars)
ds
data.table::as.data.table(ds)

library(dplyr, warn.conflicts = FALSE)
if (require("nycflights13")) {
flights2 <- tbl_dt(flights)
flights2 %>% filter(month == 1, day == 1, dest == "DFW")
flights2 %>% select(year:day)
flights2 %>% rename(Year = year)
flights2 %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE),
    n = length(arr_delay)
  )
flights2 %>%
  mutate(gained = arr_delay - dep_delay) %>%
  select(ends_with("delay"), gained)
flights2 %>%
  arrange(dest, desc(arr_delay))

by_dest <- group_by(flights2, dest)

filter(by_dest, arr_delay == max(arr_delay, na.rm = TRUE))
summarise(by_dest, arr = mean(arr_delay, na.rm = TRUE))

# Normalise arrival and departure delays by airport
by_dest %>%
  mutate(arr_z = scale(arr_delay), dep_z = scale(dep_delay)) %>%
  select(starts_with("arr"), starts_with("dep"))

arrange(by_dest, desc(arr_delay))
```

```
select(by_dest, -(day:tailnum))
rename(by_dest, Year = year)

# All manip functions preserve grouping structure, except for summarise
# which removes a grouping level
by_day <- group_by(flights2, year, month, day)
by_month <- summarise(by_day, delayed = sum(arr_delay > 0, na.rm = TRUE))
by_month
summarise(by_month, delayed = sum(delayed))

# You can also manually ungroup:
ungroup(by_day)
}
```

Index