Package 'dtplyr'

February 20, 2021

Title Data Table Back-End for 'dplyr'

Version 1.1.0

Description Provides a data.table backend for 'dplyr'. The goal of 'dtplyr' is to allow you to write 'dplyr' code that is automatically translated to the equivalent, but usually much faster, data.table code.

License MIT + file LICENSE

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

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

Depends R (>= 3.3)

Imports crayon, data.table (>= 1.12.4), dplyr (>= 1.0.3), ellipsis, glue, lifecycle, rlang, tibble, tidyselect, vctrs

Suggests bench, covr, knitr, rmarkdown, testthat (>= 3.0.0), tidyr (>= 1.1.0)

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Config/testthat/edition 3

NeedsCompilation no

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

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2021-02-20 01:50:05 UTC

R topics documented:

`arrange.dtplyr_step` *Arrange rows by column values*

Description

This is a method for dplyr generic `arrange()`. It is translated to an `order()` call in the `i` argument of [.data.table.

Usage

```
## S3 method for class 'dtplyr_step'
arrange(.data, ..., .by_group = FALSE)
```

Arguments

`.data`	A `lazy_dt()`.
`...`	<`data-masking`> Variables, or functions of variables. Use `desc()` to sort a variable in descending order.
`.by_group`	If `TRUE`, will sort first by grouping variable. Applies to grouped data frames only.

Examples

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

dt <- lazy_dt(mtcars)
dt %>% arrange(vs, cyl)
dt %>% arrange(desc(vs), cyl)
dt %>% arrange(across(mpg:disp))
```

`collect.dtplyr_step` *Force computation of a lazy data.table*

Description

- `collect()` returns a tibble, grouped if needed.
- `compute()` generates an intermediate assignment in the translation.
- `as.data.table()` returns a data.table.
- `as.data.frame()` returns a data frame.
- `as_tibble()` returns a tibble.

Usage

```
## S3 method for class 'dtplyr_step'
collect(x, ...)

## S3 method for class 'dtplyr_step'
compute(x, name = unique_name(), ...)

## S3 method for class 'dtplyr_step'
as.data.table(x, keep.rownames = FALSE, ...)

## S3 method for class 'dtplyr_step'
as.data.frame(x, ...)

## S3 method for class 'dtplyr_step'
as_tibble(x, ..., .name_repair = "check_unique")
```

Arguments

`x`	A lazy_dt
`...`	Arguments used by other methods.
`name`	Name of intermediate data.table.
`keep.rownames`	Ignored as dplyr never preserves rownames.
`.name_repair`	Treatment of problematic column names

Examples

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

dt <- lazy_dt(mtcars)

# Generate translation
avg_mpg <- dt %>%
  filter(am == 1) %>%
  group_by(cyl) %>%
  summarise(mpg = mean(mpg))

# Show translation and temporarily compute result
avg_mpg

# compute and return tibble
avg_mpg_tb <- as_tibble(avg_mpg)
avg_mpg_tb

# compute and return data.table
avg_mpg_dt <- data.table::as.data.table(avg_mpg)
avg_mpg_dt

# modify translation to use intermediate assignment
compute(avg_mpg)
```

---

count.dtplyr_step *Count observations by group*

Description

This is a method for the dplyr `count()` generic. It is translated using `.N` in the `j` argument, and supplying groups to `keyby` as appropriate.

Usage

```
## S3 method for class 'dtplyr_step'
count(.data, ..., wt = NULL, sort = FALSE, name = NULL)
```

Arguments

`.data`	A `lazy_dt()`
`...`	<`data-masking`> Variables to group by.
`wt`	<`data-masking`> Frequency weights. Can be `NULL` or a variable: • If `NULL` (the default), counts the number of rows in each group. • If a variable, computes `sum(wt)` for each group.
`sort`	If `TRUE`, will show the largest groups at the top.

name	The name of the new column in the output. If omitted, it will default to n. If there's already a column called n, it will error, and require you to specify the name.

Examples

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

dt <- lazy_dt(dplyr::starwars)
dt %>% count(species)
dt %>% count(species, sort = TRUE)
dt %>% count(species, wt = mass, sort = TRUE)
```

---

distinct.dtplyr_step *Subset distinct/unique rows*

Description

This is a method for the dplyr `distinct()` generic. It is translated to `data.table::unique.data.table()`.

Usage

```
## S3 method for class 'dtplyr_step'
distinct(.data, ..., .keep_all = FALSE)
```

Arguments

.data	A `lazy_dt()`
...	<data-masking> Optional variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, will use all variables.
.keep_all	If TRUE, keep all variables in .data. If a combination of ... is not distinct, this keeps the first row of values.

Examples

```
library(dplyr, warn.conflicts = FALSE)
df <- lazy_dt(data.frame(
  x = sample(10, 100, replace = TRUE),
  y = sample(10, 100, replace = TRUE)
))

df %>% distinct(x)
df %>% distinct(x, y)
df %>% distinct(x, .keep_all = TRUE)
```

filter.dtplyr_step *Subset rows using column values*

Description

This is a method for the dplyr `arrange()` generic. It is translated to the `i` argument of [.data.table

Usage

```
## S3 method for class 'dtplyr_step'
filter(.data, ..., .preserve = FALSE)
```

Arguments

`.data`	A `lazy_dt()`.
`...`	<`data-masking`> Expressions that return a logical value, and are defined in terms of the variables in `.data`. If multiple expressions are included, they are combined with the `&` operator. Only rows for which all conditions evaluate to `TRUE` are kept.
`.preserve`	Ignored

Examples

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

dt <- lazy_dt(mtcars)
dt %>% filter(cyl == 4)
dt %>% filter(vs, am)

dt %>%
  group_by(cyl) %>%
  filter(mpg > mean(mpg))
```

group_by.dtplyr_step *Group and ungroup*

Description

These are methods for dplyr's `group_by()` and `ungroup()` generics. Grouping is translated to the either `keyby` and `by` argument of [.data.table depending on the value of the `arrange` argument.

Usage

```
## S3 method for class 'dtplyr_step'
group_by(.data, ..., .add = FALSE, add = deprecated(), arrange = TRUE)

## S3 method for class 'dtplyr_step'
ungroup(.data, ...)
```

Arguments

.data	A lazy_dt()
...	In group_by(), variables or computations to group by. In ungroup(), variables to remove from the grouping.
.add, add	When FALSE, the default, group_by() will override existing groups. To add to the existing groups, use .add = TRUE. This argument was previously called add, but that prevented creating a new grouping variable called add, and conflicts with our naming conventions.
arrange	If TRUE, will automatically arrange the output of subsequent grouped operations by group. If FALSE, output order will be left unchanged. In the generated data.table code this switches between using the keyby (TRUE) and by (FALSE) arguments.

Examples

```
library(dplyr, warn.conflicts = FALSE)
dt <- lazy_dt(mtcars)

# group_by() is usually translated to `keyby` so that the groups
# are ordered in the output
dt %>%
 group_by(cyl) %>%
 summarise(mpg = mean(mpg))

# use `arrange = FALSE` to instead use `by` so the original order
# or groups is preserved
dt %>%
 group_by(cyl, arrange = FALSE) %>%
 summarise(mpg = mean(mpg))
```

---

group_modify.dtplyr_step

Apply a function to each group

Description

These are methods for the dplyr group_map() and group_modify() generics. They are both translated to [.data.table.

Usage

```
## S3 method for class 'dtplyr_step'
group_modify(.tbl, .f, ..., keep = FALSE)

## S3 method for class 'dtplyr_step'
group_map(.tbl, .f, ..., keep = FALSE)
```

Arguments

.tbl	A lazy_dt()
.f	The name of a two argument function. The first argument is passed .SD,the data.table representing the current group; the second argument is passed .BY, a list giving the current values of the grouping variables. The function should return a list or data.table.
...	Additional arguments passed to .f
keep	Not supported for lazy_dt.

Value

group_map() applies .f to each group, returning a list. group_modify() replaces each group with the results of .f, returning a modified lazy_dt().

Examples

```
library(dplyr)

dt <- lazy_dt(mtcars)

dt %>%
  group_by(cyl) %>%
  group_modify(head, n = 2L)

dt %>%
  group_by(cyl) %>%
  group_map(head, n = 2L)
```

---

head.dtplyr_step *Subset first or last rows*

Description

These are methods for the base generics head() and tail(). They are not translated.

Usage

```
## S3 method for class 'dtplyr_step'
head(x, n = 6L, ...)

## S3 method for class 'dtplyr_step'
tail(x, n = 6L, ...)
```

Arguments

x	A lazy_dt()
n	Number of rows to select. Can use a negative number to instead drop rows from the other end.
...	Passed on to head()/tail().

Examples

```
library(dplyr, warn.conflicts = FALSE)
dt <- lazy_dt(data.frame(x = 1:10))

# first three rows
head(dt, 3)
# last three rows
tail(dt, 3)

# drop first three rows
tail(dt, -3)
```

`intersect.dtplyr_step` *Set operations*

Description

These are methods for the dplyr generics `intersect()`, `union()`, `union_all()`, and `setdiff()`. They are translated to `data.table::fintersect()`, `data.table::funion()`, and `data.table::fsetdiff()`.

Usage

```
## S3 method for class 'dtplyr_step'
intersect(x, y, ...)

## S3 method for class 'dtplyr_step'
union(x, y, ...)

## S3 method for class 'dtplyr_step'
union_all(x, y, ...)

## S3 method for class 'dtplyr_step'
setdiff(x, y, ...)
```

Arguments

`x, y`	A pair of `lazy_dt()`s.
`...`	Ignored

Examples

```
dt1 <- lazy_dt(data.frame(x = 1:4))
dt2 <- lazy_dt(data.frame(x = c(2, 4, 6)))

intersect(dt1, dt2)
union(dt1, dt2)
setdiff(dt1, dt2)
```

lazy_dt *Create a "lazy" data.table for use with dplyr verbs*

Description

A lazy data.table lazy captures the intent of dplyr verbs, only actually performing computation when requested (with `collect()`, `pull()`, `as.data.frame()`, `data.table::as.data.table()`, or `tibble::as_tibble()`). This allows dtplyr to convert dplyr verbs into as few data.table expressions as possible, which leads to a high performance translation.

See `vignette("translation")` for the details of the translation.

Usage

```
lazy_dt(x, name = NULL, immutable = TRUE, key_by = NULL)
```

Arguments

`x`	A data table (or something can can be coerced to a data table).
`name`	Optionally, supply a name to be used in generated expressions. For expert use only.
`immutable`	If `TRUE`, `x` is treated as immutable and will never be modified by any code generated by dtplyr. Alternatively, you can set `immutable = FALSE` to allow dtplyr to modify the input object.
`key_by`	Set keys for data frame, using `select()` semantics (e.g. `key_by = c(key1, key2)`. This uses `data.table::setkey()` to sort the table and build an index. This will considerably improve performance for subsets, summaries, and joins that use the keys. See `vignette("datatable-keys-fast-subset")` for more details.

Examples

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

# If you have a data.table, using it with any dplyr generic will
# automatically convert it to a lazy_dt object
dt <- data.table::data.table(x = 1:10, y = 10:1)
dt %>% filter(x == y)
dt %>% mutate(z = x + y)

# Note that dtplyr will avoid mutating the input data.table, so the
# previous translation includes an automatic copy(). You can avoid this
# with a manual call to lazy_dt()
dt %>%
  lazy_dt(immutable = FALSE) %>%
  mutate(z = x + y)

# If you have a data frame, you can use lazy_dt() to convert it to
```

```
# a data.table:
mtcars2 <- lazy_dt(mtcars)
mtcars2
mtcars2 %>% select(mpg:cyl)
mtcars2 %>% select(x = mpg, y = cyl)
mtcars2 %>% filter(cyl == 4) %>% select(mpg)
mtcars2 %>% select(mpg, cyl) %>% filter(cyl == 4)
mtcars2 %>% mutate(cyl2 = cyl * 2, cyl4 = cyl2 * 2)
mtcars2 %>% transmute(cyl2 = cyl * 2, vs2 = vs * 2)
mtcars2 %>% filter(cyl == 8) %>% mutate(cyl2 = cyl * 2)

# Learn more about translation in vignette("translation")
by_cyl <- mtcars2 %>% group_by(cyl)
by_cyl %>% summarise(mpg = mean(mpg))
by_cyl %>% mutate(mpg = mean(mpg))
by_cyl %>%
  filter(mpg < mean(mpg)) %>%
  summarise(hp = mean(hp))
```

`left_join.dtplyr_step` *Join data tables*

Description

These are methods for the dplyr generics `left_join()`, `right_join()`, `inner_join()`, `full_join()`, `anti_join()`, and `semi_join()`. The mutating joins (left, right, inner, and full) are translated to `data.table::merge.data.table()`, except for the special cases where it's possible to translate to [.data.table. Semi- and anti-joins have no direct data.table equivalent.

Usage

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

Arguments

`x, y`	A pair of `lazy_dt()`s.
`...`	Other parameters passed onto methods.
`by`	A character vector of variables to join by. If `NULL`, the default, *_join() will perform a natural join, using all variables in common across `x` and `y`. A message lists the variables so that you can check they're correct; suppress the message by supplying `by` explicitly. 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`. To join by multiple variables, use a vector with length > 1. For example, `by = c("a","b")` will match `x$a` to `y$a` and `x$b` to `y$b`. Use a named vector to match different variables in `x` and `y`. For example, `by = c("a" = "b","c" = "d")` will match `x$a` to `y$b` and `x$c` to `y$d`. To perform a cross-join, generating all combinations of `x` and `y`, use `by = character()`.

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.

Examples

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

band_dt <- lazy_dt(dplyr::band_members)
instrument_dt <- lazy_dt(dplyr::band_instruments)

band_dt %>% left_join(instrument_dt)
band_dt %>% right_join(instrument_dt)
band_dt %>% inner_join(instrument_dt)
band_dt %>% full_join(instrument_dt)

band_dt %>% semi_join(instrument_dt)
band_dt %>% anti_join(instrument_dt)
```

---

mutate.dtplyr_step *Create and modify columns*

Description

This is a method for the dplyr mutate() generic. It is translated to the j argument of [.data.table, using := to modify "in place".

Usage

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

Arguments

.data	A lazy_dt().
...	<data-masking> Name-value pairs. The name gives the name of the column in the output, and the value should evaluate to a vector.

Examples

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

dt <- lazy_dt(data.frame(x = 1:5, y = 5:1))
dt %>%
  mutate(a = (x + y) / 2, b = sqrt(x^2 + y^2))

# It uses a more sophisticated translation when newly created variables
```

```
# are used in the same expression
dt %>%
  mutate(x1 = x + 1, x2 = x1 + 1)
```

`pivot_wider.dtplyr_step` *Pivot data from long to wide*

Description

This is a method for the tidyr `pivot_wider()` generic. It is translated to `data.table::dcast()`

Usage

```
pivot_wider.dtplyr_step(
  data,
  id_cols = NULL,
  names_from = name,
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = value,
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

Arguments

`data`	A `lazy_dt()`.
`id_cols`	<`tidy-select`> A set of columns that uniquely identifies each observation. Defaults to all columns in `data` except for the columns specified in `names_from` and `values_from`. Typically used when you have redundant variables, i.e. variables whose values are perfectly correlated with existing variables.
`names_from`	<`tidy-select`> A pair of arguments describing which column (or columns) to get the name of the output column (`names_from`), and which column (or columns) to get the cell values from (`values_from`). If `values_from` contains multiple values, the value will be added to the front of the output column.
`names_prefix`	String added to the start of every variable name. This is particularly useful if `names_from` is a numeric vector and you want to create syntactic variable names.
`names_sep`	If `names_from` or `values_from` contains multiple variables, this will be used to join their values together into a single string to use as a column name.

names_glue	Instead of names_sep and names_prefix, you can supply a glue specification that uses the names_from columns (and special .value) to create custom column names.
names_sort	Should the column names be sorted? If FALSE, the default, column names are ordered by first appearance.
names_repair	What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates in the output, or "unique" to de-duplicated by adding numeric suffixes. See vctrs::vec_as_names() for more options.
values_from	<tidy-select> A pair of arguments describing which column (or columns) to get the name of the output column (names_from), and which column (or columns) to get the cell values from (values_from). If values_from contains multiple values, the value will be added to the front of the output column.
values_fill	Optionally, a (scalar) value that specifies what each value should be filled in with when missing. This can be a named list if you want to apply different aggregations to different value columns.
values_fn	A function, the default is length(). Note this is different behavior than tidyr::pivot_wider(), which returns a list column by default.
...	Additional arguments passed on to methods.

Examples

```
library(tidyr)

fish_encounters_dt <- lazy_dt(fish_encounters)
fish_encounters_dt
fish_encounters_dt %>%
  pivot_wider(names_from = station, values_from = seen)
# Fill in missing values
fish_encounters_dt %>%
  pivot_wider(names_from = station, values_from = seen, values_fill = 0)

# Generate column names from multiple variables
us_rent_income_dt <- lazy_dt(us_rent_income)
us_rent_income_dt
us_rent_income_dt %>%
  pivot_wider(names_from = variable, values_from = c(estimate, moe))

# When there are multiple `names_from` or `values_from`, you can use
# use `names_sep` or `names_glue` to control the output variable names
us_rent_income_dt %>%
  pivot_wider(
    names_from = variable,
    names_sep = ".",
    values_from = c(estimate, moe)
  )
```

```
# Can perform aggregation with values_fn
warpbreaks_dt <- lazy_dt(as_tibble(warpbreaks[c("wool", "tension", "breaks")]))
warpbreaks_dt
warpbreaks_dt %>%
  pivot_wider(
    names_from = wool,
    values_from = breaks,
    values_fn = mean
  )
```

`relocate.dtplyr_step` *Relocate variables using their names*

Description

This is a method for the dplyr `relocate()` generic. It is translated to the `j` argument of [.data.table.

Usage

```
## S3 method for class 'dtplyr_step'
relocate(.data, ..., .before = NULL, .after = NULL)
```

Arguments

`.data`	A `lazy_dt()`.
`...`	<`tidy-select`> Columns to move.
`.before`	<`tidy-select`> Destination of columns selected by `...`. Supplying neither will move columns to the left-hand side; specifying both is an error.
`.after`	<`tidy-select`> Destination of columns selected by `...`. Supplying neither will move columns to the left-hand side; specifying both is an error.

Examples

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

dt <- lazy_dt(data.frame(x = 1, y = 2, z = 3))

dt %>% relocate(z)
dt %>% relocate(y, .before = x)
dt %>% relocate(y, .after = y)
```

rename.dtplyr_step *Rename columns using their names*

Description

These are methods for the dplyr generics `rename()` and `rename_with()`. They are both translated to `data.table::setnames()`.

Usage

```
## S3 method for class 'dtplyr_step'
rename(.data, ...)

## S3 method for class 'dtplyr_step'
rename_with(.data, .fn, .cols = everything(), ...)
```

Arguments

`.data`	A `lazy_dt()`
`...`	For `rename()`: <`tidy-select`> Use `new_name = old_name` to rename selected variables. For `rename_with()`: additional arguments passed onto `.fn`.
`.fn`	A function used to transform the selected `.cols`. Should return a character vector the same length as the input.
`.cols`	<`tidy-select`> Columns to rename; defaults to all columns.

Examples

```
library(dplyr, warn.conflicts = FALSE)
dt <- lazy_dt(data.frame(x = 1, y = 2, z = 3))
dt %>% rename(new_x = x, new_y = y)
dt %>% rename_with(toupper)
```

select.dtplyr_step *Subset columns using their names*

Description

This is a method for the dplyr `select()` generic. It is translated to the `j` argument of [.data.table.

Usage

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

Arguments

.data	A lazy_dt().
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.

Examples

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

dt <- lazy_dt(data.frame(x1 = 1, x2 = 2, y1 = 3, y2 = 4))

dt %>% select(starts_with("x"))
dt %>% select(ends_with("2"))
dt %>% select(z1 = x1, z2 = x2)
```

---

slice.dtplyr_step *Subset rows using their positions*

Description

These are methods for the dplyr slice(), slice_head(), slice_tail(), slice_min(), slice_max() and slice_sample() generics. slice() and slice_sample() are translated to the i argument of [.data.table, all others are translated to the j argument.

Unlike dplyr, slice() (and slice() alone) returns the same number of rows per group, regardless of whether or not the indices appear in each group.

Usage

```
## S3 method for class 'dtplyr_step'
slice(.data, ...)

## S3 method for class 'dtplyr_step'
slice_head(.data, ..., n, prop)

## S3 method for class 'dtplyr_step'
slice_tail(.data, ..., n, prop)

## S3 method for class 'dtplyr_step'
slice_min(.data, order_by, ..., n, prop, with_ties = TRUE)

## S3 method for class 'dtplyr_step'
slice_max(.data, order_by, ..., n, prop, with_ties = TRUE)
```

Arguments

.data	A lazy_dt().
...	Positive integers giving rows to select, or negative integers giving rows to drop.
n, prop	Provide either n, the number of rows, or prop, the proportion of rows to select. If neither are supplied, n = 1 will be used. If n is greater than the number of rows in the group (or prop > 1), the result will be silently truncated to the group size. If the proportion of a group size is not an integer, it is rounded down.
order_by	Variable or function of variables to order by.
with_ties	Should ties be kept together? The default, TRUE, may return more rows than you request. Use FALSE to ignore ties, and return the first n rows.

Examples

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

dt <- lazy_dt(mtcars)
dt %>% slice(1, 5, 10)
dt %>% slice(-(1:4))

# First and last rows based on existing order
dt %>% slice_head(n = 5)
dt %>% slice_tail(n = 5)

# Rows with minimum and maximum values of a variable
dt %>% slice_min(mpg, n = 5)
dt %>% slice_max(mpg, n = 5)

# slice_min() and slice_max() may return more rows than requested
# in the presence of ties. Use with_ties = FALSE to suppress
dt %>% slice_min(cyl, n = 1)
dt %>% slice_min(cyl, n = 1, with_ties = FALSE)

# slice_sample() allows you to random select with or without replacement
dt %>% slice_sample(n = 5)
dt %>% slice_sample(n = 5, replace = TRUE)

# you can optionally weight by a variable - this code weights by the
# physical weight of the cars, so heavy cars are more likely to get
# selected
dt %>% slice_sample(weight_by = wt, n = 5)
```

---

summarise.dtplyr_step *Summarise each group to one row*

Description

This is a method for the dplyr summarise() generic. It is translated to the j argument of [.data.table.

Usage

```
## S3 method for class 'dtplyr_step'
summarise(.data, ...)
```

Arguments

.data	A lazy_dt().
...	<data-masking> Name-value pairs of summary functions. The name will be the name of the variable in the result. The value can be: • A vector of length 1, e.g. min(x), n(), or sum(is.na(y)). • A vector of length n, e.g. quantile(). • A data frame, to add multiple columns from a single expression.

Examples

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

dt <- lazy_dt(mtcars)

dt %>%
  group_by(cyl) %>%
  summarise(vs = mean(vs))

dt %>%
  group_by(cyl) %>%
  summarise(across(disp:wt, mean))
```

---

transmute.dtplyr_step *Create new columns, dropping old*

Description

This is a method for the dplyr transmute() generic. It is translated to the j argument of [.data.table.

Usage

```
## S3 method for class 'dtplyr_step'
transmute(.data, ...)
```

Arguments

.data	A lazy_dt().
...	<data-masking> Name-value pairs. The name gives the name of the column in the output, and the value should evaluate to a vector.

Examples

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

dt <- lazy_dt(dplyr::starwars)
dt %>% transmute(name, sh = paste0(species, "/", homeworld))
```

Index