Package 'table.express'

April 3, 2023

Type Package

Title Build 'data.table' Expressions with Data Manipulation Verbs

Description A specialization of 'dplyr' data manipulation verbs that parse and build expressions which are ultimately evaluated by 'data.table', letting it handle all optimizations. A set of additional verbs is also provided to facilitate some common operations on a subset of the data.

Version 0.4.2

Depends R (>= 3.2.0)

Imports methods, stats, utils, data.table (>= 1.9.8), dplyr, magrittr, R6, rlang (>= 0.3.1), tidyselect

Suggests knitr, rex, rmarkdown, testthat

Date 2023-04-02

BugReports https://github.com/asardaes/table.express/issues

License MPL-2.0

URL https://asardaes.github.io/table.express/, https://github.com/asardaes/table.express

Language en-US

Encoding UTF-8

RoxygenNote 7.2.3

VignetteBuilder knitr

Collate 'DELIMITERS-chain.R' 'DELIMITERS-end_expr.R' 'DELIMITERS-start_expr.R' 'R6-ExprBuilder.R' 'R6-EagerExprBuilder.R' 'UTILS-frame_append.R' 'UTILS-joins.R' 'UTILS-misc.R' 'UTILS-nest_expr.R' 'UTILS-tidyselect.R' 'VERBS-anti_join.R' 'VERBS-arrange.R' 'VERBS-distinct.R' 'VERBS-filter.R' 'VERBS-filter_on.R' 'VERBS-filter_sd.R' 'VERBS-full_join.R' 'VERBS-group_by.R' 'VERBS-inner_join.R' 'VERBS-key_by.R' 'VERBS-left_join.R' 'VERBS-max_by.R' 'VERBS-min_by.R' 'pkg.R' 'VERBS-mutate.R' 'VERBS-mutate_join.R' 'VERBS-mutate_sd.R' 'VERBS-order_by.R' 'VERBS-right_join.R' 'VERBS-select.R' 'VERBS-semi_join.R' 'VERBS-summarize.R' 'VERBS-transmute.R' 'VERBS-transmute_sd.R' 'VERBS-where.R'

NeedsCompilation no

Author Alexis Sarda-Espinosa [cre, aut]

Maintainer Alexis Sarda-Espinosa <alexis.sarda@gmail.com>

Repository CRAN

Date/Publication 2023-04-02 22:30:02 UTC

R topics documented:

`table.express-package` *Building 'data.table' expressions with data manipulation verbs*

Description

A specialization of `dplyr` verbs, as well as a set of custom ones, that build expressions that can be used within a `data.table`'s frame.

Note

Note that since version 0.3.0, it is not possible to load **table.express** and **dtplyr** at the same time, since they define the same `data.table` methods for many **dplyr** generics.

Bearing in mind that `data.tables` are also `data.frames`, we have to consider that other packages may uses `dplyr` internally without importing `data.table`. Since `dplyr`'s methods are generic, calls to these methods in such packages would fail. The functions in this package try to detect when this happens and delegate to the `data.frame` methods with a warning, which can be safely ignored if you know that the error originates from a package that is not meant to work with `data.table`. To avoid the warning, use `options(table.express.warn.cedta = FALSE)`.

This software package was developed independently of any organization or institution that is or has been associated with the author.

Author(s)

Alexis Sarda-Espinosa

See Also

Useful links:

- https://asardaes.github.io/table.express/
- https://github.com/asardaes/table.express
- Report bugs at https://github.com/asardaes/table.express/issues

Examples

```
require("data.table")

data("mtcars")

DT <- as.data.table(mtcars)

# ============================================================================
# Simple dplyr-like transformations

DT %>%
    group_by(cyl) %>%
    filter(vs == 0, am == 1) %>%
    transmute(mean_mpg = mean(mpg)) %>%
    arrange(-cyl)

# Equivalent to previous
DT %>%
    start_expr %>%
    transmute(mean_mpg = mean(mpg)) %>%
    where(vs == 0, am == 1) %>%
    group_by(cyl) %>%
    order_by(-cyl) %>%
    end_expr
```

```
# Modification by reference
DT %>%
    where(gear %% 2 != 0, carb %% 2 == 0) %>%
    mutate(wt_squared = wt ^ 2)

print(DT)

# Deletion by reference
DT %>%
    mutate(wt_squared = NULL) %>%
    print

# Support for tidyslect helpers

DT %>%
    select(ends_with("t"))

# ==============================================================================
# Helpers to transform a subset of data

# Like DT[, (whole) := lapply(.SD, as.integer), .SDcols = whole]
whole <- names(DT)[sapply(DT, function(x) { all(x %% 1 == 0) })]
DT %>%
    mutate_sd(as.integer, .SDcols = whole)

sapply(DT, class)

# Like DT[, lapply(.SD, fun), .SDcols = ...]
DT %>%
    transmute_sd((.COL - mean(.COL)) / sd(.COL),
                 .SDcols = setdiff(names(DT), whole))

# Filter several with the same condition
DT %>%
    filter_sd(.COL == 1, .SDcols = c("vs", "am"))

# Using secondary indices, i.e. DT[.(4, 5), on = .(cyl, gear)]
DT %>%
    filter_on(cyl = 4, gear = 5) # note we don't use ==

scale_undim <- function(...) {
    as.numeric(scale(...)) # remove dimensions
}

# Chaining
DT %>%
    start_expr %>%
    mutate_sd(as.integer, .SDcols = whole) %>%
    chain %>%
    filter_sd(.COL == 1, .SDcols = c("vs", "am"), .collapse = `|`) %>%
    transmute_sd(scale_undim, .SDcols = !is.integer(.COL)) %>%
    end_expr
```

```
# The previous is quivalent to
DT[, (whole) := lapply(.SD, as.integer), .SDcols = whole
   ][vs == 1 | am == 1,
     lapply(.SD, scale_undim),
     .SDcols = names(DT)[sapply(DT, Negate(is.integer))]]

# Alternative to keep all columns (*copying* non-scaled ones)
scale_non_integers <- function(x) {
    if (is.integer(x)) x else scale_undim(x)
}

DT %>%
    filter_sd(.COL == 1, .SDcols = c("vs", "am"), .collapse = `|`) %>%
    transmute_sd(everything(), scale_non_integers)

# Without copying non-scaled
DT %>%
    where(vs == 1 | am == 1) %>%
    mutate_sd(scale, .SDcols = names(DT)[sapply(DT, Negate(is.integer))])

print(DT)
```

`arrange-table.express` *Arrange rows*

Description

Alias for order_by-table.express.

Usage

```
## S3 method for class 'ExprBuilder'
arrange(.data, ...)

## S3 method for class 'data.table'
arrange(.data, ...)
```

Arguments

`.data`	An instance of ExprBuilder.
`...`	See order_by-table.express.

Details

To see more examples, check the vignette, or the table.express-package entry.

chain *Chain*

Description

Build a chain of similar objects/operations.

Usage

```
chain(.data, ...)

## S3 method for class 'ExprBuilder'
chain(.data, ..., .parent_env = rlang::caller_env())
```

Arguments

.data	Object to be chained.
...	Arguments for the specific methods.
.parent_env	See end_expr().

Details

The chaining for ExprBuilder is equivalent to calling `end_expr()` followed by `start_expr()`. The ellipsis (`...`) is passed to both functions.

To see more examples, check the vignette, or the table.express-package entry.

distinct-table.express

Rows with distinct combinations of columns

Description

Rows with distinct combinations of columns

Usage

```
## S3 method for class 'ExprBuilder'
distinct(
  .data,
  ...,
  .keep = TRUE,
  .n = 1L,
  .parse = getOption("table.express.parse", FALSE)
)

## S3 method for class 'data.table'
distinct(.data, ...)
```

Arguments

.data	An instance of ExprBuilder.
...	Which columns to use to determine uniqueness.
.keep	See details below.
.n	Indices of rows to return *for each* unique combination of the chosen columns. See details.
.parse	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.

Details

If `.keep = TRUE` (the default), the columns not mentioned in `...` are also kept. However, if a new column is created in one of the expressions therein, `.keep` can also be set to a character vector containing the names of *all* the columns that should be in the result in addition to the ones mentioned in `....` See the examples.

The value of `.n` is only relevant when `.keep` is *not* `FALSE`. It is used to subset `.SD` in the built `data.table` expression. For example, we could get 2 rows per combination by setting `.n` to `1:2`, or get the last row instead of the first by using `.N`. If more than one index is used, and not enough rows are found, some rows will have `NA`. Do note that, at least as of version 1.12.2 of `data.table`, only expressions with single indices are internally optimized.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

# compare with .keep = TRUE
data.table::as.data.table(mtcars) %>%
    distinct(amvs = am + vs, .keep = names(mtcars))
```

EagerExprBuilder *Eager frame expression builder*

Description

Like ExprBuilder, but eager in some regards. This shouldn't be used directly.

Super class

`table.express::ExprBuilder` -> `EagerExprBuilder`

Methods

Public methods:

- EagerExprBuilder$new()
- EagerExprBuilder$chain()
- EagerExprBuilder$chain_if_set()
- EagerExprBuilder$clone()

Method `new()`**:** Constructor.

Usage:

```
EagerExprBuilder$new(DT, ...)
```

Arguments:

`DT` A data.table::data.table.

`...` Ignored.

Method `chain()`**:** Override to abort if chaining is attempted.

Usage:

```
EagerExprBuilder$chain(...)
```

Arguments:

`...` Ignored.

Method `chain_if_set()`**:** Override to abort if chaining is attempted.

Usage:

```
EagerExprBuilder$chain_if_set(...)
```

Arguments:

`...` Ignored.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EagerExprBuilder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`end_expr` *End and evaluate expression*

Description

Finish the expression-building process and evaluate it.

Usage

```
end_expr(.data, ...)

## S3 method for class 'ExprBuilder'
end_expr(.data, ..., .by_ref = TRUE, .parent_env)
```

Arguments

`.data`	The expression.
`...`	Arguments for the specific methods.
`.by_ref`	If `FALSE`, `data.table::copy()` is used before evaluation.
`.parent_env`	Optionally, the *enclosing* environment of the expression's evaluation environment. Defaults to the caller environment.

Details

The ExprBuilder method returns a data.table::data.table.

To see more examples, check the vignette, or the table.express-package entry.

`ExprBuilder` *Frame expression builder*

Description

Build an expression that will be used inside a data.table::data.table's frame. This shouldn't be used directly.

Value

In general, a modified `self` with extended expression.

Active bindings

`appends` Extra expressions that go at the end.

`expr` The final expression that can be evaluated with `base::eval()` or `rlang::eval_bare()`.

Methods

Public methods:

- `ExprBuilder$new()`
- `ExprBuilder$set_i()`
- `ExprBuilder$set_j()`
- `ExprBuilder$set_by()`
- `ExprBuilder$chain()`
- `ExprBuilder$chain_if_set()`

- ExprBuilder$seek_and_nestroy()
- ExprBuilder$eval()
- ExprBuilder$tidy_select()
- ExprBuilder$print()
- ExprBuilder$clone()

Method `new()`**:** Constructor.

Usage:

```
ExprBuilder$new(
  DT,
  dt_pronouns = list(),
  nested = list(),
  verbose = getOption("table.express.verbose", FALSE)
)
```

Arguments:

`DT` A data.table::data.table.

`dt_pronouns, nested` Internal parameters for joins.

`verbose` Print more information during the process of building expressions.

Method `set_i()`**:** Set the `i` clause expression(s), starting a new frame if the current one already has said expression set.

Usage:

```
ExprBuilder$set_i(value, chain_if_needed)
```

Arguments:

`value` A captured expression.

`chain_if_needed` Whether chaining is allowed during this step.

Method `set_j()`**:** Like `set_i` but for the `j` clause.

Usage:

```
ExprBuilder$set_j(value, chain_if_needed)
```

Arguments:

`value` A captured expression.

`chain_if_needed` Whether chaining is allowed during this step.

Method `set_by()`**:** Set the by clause expression.

Usage:

```
ExprBuilder$set_by(value, chain_if_needed)
```

Arguments:

`value` A captured expression.

`chain_if_needed` Whether chaining is allowed during this step.

Method `chain()`**:** By default, start a new expression with the current one as its parent. If `type = "pronoun"`, `dt` is used to start a new expression that joins the current one.

Usage:

```
ExprBuilder$chain(type = "frame", next_dt, parent_env, to_eager = FALSE)
```

Arguments:

`type` One of "frame", "pronoun".

`next_dt` Next data table when chaining pronoun.

`parent_env` Where to evaluate current expression when chaining pronoun.

`to_eager` Whether or not to use an EagerExprBuilder in the new chain

Method `chain_if_set()`**:** Chain if any clause values are already set.

Usage:

```
ExprBuilder$chain_if_set(...)
```

Arguments:

`...` Clause values.

Method `seek_and_nestroy()`**:** Helper for `nest_expr`.

Usage:

```
ExprBuilder$seek_and_nestroy(.exprs)
```

Arguments:

`.exprs` List of expressions.

Method `eval()`**:** Evaluate the final expression with `parent_env` as the enclosing environment. If `by_ref = FALSE`, `data.table::copy()` is called before. The ellipsis' contents are assigned to the expression's evaluation environment.

Usage:

```
ExprBuilder$eval(parent_env, by_ref, ...)
```

Arguments:

`parent_env` Enclosing environment.

`by_ref` Flag to control deep copies.

`...` Additional variables for the evaluation environment.

Method `tidy_select()`**:** Evaluate a `tidyselect` call using the currently captured table.

Usage:

```
ExprBuilder$tidy_select(select_expr)
```

Arguments:

`select_expr` The selection expression.

Method `print()`**:** Prints the built `expr`.

Usage:

```
ExprBuilder$print(...)
```

Arguments:

`...` Ignored.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ExprBuilder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

extrema_by *Find rows with extrema in specific columns*

Description

Find rows with maxima/minima in given columns.

Usage

```
max_by(.data, .col, ...)

## S3 method for class 'ExprBuilder'
max_by(
  .data,
  .col,
  ...,
  .some = FALSE,
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
max_by(.data, .col, ..., .expr = FALSE)

min_by(.data, .col, ...)

## S3 method for class 'ExprBuilder'
min_by(
  .data,
  .col,
  ...,
  .some = FALSE,
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
min_by(.data, .col, ..., .expr = FALSE)
```

Arguments

.data	An instance of ExprBuilder.
.col	A character vector indicating the columns that will be searched for extrema.
...	Optionally, columns to group by, either as characters or symbols.
.some	If TRUE the rows where *any* of the columns specified in .col have extrema are returned.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?

`.expr` If the input is a `data.table` and `.expr` is `TRUE`, an instance of EagerExprBuilder will be returned. Useful if you want to add clauses to `j`, e.g. with mutate-table.express.

Details

These verbs implement the idiom shown here by leveraging `nest_expr()`. The whole nested expression is assigned to `i` in the `data.table`'s frame. It is probably a good idea to use this on a frame that has no other frames preceding it in the current expression, given that `nest_expr()` uses the captured `data.table`, so consider using `chain()` when needed.

Several columns can be specified in `.col`, and depending on the value of `.some`, the rows with all or some extrema are returned, using `&` or `|` respectively. Depending on your data, using more than one column might not make sense, resulting in an empty `data.table`.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    max_by("mpg", "vs")
```

`filter-table.express` *Filter rows*

Description

Filter rows

Usage

```
## S3 method for class 'ExprBuilder'
filter(.data, ..., .preserve)

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

Arguments

`.data`	An instance of ExprBuilder.
`...`	See where-table.express.
`.preserve`	Ignored.

Details

The ExprBuilder method is an alias for where-table.express.

The data.table::data.table method works eagerly like `dplyr::filter()`.

To see more examples, check the vignette, or the table.express-package entry.

`filter_on` *Filter with secondary indices*

Description

Helper to filter specifying the on part of the data.table::data.table query.

Usage

```
filter_on(.data, ...)

## S3 method for class 'ExprBuilder'
filter_on(
  .data,
  ...,
  which = FALSE,
  nomatch = getOption("datatable.nomatch"),
  mult = "all",
  .negate = FALSE,
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
filter_on(.data, ..., .expr = FALSE)
```

Arguments

`.data`	An instance of ExprBuilder.
`...`	Key-value pairs, maybe with empty keys if the `data.table` already has them. See details.
`which, nomatch, mult`	See data.table::data.table.
`.negate`	Whether to negate the expression and search only for rows that don't contain the given values.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
`.expr`	If the input is a `data.table` and `.expr` is `TRUE`, an instance of EagerExprBuilder will be returned. Useful if you want to add clauses to `j`, e.g. with mutate-table.express.

Details

The key-value pairs in '...' are processed as follows:

- The names are used as `on` in the `data.table` frame. If any name is empty, `on` is left missing.
- The values are packed in a list and used as `i` in the `data.table` frame.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    filter_on(cyl = 4, gear = 5)
```

`filter_sd` *Filter subset of data*

Description

Helper to filter rows with the same condition applied to a subset of the data.

Usage

```
filter_sd(.data, .SDcols, .how = Negate(is.na), ...)

## S3 method for class 'ExprBuilder'
filter_sd(
  .data,
  .SDcols,
  .how = Negate(is.na),
  ...,
  which,
  .collapse = `&`,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE),
  .caller_env_n = 1L
)

## S3 method for class 'data.table'
filter_sd(.data, ..., .expr = FALSE)
```

Arguments

.data	An instance of ExprBuilder.
.SDcols	See data.table::data.table and the details here.
.how	The filtering function or predicate.
...	Possibly more arguments for .how.
which	Passed to data.table::data.table.
.collapse	See where-table.express.
.parse	Logical. Whether to apply rlang::parse_expr() to obtain the expressions.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
.caller_env_n	Internal. Passed to rlang::caller_env() to find the function specified in .how and standardize its call.
.expr	If the input is a data.table and .expr is TRUE, an instance of EagerExprBuilder will be returned. Useful if you want to add clauses to j, e.g. with mutate-table.express.

Details

This function adds/chains an i expression that will be evaluated by data.table::data.table, and it supports the .COL pronoun and lambdas as formulas. The .how condition is applied to all .SDcols.

Additionally, .SDcols supports:

- tidyselect::select_helpers
- A predicate using the .COL pronoun that should return a single logical when .COL is replaced by a *column* of the data.
- A formula using . or .x instead of the aforementioned .COL.

The caveat is that the expression is evaluated eagerly, i.e. with the currently captured data.table. Consider using chain() to explicitly capture intermediate results as actual data.tables.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    filter_sd(c("vs", "am"), ~ .x == 1)
```

frame_append *Append expressions to the frame*

Description

Add named expressions for the data.table::data.table frame.

Usage

```
frame_append(.data, ..., .parse = getOption("table.express.parse", FALSE))
```

Arguments

.data	An instance of ExprBuilder.
...	Expressions to add to the frame.
.parse	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.

Examples

```
data.table::data.table() %>%
    start_expr %>%
    frame_append(anything = "goes")
```

group_by-table.express *Grouping clauses*

Description

Grouping by columns of a data.table::data.table.

Usage

```
## S3 method for class 'ExprBuilder'
group_by(
  .data,
  ...,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
group_by(.data, ...)
```

Arguments

.data	An instance of ExprBuilder.
...	Clause for grouping on columns. The by inside the data.table's frame.
.parse	Logical. Whether to apply rlang::parse_expr() to obtain the expressions.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?

Details

Everything in ... will be wrapped in a call to list.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    start_expr %>%
    group_by(cyl, gear)
```

---

joins *Joining verbs*

Description

Two-table joins. Check the "Joining verbs" vignette for more information.

Usage

```
## S3 method for class 'ExprBuilder'
anti_join(x, y, ...)

## S3 method for class 'data.table'
anti_join(x, ..., .expr = FALSE)

## S3 method for class 'ExprBuilder'
full_join(x, y, ..., sort = TRUE, allow = TRUE, .parent_env)

## S3 method for class 'data.table'
full_join(x, ...)

## S3 method for class 'ExprBuilder'
inner_join(x, y, ...)
```

```
## S3 method for class 'data.table'
inner_join(x, ..., .expr = FALSE)

## S3 method for class 'ExprBuilder'
left_join(
  x,
  y,
  ...,
  nomatch,
  mult,
  roll,
  rollends,
  .parent_env,
  .to_eager = FALSE
)

## S3 method for class 'data.table'
left_join(x, y, ..., allow = FALSE, .expr = FALSE)

mutate_join(x, y, ...)

## S3 method for class 'ExprBuilder'
mutate_join(
  x,
  y,
  ...,
  .SDcols,
  mult,
  roll,
  rollends,
  allow = FALSE,
  .by_each = NULL,
  .parent_env
)

## S3 method for class 'EagerExprBuilder'
mutate_join(x, ..., .parent_env = rlang::caller_env())

## S3 method for class 'data.table'
mutate_join(x, y, ...)

## S3 method for class 'ExprBuilder'
right_join(
  x,
  y,
  ...,
  allow = FALSE,
  which,
```

```
  nomatch,
  mult,
  roll,
  rollends,
  .selecting,
  .framing
)

## S3 method for class 'data.table'
right_join(x, y, ..., allow = FALSE, .expr = FALSE, .selecting, .framing)

## S3 method for class 'ExprBuilder'
semi_join(x, y, ..., allow = FALSE, .eager = FALSE)

## S3 method for class 'data.table'
semi_join(x, y, ..., allow = FALSE, .eager = FALSE)
```

Arguments

x	An ExprBuilder instance.
y	A data.table::data.table or, for some verbs (see details), a call to `nest_expr()`.
...	Expressions for the on part of the join.
.expr	If the input is a `data.table` and `.expr` is `TRUE`, an instance of EagerExprBuilder will be returned. Useful if you want to add clauses to `j`, e.g. with mutate-table.express.
sort	Passed to data.table::merge.
allow	Passed as `data.table`'s `allow.cartesian`.
.parent_env	See `end_expr()`.
nomatch, mult, roll, rollends	See data.table::data.table.
.to_eager	Internal, should be left as `FALSE` in all external calls.
.SDcols	For `mutate_join`. See the details below.
.by_each	For `mutate_join`. See the details below.
which	If `TRUE`, return the row numbers that matched in `x` instead of the result of the join.
.selecting	One or more expressions, possibly contained in a call to `list` or `.`, that will be added to `j` in the same frame as the join.
.framing	Similar to `.selecting`, but added to the frame with `frame_append()`.
.eager	For `semi_join`. If `TRUE`, it uses `nest_expr()` to build an expression like this instead of the default one. This uses the captured `data.table` eagerly, so use `chain()` when needed. The default is lazy.

Details

The following joins support `nest_expr()` in `y`:

- `anti_join`
- `inner_join`
- `right_join`

The `full_join` method is really a wrapper for `data.table::merge` that specifies `all = TRUE`. The expression in `x` gets evaluated, merged with `y`, and the result is captured in a new ExprBuilder. Useful in case you want to keep building expressions after the merge.

Mutating join

The ExprBuilder method for `mutate_join` implements the idiom described in this link. The columns specified in `.SDcols` are those that will be added to `x` from `y`. The specification can be done by:

- Using tidyselect::select_helpers.
- Passing a character vector. If the character is named, the names are taken as the new column names for the values added to `x`.
- A list, using `base::list()` or `.()`, containing:
  - Column names, either as characters or symbols.
  - Named calls expressing how the column should be summarized/modified before adding it to `x`.

The last case mentioned above is useful when the join returns many rows from `y` for each row in `x`, so they can be summarized while joining. The value of `by` in the join depends on what is passed to `.by_each`:

- If `NULL` (the default), `by` is set to `.EACHI` if a call is detected in any of the expressions from the list in `.SDcols`
- If `TRUE`, `by` is always set to `.EACHI`
- If `FALSE`, `by` is never set to `.EACHI`

See Also

data.table::data.table, dplyr::join

Examples

```
lhs <- data.table::data.table(x = rep(c("b", "a", "c"), each = 3),
                              y = c(1, 3, 6),
                              v = 1:9)

rhs <- data.table::data.table(x = c("c", "b"),
                              v = 8:7,
                              foo = c(4, 2))
```

```
rhs %>%
    anti_join(lhs, x, v)


lhs %>%
    inner_join(rhs, x)


# creates new data.table
lhs %>%
    left_join(rhs, x)


# would modify lhs by reference
lhs %>%
    start_expr %>%
    mutate_join(rhs, x, .SDcols = c("foo", rhs.v = "v"))

# would modify rhs by reference, summarizing 'y' before adding it.
rhs %>%
    start_expr %>%
    mutate_join(lhs, x, .SDcols = .(y = mean(y)))


# creates new data.table
lhs %>%
    right_join(rhs, x)


# keep only columns from lhs
lhs %>%
    semi_join(rhs, x)
```

---

key_by *Set key to group by*

Description

Group by setting key of the input.

Usage

```
key_by(.data, ...)

## S3 method for class 'ExprBuilder'
key_by(
  .data,
  ...,
  .parse = getOption("table.express.parse", FALSE),
```

```
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
key_by(.data, ...)
```

Arguments

.data	Object to be grouped and subsequently keyed.
...	Arguments for the specific methods.
.parse	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?

Details

Everything in `...` will be wrapped in a call to `list`. Its contents work like Clauses for grouping on columns. The `keyby` inside the data.table::data.table frame.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    start_expr %>%
    key_by(cyl, gear)
```

---

mutate-table.express *Add or update columns*

Description

Add or update columns of a data.table::data.table, possibly by reference using `:=`.

Usage

```
## S3 method for class 'ExprBuilder'
mutate(
  .data,
  ...,
  .sequential = FALSE,
  .unquote_names = TRUE,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
```

```
)

## S3 method for class 'EagerExprBuilder'
mutate(.data, ..., .parent_env = rlang::caller_env())

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

Arguments

.data	An instance of ExprBuilder.
...	Mutation clauses.
.sequential	If TRUE, each expression in ... is assigned to a nested body within curly braces to allow them to use variables created by previous expressions. The default is FALSE because enabling this may turn off some data.table optimizations.
.unquote_names	Passed to rlang::enexprs(). Set to FALSE if you want to pass the single := expression.
.parse	Logical. Whether to apply rlang::parse_expr() to obtain the expressions.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
.parent_env	See end_expr()

Details

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")
data.table::as.data.table(mtcars) %>%
    start_expr %>%
    mutate(mpg_squared = mpg ^ 2)
```

---

mutate_sd *Mutate subset of data*

Description

Like mutate-table.express but possibly recycling calls.

Usage

```
mutate_sd(.data, .SDcols, .how = identity, ...)

## S3 method for class 'ExprBuilder'
mutate_sd(
  .data,
  .SDcols,
  .how = identity,
  ...,
  .pairwise = TRUE,
  .prefix,
  .suffix,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'EagerExprBuilder'
mutate_sd(.data, ..., .parent_env = rlang::caller_env())

## S3 method for class 'data.table'
mutate_sd(.data, ...)
```

Arguments

`.data`	An instance of ExprBuilder.
`.SDcols`	See data.table::data.table and the details here.
`.how`	The function(s) or function call(s) that will perform the transformation. If many, a list should be used, either with `list()` or `.()`. If the list is named, the names will be used for the new columns' names. Lambdas specified as formulas are supported.
`...`	Possibly more arguments for *all* functions/calls in `.how`.
`.pairwise`	If `FALSE`, each function in `.how` is applied to each column in `.SDcols` (like a cartesian product).
`.prefix, .suffix`	Only relevant when `.how` is a function: add a prefix or suffix to the new column's name. If neither is missing, `.prefix` has preference.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
`.parent_env`	See `end_expr()`

Details

This function works similar to `transmute_sd()` but keeps all columns and *can* modify by reference, like mutate-table.express. It can serve like dplyr's scoped mutation variants depending on what's given to `.SDcols`.

Additionally, `.SDcols` supports:

- tidyselect::select_helpers
- A predicate using the `.COL` pronoun that should return a single logical when `.COL` is replaced by a *column* of the data.
- A formula using `.` or `.x` instead of the aforementioned `.COL`.

The caveat is that the expression is evaluated eagerly, i.e. with the currently captured `data.table`. Consider using `chain()` to explicitly capture intermediate results as actual `data.tables`.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    start_expr %>%
    mutate_sd(c("mpg", "cyl"), ~ .x * 2)
```

`nest_expr` *Nest expressions as a functional chain*

Description

Nest expressions as a functional chain

Usage

```
nest_expr(
  ...,
  .start = TRUE,
  .end = .start,
  .parse = getOption("table.express.parse", FALSE)
)
```

Arguments

`...`	Expressions that will be part of the functional chain.
`.start`	Whether to add a `start_expr()` call at the beginning of the chain.
`.end`	Whether to add an `end_expr()` call at the end of the chain.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.

Details

All expressions in `...` are "collapsed" with `%>%`, passing the ExprBuilder's captured `data.table` as the initial parameter. Names are silently dropped.

The chain is evaluated eagerly and saved in the `ExprBuilder` instance to be used during final expression evaluation.

To see more examples, check the vignette, or the table.express-package entry.

`order_by-table.express` *Order by clause*

Description

Clause for ordering rows.

Usage

```
order_by(.data, ...)

## S3 method for class 'ExprBuilder'
order_by(
  .data,
  ...,
  .collapse,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
order_by(.data, ...)
```

Arguments

`.data`	The input data.
`...`	Arguments for the specific methods.
`.collapse`	Ignored. See details.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?

Details

The ExprBuilder method dispatches to where-table.express, but doesn't forward the `.collapse` argument.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    order_by(-cyl, gear)
```

`select-table.express` *Select clause*

Description

Select columns of a data.table::data.table.

Usage

```
## S3 method for class 'ExprBuilder'
select(
  .data,
  ...,
  .negate = FALSE,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'EagerExprBuilder'
select(.data, ..., .parent_env = rlang::caller_env())

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

Arguments

`.data`	An instance of ExprBuilder.
`...`	Clause for selecting columns. For `j` inside the `data.table`'s frame.
`.negate`	Whether to negate the selection semantics and keep only columns that do *not* match what's given in `...`.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
`.parent_env`	See `end_expr()`

Details

The expressions in `...` support tidyselect::select_helpers.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    select(mpg:cyl)
```

---

start_expr *Start expression*

Description

Start building an expression.

Usage

```
start_expr(.data, ...)

## S3 method for class 'data.table'
start_expr(.data, ..., .verbose = getOption("table.express.verbose", FALSE))
```

Arguments

.data	Optionally, something to capture for the expression.
...	Arguments for the specific methods.
.verbose	Whether to print more information during the expression-building process.

Details

The data.table::data.table method returns an ExprBuilder instance.

To see more examples, check the vignette, or the table.express-package entry.

---

summarize-table.express *Summarize columns*

Description

Compute summaries for columns, perhaps by group.

Usage

```
## S3 method for class 'ExprBuilder'
summarize(
  .data,
  ...,
  .assume_optimized = NULL,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'ExprBuilder'
summarise(
  .data,
  ...,
  .assume_optimized = NULL,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'EagerExprBuilder'
summarize(.data, ..., .parent_env = rlang::caller_env())

## S3 method for class 'EagerExprBuilder'
summarise(.data, ..., .parent_env = rlang::caller_env())

## S3 method for class 'data.table'
summarize(.data, ...)

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

Arguments

`.data`	An instance of ExprBuilder.
`...`	Clauses for transmuting columns. For `j` inside the `data.table`'s frame.
`.assume_optimized`	An optional character vector with function names that you know `data.table` can optimize. This will be added to this set of known names: min, max, mean, median, var, sd, sum, prod, first, last. Note that using those functions (and only those in a given call to this function) will prevent the expressions from using variables created by previous expressions.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
`.parent_env`	See `end_expr()`

Details

The built expression is similar to what `transmute` builds, but the function also checks that the results have length 1.

To see more examples, check the vignette, or the table.express-package entry.

`transmute-table.express`

Compute new columns

Description

Compute and keep only new columns.

Usage

```
## S3 method for class 'ExprBuilder'
transmute(
  .data,
  ...,
  .enlist = TRUE,
  .sequential = FALSE,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'EagerExprBuilder'
transmute(.data, ..., .parent_env = rlang::caller_env())

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

Arguments

`.data`	An instance of ExprBuilder.
`...`	Clauses for transmuting columns. For `j` inside the `data.table`'s frame.
`.enlist`	See details.
`.sequential`	If `TRUE`, each expression in `...` is assigned to a nested body within curly braces to allow them to use variables created by previous expressions. The default is `FALSE` because enabling this may turn off some data.table optimizations.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
`.parent_env`	See `end_expr()`

Details

Everything in `...` is wrapped in a call to `list` by default. If only one expression is given, you can set `.enlist` to `FALSE` to skip the call to `list`.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    transmute(ans = mpg * 2)
```

`transmute_sd` *Transmute subset of data*

Description

Like transmute-table.express but for a single call and maybe specifying `.SDcols`.

Usage

```
transmute_sd(.data, .SDcols = everything(), .how = identity, ...)

## S3 method for class 'ExprBuilder'
transmute_sd(
  .data,
  .SDcols = everything(),
  .how = identity,
  ...,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'EagerExprBuilder'
transmute_sd(.data, ..., .parent_env = rlang::caller_env())

## S3 method for class 'data.table'
transmute_sd(.data, ...)
```

Arguments

`.data`	An instance of ExprBuilder.
`.SDcols`	See data.table::data.table and the details here.

`.how`	The function(s) or function call(s) that will perform the transformation. If many, a list should be used, either with `list()` or `.()`. If the list is named, the names will be used for the new columns' names. Lambdas specified as formulas are supported.
`...`	Possibly more arguments for *all* functions/calls in `.how`.
`.parse`	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
`.chain`	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
`.parent_env`	See `end_expr()`

Details

Like transmute-table.express, this function never modifies the input by reference. This function adds/chains a `select` expression that will be evaluated by data.table::data.table, possibly specifying the helper function `.transmute_matching`, which is assigned to the final expression's evaluation environment when calling `end_expr()` (i.e., ExprBuilder's `eval` method).

Said function supports two pronouns that can be used by `.how` and `.SDcols`:

- `.COL`: the actual values of the column.
- `.COLNAME`: the name of the column currently being evaluated.

Additionally, lambdas specified as formulas are also supported. In those cases, `.x` is equivalent to `.COL` and `.y` to `.COLNAME`.

Unlike a call like `DT[, (vars) := expr]`, `.SDcols` can be created dynamically with an expression that evaluates to something that would be used in place of `vars` *without* eagerly using the captured `data.table`. See the examples here or in table.express-package.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    transmute_sd(~ grepl("^d", .y), ~ .x * 2)

data.table::as.data.table(mtcars) %>%
    transmute_sd(~ is.numeric(.x), ~ .x * 2)
```

`where-table.express` *Where clause*

Description

Clause for subsetting rows.

Usage

```
where(.data, ...)

## S3 method for class 'ExprBuilder'
where(
  .data,
  ...,
  which,
  .collapse = `&`,
  .parse = getOption("table.express.parse", FALSE),
  .chain = getOption("table.express.chain", TRUE)
)

## S3 method for class 'data.table'
where(.data, ...)
```

Arguments

.data	The input data.
...	Arguments for the specific methods.
which	Passed to data.table::data.table.
.collapse	A boolean function which will be used to "concatenate" all conditions in ....
.parse	Logical. Whether to apply `rlang::parse_expr()` to obtain the expressions.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?

Details

For ExprBuilder, the expressions in ... can call `nest_expr()`, and are eagerly nested if they do.

The data.table::data.table method is **lazy**, so it expects another verb to follow *afterwards*.

To see more examples, check the vignette, or the table.express-package entry.

Examples

```
data("mtcars")

data.table::as.data.table(mtcars) %>%
    start_expr %>%
    where(vs == 0, am == 1)


data.table::as.data.table(mtcars) %>%
    where(vs == 0) %>%
    transmute(mpg = round(mpg))
```

Index