

# Package: table.express (via r-universe)

September 3, 2024

**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

**Roxygen** list(markdown = TRUE)

**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'

**Repository** <https://asardaes.r-universe.dev>

**RemoteUrl** <https://github.com/asardaes/table.express>

**RemoteRef** HEAD

**RemoteSha** e7864291f91fa68abbf7c82396d275122453106d

## Contents

table.express-package . . . . .	3
arrange-table.express . . . . .	5
chain . . . . .	6
distinct-table.express . . . . .	7
EagerExprBuilder . . . . .	8
end_expr . . . . .	9
ExprBuilder . . . . .	10
extrema_by . . . . .	12
filter-table.express . . . . .	14
filter_on . . . . .	15
filter_sd . . . . .	16
frame_append . . . . .	17
group_by-table.express . . . . .	18
joins . . . . .	19
key_by . . . . .	23
mutate-table.express . . . . .	24
mutate_sd . . . . .	25
nest_expr . . . . .	27
order_by-table.express . . . . .	27
select-table.express . . . . .	28
start_expr . . . . .	30
summarize-table.express . . . . .	30
transmute-table.express . . . . .	32
transmute_sd . . . . .	33
where-table.express . . . . .	34

**Index**

**36**

---

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 use `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 equivalent 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	<i>Chain</i>
-------	--------------

---

**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

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>...</code>	Which columns to use to determine uniqueness.
<code>.keep</code>	See details below.
<code>.n</code>	Indices of rows to return <i>for each</i> unique combination of the chosen columns. See details.
<code>.parse</code>	Logical. Whether to apply <code>rlang::parse_expr()</code> 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.exprs::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, <code>data.table::copy()</code> is used before evaluation.
.parent_env	Optionally, the <i>enclosing</i> 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**

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>.col</code>	A character vector indicating the columns that will be searched for extrema.
<code>...</code>	Optionally, columns to group by, either as characters or symbols.
<code>.some</code>	If TRUE the rows where <i>any</i> of the columns specified in <code>.col</code> have extrema are returned.
<code>.chain</code>	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
<code>.expr</code>	If the input is a <code>data.table</code> and <code>.expr</code> is TRUE, an instance of <a href="#">EagerExprBuilder</a> will be returned. Useful if you want to add clauses to <code>j</code> , e.g. with <a href="#">mutate-table.express</a> .

**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

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>...</code>	See <a href="#">where-table.express</a> .
<code>.preserve</code>	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	<i>Filter with secondary indices</i>
-----------	--------------------------------------

---

### 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

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

### 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 <a href="#">ExprBuilder</a> .
.SDcols	See <a href="#">data.table::data.table</a> and the details here.
.how	The filtering function or predicate.
...	Possibly more arguments for .how.
which	Passed to <a href="#">data.table::data.table</a> .
.collapse	See <a href="#">where-table.express</a> .
.parse	Logical. Whether to apply <a href="#">rlang::parse_expr()</a> to obtain the expressions.
.chain	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?



<code>.caller_env_n</code>	Internal. Passed to <code>rlang::caller_env()</code> to find the function specified in <code>.how</code> and <code>standardize</code> its call.
<code>.expr</code>	If the input is a <code>data.table</code> and <code>.expr</code> is <code>TRUE</code> , an instance of <code>EagerExprBuilder</code> will be returned. Useful if you want to add clauses to <code>j</code> , e.g. with <code>mutate-table.express</code> .

## 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	<i>Append expressions to the frame</i>
--------------	--

---

## Description

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

## Usage

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

## Arguments

<code>.data</code>	An instance of <code>ExprBuilder</code> .
<code>...</code>	Expressions to add to the frame.
<code>.parse</code>	Logical. Whether to apply <code>rlang::parse_expr()</code> 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**

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>...</code>	Clause for grouping on columns. The <code>by</code> inside the <code>data.table</code> 's frame.
<code>.parse</code>	Logical. Whether to apply <a href="#">rlang::parse_expr()</a> to obtain the expressions.
<code>.chain</code>	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

<code>x</code>	An <a href="#">ExprBuilder</a> instance.
<code>y</code>	A <a href="#">data.table::data.table</a> or, for some verbs (see details), a call to <a href="#">nest_expr()</a> .
<code>...</code>	Expressions for the on part of the join.
<code>.expr</code>	If the input is a <a href="#">data.table</a> and <code>.expr</code> is TRUE, an instance of <a href="#">EagerExprBuilder</a>

	will be returned. Useful if you want to add clauses to <code>j</code> , e.g. with <code>mutate_table.express</code> .
<code>sort</code>	Passed to <code>data.table::merge</code> .
<code>allow</code>	Passed as <code>data.table</code> 's <code>allow.cartesian</code> .
<code>.parent_env</code>	See <code>end_expr()</code> .
<code>nomatch, mult, roll, rollends</code>	See <code>data.table::data.table</code> .
<code>.to_eager</code>	Internal, should be left as <code>FALSE</code> in all external calls.
<code>.SDcols</code>	For <code>mutate_join</code> . See the details below.
<code>.by_each</code>	For <code>mutate_join</code> . See the details below.
<code>which</code>	If <code>TRUE</code> , return the row numbers that matched in <code>x</code> instead of the result of the join.
<code>.selecting</code>	One or more expressions, possibly contained in a call to <code>list</code> or <code>.</code> , that will be added to <code>j</code> in the same frame as the join.
<code>.framing</code>	Similar to <code>.selecting</code> , but added to the frame with <code>frame_append()</code> .
<code>.eager</code>	For <code>semi_join</code> . If <code>TRUE</code> , it uses <code>nest_expr()</code> to build an expression like <code>this</code> instead of the default one. This uses the captured <code>data.table</code> eagerly, so use <code>chain()</code> 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	<i>Set key to group by</i>
--------	----------------------------

---

### 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 <code>rlang::parse_expr()</code> 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

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>...</code>	Mutation clauses.
<code>.sequential</code>	If TRUE, each expression in <code>...</code> 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 <a href="#">data.table optimizations</a> .
<code>.unquote_names</code>	Passed to <a href="#">rlang::enexprs()</a> . Set to FALSE if you want to pass the single <code>:=</code> expression.
<code>.parse</code>	Logical. Whether to apply <a href="#">rlang::parse_expr()</a> to obtain the expressions.
<code>.chain</code>	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
<code>.parent_env</code>	See <a href="#">end_expr()</a>



**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	<i>Mutate subset of data</i>
-----------	------------------------------

---

**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.

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

## 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 <a href="#">start_expr()</a> call at the beginning of the chain.
.end	Whether to add an <a href="#">end_expr()</a> call at the end of the chain.
.parse	Logical. Whether to apply <code>rlang::parse_expr()</code> 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**

<code>.data</code>	The input data.
<code>...</code>	Arguments for the specific methods.
<code>.collapse</code>	Ignored. See details.
<code>.parse</code>	Logical. Whether to apply <code>rlang::parse_expr()</code> to obtain the expressions.
<code>.chain</code>	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**

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

**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	<i>Start expression</i>
------------	-------------------------

---

**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	<i>Summarize columns</i>
-------------------------	--------------------------

---

**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

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>...</code>	Clauses for transmuting columns. For <code>j</code> inside the <code>data.table</code> 's frame.
<code>.assume_optimized</code>	An optional character vector with function names that you know <code>data.table</code> can optimize. This will be added to this set of known names: <code>min</code> , <code>max</code> , <code>mean</code> , <code>median</code> , <code>var</code> , <code>sd</code> , <code>sum</code> , <code>prod</code> , <code>first</code> , <code>last</code> . 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.
<code>.parse</code>	Logical. Whether to apply <code>rlang::parse_expr()</code> to obtain the expressions.
<code>.chain</code>	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
<code>.parent_env</code>	See <a href="#">end_expr()</a>

## 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

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>...</code>	Clauses for transmuting columns. For <code>j</code> inside the <code>data.table</code> 's frame.
<code>.enlist</code>	See details.
<code>.sequential</code>	If TRUE, each expression in <code>...</code> 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 <a href="#">data.table optimizations</a> .
<code>.parse</code>	Logical. Whether to apply <code>rlang::parse_expr()</code> to obtain the expressions.
<code>.chain</code>	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
<code>.parent_env</code>	See <a href="#">end_expr()</a>

## 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	<i>Transmute subset of data</i>
--------------	---------------------------------

---

**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**

<code>.data</code>	An instance of <a href="#">ExprBuilder</a> .
<code>.SDcols</code>	See <a href="#">data.table::data.table</a> and the details here.
<code>.how</code>	The function(s) or function call(s) that will perform the transformation. If many, a list should be used, either with <code>list()</code> or <code>.</code> . If the list is named, the names will be used for the new columns' names. Lambdas specified as formulas are supported.
<code>...</code>	Possibly more arguments for <i>all</i> functions/calls in <code>.how</code> .
<code>.parse</code>	Logical. Whether to apply <a href="#">rlang::parse_expr()</a> to obtain the expressions.
<code>.chain</code>	Logical. Should a new frame be automatically chained to the expression if the clause being set already exists?
<code>.parent_env</code>	See <a href="#">end_expr()</a>

## 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 <a href="#">data.table::data.table</a> .
.collapse	A boolean function which will be used to "concatenate" all conditions in ...
.parse	Logical. Whether to apply <a href="#">rlang::parse_expr()</a> 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

`:=`, 24  
`%>%`, 27

`anti_join.data.table` (joins), 19  
`anti_join.ExprBuilder` (joins), 19  
`arrange-table.express`, 5  
`arrange.data.table`  
    (`arrange-table.express`), 5  
`arrange.ExprBuilder`  
    (`arrange-table.express`), 5

`base::eval()`, 10  
`base::list()`, 21

`chain`, 6  
`chain()`, 14, 17, 21, 26

`data.table`, 3, 21  
`data.table` optimizations, 24, 32  
`data.table::copy()`, 9, 12  
`data.table::data.table`, 8–10, 14–18, 20–25, 28, 30, 33–35  
`data.table::merge`, 21  
`distinct-table.express`, 7  
`distinct.data.table`  
    (`distinct-table.express`), 7  
`distinct.ExprBuilder`  
    (`distinct-table.express`), 7

`dplyr`, 3  
`dplyr`'s scoped mutation variants, 26  
`dplyr::filter()`, 14  
`dplyr::join`, 22

`EagerExprBuilder`, 8, 11, 13, 15, 17, 20  
`end_expr`, 9  
`end_expr()`, 6, 21, 24, 26, 27, 29, 31–34  
`ExprBuilder`, 6–9, 10, 13–18, 20, 21, 24, 25, 27–35  
`extrema_by`, 12

`filter-table.express`, 14

`filter.data.table`  
    (`filter-table.express`), 14  
`filter.ExprBuilder`  
    (`filter-table.express`), 14  
`filter_on`, 15  
`filter_sd`, 16  
`frame_append`, 17  
`frame_append()`, 21  
`full_join.data.table` (joins), 19  
`full_join.ExprBuilder` (joins), 19

`group_by-table.express`, 18  
`group_by.data.table`  
    (`group_by-table.express`), 18  
`group_by.ExprBuilder`  
    (`group_by-table.express`), 18

`inner_join.data.table` (joins), 19  
`inner_join.ExprBuilder` (joins), 19

joins, 19

`key_by`, 23

`left_join.data.table` (joins), 19  
`left_join.ExprBuilder` (joins), 19

`max_by` (`extrema_by`), 12  
`min_by` (`extrema_by`), 12  
`mutate-table.express`, 13, 15, 17, 21, 24, 25, 26  
`mutate.data.table`  
    (`mutate-table.express`), 24  
`mutate.EagerExprBuilder`  
    (`mutate-table.express`), 24  
`mutate.ExprBuilder`  
    (`mutate-table.express`), 24  
`mutate_join` (joins), 19  
`mutate_sd`, 25

`nest_expr`, 27

- nest\_expr(), [13](#), [14](#), [20](#), [21](#), [35](#)
- order\_by (order\_by-table.express), [27](#)
- order\_by-table.express, [5](#), [6](#), [27](#)
- order\_by.data.table
  - (order\_by-table.express), [27](#)
- order\_by.ExprBuilder
  - (order\_by-table.express), [27](#)
- right\_join.data.table (joins), [19](#)
- right\_join.ExprBuilder (joins), [19](#)
- rlang::caller\_env(), [17](#)
- rlang::enexprs(), [24](#)
- rlang::eval\_bare(), [10](#)
- rlang::parse\_expr(), [7](#), [16–18](#), [23](#), [24](#), [26–29](#), [31–33](#), [35](#)
- select-table.express, [28](#)
- select.data.table
  - (select-table.express), [28](#)
- select.EagerExprBuilder
  - (select-table.express), [28](#)
- select.ExprBuilder
  - (select-table.express), [28](#)
- semi\_join.data.table (joins), [19](#)
- semi\_join.ExprBuilder (joins), [19](#)
- standardize, [17](#)
- start\_expr, [30](#)
- start\_expr(), [6](#), [27](#)
- summarise.data.table
  - (summarize-table.express), [30](#)
- summarise.EagerExprBuilder
  - (summarize-table.express), [30](#)
- summarise.ExprBuilder
  - (summarize-table.express), [30](#)
- summarize-table.express, [30](#)
- summarize.data.table
  - (summarize-table.express), [30](#)
- summarize.EagerExprBuilder
  - (summarize-table.express), [30](#)
- summarize.ExprBuilder
  - (summarize-table.express), [30](#)
- table.express (table.express-package), [3](#)
- table.express-package, [3](#), [6](#), [7](#), [9](#), [14](#), [15](#), [17](#), [18](#), [23](#), [25–32](#), [34](#), [35](#)
- table.express::ExprBuilder, [8](#)
- tidyselect::select\_helpers, [17](#), [21](#), [26](#), [29](#)
- transmute-table.express, [32](#), [33](#), [34](#)
- transmute.data.table
  - (transmute-table.express), [32](#)
- transmute.EagerExprBuilder
  - (transmute-table.express), [32](#)
- transmute.ExprBuilder
  - (transmute-table.express), [32](#)
- transmute\_sd, [33](#)
- transmute\_sd(), [26](#)
- where (where-table.express), [34](#)
- where-table.express, [14](#), [16](#), [28](#), [34](#)
- where.data.table (where-table.express), [34](#)
- where.ExprBuilder
  - (where-table.express), [34](#)