

Package ‘fjoin’

May 8, 2026

Type Package

Title Data Frame Joins Leveraging 'data.table'

Version 0.1.1

Description Extends 'data.table' join functionality, lets it work with any data frame class, and provides a familiar 'x'/y'-style interface, enabling broad use across R. Offers NA-safe matching by default, on-the-fly column selection, multiple match-handling on both sides, 'x' or 'y' row order, and a row origin indicator. Performs inner, left, right, full, semi- and anti-joins with equality and inequality conditions, plus cross joins. Specific support for 'data.table', (grouped) tibble, and 'sf'/sfc' objects and their attributes; returns a plain data frame otherwise. Avoids data-copying of inputs and outputs. Allows displaying the 'data.table' code instead of (or as well as) executing it.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 3.3.0)

Imports data.table

Suggests dplyr (>= 1.1.0), sf, testthat (>= 3.0.0), knitr, rmarkdown, quarto, bench, ggplot2

URL <https://trobx.github.io/fjoin/>

BugReports <https://github.com/trobx/fjoin/issues>

Config/testthat/edition 3

VignetteBuilder knitr, quarto

RoxygenNote 7.3.3

NeedsCompilation no

Author Toby Robertson [aut, cre]

Maintainer Toby Robertson <trobx@proton.me>

Repository CRAN

Date/Publication 2026-03-18 11:40:02 UTC

Contents

dtjoin	2
dtjoin_anti	6
dtjoin_cross	8
dtjoin_semi	10
fjoin_cross	12
fjoin_full	13
fjoin_inner	19
fjoin_left	25
fjoin_left_anti	31
fjoin_left_semi	34
fjoin_right	37
fjoin_right_anti	43
fjoin_right_semi	46

Index	50
--------------	-----------

dtjoin	<i>Join data frame-like objects using an extended DT[i]-style interface to data.table</i>
--------	---

Description

Write (and optionally run) **data.table** code for a join using a generalisation of DT[i] syntax with extended arguments and enhanced behaviour. Accepts any data.frame-like inputs (not only data.tables), permits left, right, inner, and full joins, prevents unwanted matches on NA and NaN by default, does not garble join columns in non-equality joins, allows mult on both sides of the join, creates an optional join indicator column, allows specifying which columns to select from each input, and provides convenience options to control column order and prefixing.

If run, the join returns a data.frame, data.table, tibble, sf, or sf-tibble according to context. The generated **data.table** code can be printed to the console instead of (or as well as) being executed. This feature extends to *mock joins*, where no inputs are provided, and template code is produced.

dtjoin is the workhorse function for fjoin_inner, fjoin_left, fjoin_right, and fjoin_full, which are wrappers providing a more conventional interface for join operations. These functions are recommended over dtjoin for most users and cases.

Usage

```
dtjoin(
  .DT = NULL,
  .i = NULL,
  on,
  match.na = FALSE,
  mult = "all",
  mult.DT = "all",
  nomatch = NA,
```

```

nomatch.DT = NULL,
indicate = FALSE,
select = NULL,
select.DT = NULL,
select.i = NULL,
both = FALSE,
on.first = FALSE,
i.home = FALSE,
i.first = i.home,
prefix = if (i.home) "x." else "i.",
i.class = i.home,
do = !(is.null(.DT) && is.null(.i)),
show = !do,
verbose = FALSE,
...
)

```

Arguments

<code>.DT, .i</code>	data.frame-like objects (plain, <code>data.table</code> , <code>tibble</code> , <code>sf</code> , <code>list</code> , etc.), or else both omitted for a mock join statement with no data.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_DT == col_i", "date < date", "cost <= budget")</code> , or else <code>NA</code> for a natural join (an equality join on all same-named columns).
<code>match.na</code>	If <code>TRUE</code> , allow equality matches between <code>NA</code> s or <code>NaN</code> s. Default <code>FALSE</code> .
<code>mult</code>	(as in <code>[.data.table]</code>) When a row of <code>.i</code> has multiple matching rows in <code>.DT</code> , which to accept. One of <code>"all"</code> (the default), <code>"first"</code> , or <code>"last"</code> .
<code>mult.DT</code>	Like <code>mult</code> , but with the roles of <code>.DT</code> and <code>.i</code> reversed, i.e. when a row of <code>.DT</code> has multiple matching rows in <code>.i</code> , which to accept (default <code>"all"</code>). Can be combined with <code>mult</code> . See Details.
<code>nomatch</code>	(as in <code>[.data.table]</code>) Either <code>NA</code> (the default) to retain rows of <code>.i</code> with no match in <code>.DT</code> , or <code>NULL</code> to exclude them.
<code>nomatch.DT</code>	Like <code>nomatch</code> but with the roles of <code>.DT</code> and <code>.i</code> reversed, and a different default: either <code>NA</code> to append rows of <code>.DT</code> with no match in <code>.i</code> , or <code>NULL</code> (the default) to leave them out.
<code>indicate</code>	Whether to add a column <code>".join"</code> at the front of the result, with values <code>1L</code> if from the "home" table only, <code>2L</code> if from the "foreign" table only, and <code>3L</code> if joined from both tables (c.f. <code>_merge</code> in Stata). Default <code>FALSE</code> .
<code>select, select.DT, select.i</code>	Character vectors of columns to be selected from either input if present (<code>select</code>) or specifically from one or other (<code>select.DT</code> , <code>select.i</code>). <code>NULL</code> (the default) selects all columns. Use <code>"</code> or <code>NA</code> to select no columns. Join columns are always selected. See Details.
<code>both</code>	Whether to include equality join columns from the "foreign" table separately in the output, instead of combining them with those from the "home" table. Default <code>FALSE</code> . Note that non-equality join columns from the foreign table are always included separately.

<code>on.first</code>	Whether to place the join columns from both inputs first in the join result. Default FALSE.
<code>i.home</code>	Whether to treat <code>.i</code> as the "home" table and <code>.DT</code> as the "foreign" table for column prefixing and indicate. Default FALSE, i.e. <code>.DT</code> is the "home" table, as in <code>[.data.table]</code> .
<code>i.first</code>	Whether to place <code>.i</code> 's columns before <code>.DT</code> 's in the join result. The default is to use the value of <code>i.home</code> , i.e. bring <code>.i</code> 's columns to the front if <code>.i</code> is the "home" table.
<code>prefix</code>	A prefix to attach to column names in the "foreign" table that are the same as a column name in the "home" table. The default is <code>"i."</code> if the "foreign" table is <code>.i</code> (<code>i.home</code> is FALSE) and <code>"x."</code> if it is <code>.DT</code> (<code>i.home</code> is TRUE).
<code>i.class</code>	Whether the class of the output should be based on <code>.i</code> instead of <code>.DT</code> . The default follows <code>i.home</code> (default FALSE). See Details for how output class and other attributes are set.
<code>do</code>	Whether to execute the join. Default is TRUE unless <code>.DT</code> and <code>.i</code> are both omitted/NULL, in which case a mock join statement is produced.
<code>show</code>	Whether to print the code for the join to the console. Default is the opposite of <code>do</code> . If <code>.DT</code> and <code>.i</code> are both omitted/NULL, mock join code is displayed.
<code>verbose</code>	(passed to <code>[.data.table]</code>) Whether <code>data.table</code> should print information to the console during execution. Default FALSE.
<code>...</code>	Further arguments (for internal use).

Details

Input and output class: Each input can be any object with class `data.frame`, or a plain list of same-length vectors.

The output class depends on `.DT` by default (but `.i` with `i.class = TRUE`) and is as follows:

- a `data.table` if the input is a pure `data.table`
- a tibble if it is a tibble (and a grouped tibble if it has class `grouped_df`)
- an `sf` if it is an `sf` with its active geometry selected in the join
- a plain `data.frame` in all other cases

The following attributes are carried through and refreshed: `data.table` key, tibble groups, `sf` `agr` (and `bbox` etc. of all individual `sf`-class columns regardless of output class). See below for specifics. Other classes and attributes are not carried through.

Specifying join conditions with `on`: `on` is a required argument. For a natural join (a join by equality on all same-named column pairs), you must specify `on = NA`; you can't just omit `on` as in other packages. This is to prevent a natural join being specified by mistake, which may then go unnoticed.

Using `select`, `select.DT`, and `select.i`: Used on its own, `select` keeps the join columns plus the specified non-join columns from both inputs if present.

If `select.DT` is provided (and similarly for `select.i`) then:

- if `select` is also specified, non-join columns of `.DT` named in either `select` or `select.DT` are included

- if `select` is not specified, only non-join columns named in `select.DT` are included from `.DT`. Thus e.g. `select.DT = ""` excludes all of `.DT`'s non-join columns.

Non-existent column names are ignored without warning.

Column order: When `select` is specified but `select.DT` and `select.i` are not, the output consists of all join columns followed by the selected non-join columns from either input in the order given in `select`.

In all other cases:

- columns from `.DT` come before columns from `.i` by default (but vice versa if `i.first` is `TRUE`)
- within each group of columns, non-join columns are in the order given by `select.DT/select.i`, or in their original data order if no selection is provided
- if `on.first` is `TRUE`, join columns from both inputs are moved to the front of the overall output.

Using `mult` and `mult.DT`: If both of these arguments are not the default "all", `mult` is applied first (typically by passing directly to `[.data.table]`) and `mult.DT` is applied subsequently to eliminate all but the first or last occurrence of each row of `.DT` from the inner part of the join, producing a 1:1 result. This order of operations can affect the identity of the rows in the inner join.

Displaying code and 'mock joins': The option of displaying the join code with `show = TRUE` or by passing null inputs is aimed at `data.table` users wanting to use the package as a cookbook of recipes for adaptation. If `.DT` and `.i` are both `NULL`, template code is displayed based on join column names implied by `on`, plus sample non-join column names. `select` arguments are ignored in this case.

The code displayed is for the join operation after casting the inputs as `data.tables` if necessary, and before casting the result as a `tibble` and/or `sf` if applicable. Note that **fjoin** departs from the usual `j = list()` idiom in order to avoid a deep copy of the output made by `as.data.table.list`. (Likewise, internally it takes only shallow copies of columns when casting inputs or outputs to different classes.)

tibble groups: If the relevant input is a grouped `tibble` (class `grouped_df`), the output is grouped by the grouping columns that are selected in the result.

data.table keys: If `.i` is a keyed `data.table` and the output is also a `data.table`, it inherits `.i`'s key provided `nomatch.DT` is `NULL` (i.e. the non-matching rows of `.DT` are not included in the result). This differs from a `data.table DT[i]` join, in which the output inherits the key of `DT` provided it remains sorted on those columns. If not all of the key columns are selected in the result, the leading subset is used.

sf objects and sfc-class columns: Joins between two `sf` objects are supported. The relation-to-geometry attribute `agr` is inherited from the input supplying the active geometry. All `sfc-class` columns in the output are refreshed after joining (using `sf::st_sfc()` with `recompute_bbox = TRUE`); this is true regardless of whether or not the inputs and output are `sfs`.

Value

A `data.frame`, `data.table`, (grouped) `tibble`, `sf`, or `sf-tibble`, or else `NULL` if `do` is `FALSE`. See `Details`.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# An illustration showing:
# - two calls to fjoin_left() (commented out), differing in the `order` argument
# - the resulting calls to dtjoin(), plus `show = TRUE`
# - the generated data.table code and output

# data frames
set.seed(1)
df_x <- data.frame(id_x = 1:3, col_x = paste0("x", 1:3), val = runif(3))
df_y <- data.frame(id_y = rep(4:2, each = 2), col_y = paste0("y", 1:6), val = runif(6))

# -----

# (1) fjoin_left(df_x, df_y, on = "id_x == id_y", mult.x = "first")
dtjoin(
  df_y,
  df_x,
  on = "id_y == id_x",
  mult = "first",
  i.home = TRUE,
  prefix = "R.",
  show = TRUE
)

# (2) fjoin_left(df_x, df_y, on = "id_x == id_y", mult.x = "first", order = "right")
dtjoin(
  df_x,
  df_y,
  on = "id_x == id_y",
  mult.DT = "first",
  nomatch = NULL,
  nomatch.DT = NA,
  prefix = "R.",
  show = TRUE
)
```

dtjoin_anti

Anti-join of DT in a DT[i]-style join of data frame-like objects

Description

Write (and optionally run) **data.table** code to return the anti-join of DT (the rows of DT not joining with i) using a generalisation of DT[i] syntax.

The functions [fjoin_left_anti](#) and [fjoin_right_anti](#) provide a more conventional interface that is recommended over dtjoin_anti for most users and cases.

Usage

```
dtjoin_anti(
  .DT = NULL,
  .i = NULL,
  on,
  match.na = FALSE,
  mult = "all",
  mult.DT = "all",
  nomatch = NULL,
  nomatch.DT = NULL,
  select = NULL,
  do = !(is.null(.DT) && is.null(.i)),
  show = !do,
  verbose = FALSE,
  ...
)
```

Arguments

<code>.DT</code> , <code>.i</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.), or else both omitted for a mock join statement with no data.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_DT == col_i", "date < date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	If TRUE, allow equality matches between NAs or NaNs. Default FALSE.
<code>mult</code>	(as in [.data.table]) When a row of .i has multiple matching rows in .DT, which to accept. One of "all" (the default), "first", or "last".
<code>mult.DT</code>	Permitted for consistency with dtjoin but has no effect on the resulting anti-join.
<code>nomatch</code> , <code>nomatch.DT</code>	Permitted for consistency with dtjoin but have no effect on the resulting anti-join.
<code>select</code>	Character vector of columns of .DT to be selected. NULL (the default) selects all columns. Join columns are always selected.
<code>do</code>	Whether to execute the join. Default is TRUE unless .DT and .i are both omitted/NULL, in which case a mock join statement is produced.
<code>show</code>	Whether to print the code for the join to the console. Default is the opposite of do. If .DT and .i are both omitted/NULL, mock join code is displayed.
<code>verbose</code>	(passed to [.data.table]) Whether data.table should print information to the console during execution. Default FALSE.
<code>...</code>	Further arguments (for internal use).

Details

Details are as for `dtjoin` except for arguments controlling the order and prefixing of output columns, which do not apply.

Value

A `data.frame`, `data.table`, (grouped) tibble, `sf`, or `sf-tibble`, or else `NULL` if `do` is `FALSE`. See [Details](#).

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# Mock joins

dtjoin_anti(on = "id")
dtjoin_anti(on = c("id", "date <= date"))
dtjoin_anti(on = c("id", "date <= date"), mult = "last")
```

<code>dtjoin_cross</code>	<i>Cross join of data frame-like objects <code>DT</code> and <code>i</code> using a <code>DT[i]</code>-style interface to <code>data.table</code></i>
---------------------------	---

Description

Write (and optionally run) `data.table` code to return the cross join of two `data.frame`-like objects using a generalisation of `DT[i]` syntax.

The function [fjoin_cross](#) provides a more conventional interface that is recommended over `dtjoin_cross` for most users and cases.

Usage

```
dtjoin_cross(
  .DT = NULL,
  .i = NULL,
  select = NULL,
  select.DT = NULL,
  select.i = NULL,
  i.home = FALSE,
  i.first = i.home,
  prefix = if (i.home) "x." else "i.",
  i.class = i.home,
  do = !(is.null(.DT) && is.null(.i)),
  show = !do,
  ...
)
```

Arguments

<code>.DT, .i</code>	data.frame-like objects (plain, <code>data.table</code> , <code>tibble</code> , <code>sf</code> , <code>list</code> , etc.), or else both omitted for a mock join statement with no data.
<code>select, select.DT, select.i</code>	Character vectors of columns to be selected from either input if present (<code>select</code>) or specifically from one or other (<code>select.DT</code> , <code>select.i</code>). <code>NULL</code> (the default) selects all columns. Use <code>""</code> or <code>NA</code> to select no columns. Join columns are always selected. See Details.
<code>i.home</code>	Whether to treat <code>.i</code> as the "home" table and <code>.DT</code> as the "foreign" table for column prefixing and indicate. Default <code>FALSE</code> , i.e. <code>.DT</code> is the "home" table, as in <code>[.data.table]</code> .
<code>i.first</code>	Whether to place <code>.i</code> 's columns before <code>.DT</code> 's in the join result. The default is to use the value of <code>i.home</code> , i.e. bring <code>.i</code> 's columns to the front if <code>.i</code> is the "home" table.
<code>prefix</code>	A prefix to attach to column names in the "foreign" table that are the same as a column name in the "home" table. The default is <code>"i."</code> if the "foreign" table is <code>.i</code> (<code>i.home</code> is <code>FALSE</code>) and <code>"x."</code> if it is <code>.DT</code> (<code>i.home</code> is <code>TRUE</code>).
<code>i.class</code>	Whether the class of the output should be based on <code>.i</code> instead of <code>.DT</code> . The default follows <code>i.home</code> (default <code>FALSE</code>). See Details for how output class and other attributes are set.
<code>do</code>	Whether to execute the join. Default is <code>TRUE</code> unless <code>.DT</code> and <code>.i</code> are both omitted/ <code>NULL</code> , in which case a mock join statement is produced.
<code>show</code>	Whether to print the code for the join to the console. Default is the opposite of <code>do</code> . If <code>.DT</code> and <code>.i</code> are both omitted/ <code>NULL</code> , mock join code is displayed.
<code>...</code>	Further arguments (for internal use).

Details

Details are as for `dtjoin` except for remarks about join columns and matching logic, which do not apply.

Value

A `data.frame`, `data.table`, (grouped) `tibble`, `sf`, or `sf-tibble`, or else `NULL` if `do` is `FALSE`. See Details.

See Also

See the package-level documentation `fjoin` for related functions.

Examples

```
# data frames
df1 <- data.table::fread(data.table = FALSE, input = "
bread    kcal
Brown    150
White    180
```

```

Baguette 250
")

df2 <- data.table::fread(data.table = FALSE, input = "
filling kcal
Cheese 200
Pâté 160
")

dtjoin_cross(df1, df2)

```

dtjoin_semi

Semi-join of DT in a DT[i]-style join of data frame-like objects

Description

Write (and optionally run) **data.table** code to return the semi-join of DT (the rows of DT that join with i) using a generalisation of DT[i] syntax.

The functions `fjoin_left_semi` and `fjoin_right_semi` provide a more conventional interface that is recommended over `dtjoin_semi` for most users and cases.

Usage

```

dtjoin_semi(
  .DT = NULL,
  .i = NULL,
  on,
  match.na = FALSE,
  mult = "all",
  mult.DT = "all",
  nomatch = NULL,
  nomatch.DT = NULL,
  select = NULL,
  do = !(is.null(.DT) && is.null(.i)),
  show = !do,
  verbose = FALSE,
  ...
)

```

Arguments

<code>.DT, .i</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.), or else both omitted for a mock join statement with no data.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_DT == col_i", "date < date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).

<code>match.na</code>	If TRUE, allow equality matches between NAs or NaNs. Default FALSE.
<code>mult</code>	(as in <code>[.data.table]</code>) When a row of <code>.i</code> has multiple matching rows in <code>.DT</code> , which to accept. One of "all" (the default), "first", or "last".
<code>mult.DT</code>	Permitted for consistency with <code>dtjoin</code> but has no effect on the resulting semi-join.
<code>nomatch, nomatch.DT</code>	Permitted for consistency with <code>dtjoin</code> but have no effect on the resulting semi-join.
<code>select</code>	Character vector of columns of <code>.DT</code> to be selected. NULL (the default) selects all columns. Join columns are always selected.
<code>do</code>	Whether to execute the join. Default is TRUE unless <code>.DT</code> and <code>.i</code> are both omitted/NULL, in which case a mock join statement is produced.
<code>show</code>	Whether to print the code for the join to the console. Default is the opposite of <code>do</code> . If <code>.DT</code> and <code>.i</code> are both omitted/NULL, mock join code is displayed.
<code>verbose</code>	(passed to <code>[.data.table]</code>) Whether <code>data.table</code> should print information to the console during execution. Default FALSE.
<code>...</code>	Further arguments (for internal use).

Details

Details are as for `dtjoin` except for arguments controlling the order and prefixing of output columns, which do not apply.

Value

A `data.frame`, `data.table`, (grouped) tibble, `sf`, or `sf-tibble`, or else NULL if `do` is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# Mock joins

dtjoin_semi(on = "id")
dtjoin_semi(on = c("id", "date <= date"))
dtjoin_semi(on = c("id", "date <= date"), mult = "last")
```

fjoin_cross

*Cross join***Description**

Cross join of x and y

Usage

```
fjoin_cross(
  x = NULL,
  y = NULL,
  order = "left",
  select = NULL,
  select.x = NULL,
  select.y = NULL,
  prefix.y = "R.",
  do = !(is.null(x) && is.null(y)),
  show = !do
)
```

Arguments

x, y	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
order	Whether the row order of the result should reflect x then y ("left") or y then x ("right"). Default "left".
select, select.x, select.y	Character vectors of columns to be selected from either input if present (select) or specifically from one or other of them (e.g. select.x). NULL (the default) selects all columns. Use "" or NA to select no columns. Join columns are always selected. See Details.
prefix.y	A prefix to attach to column names in y that are the same as a column name in x. Default "R. ".
do	Whether to execute the join. If FALSE, show is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless x and y are both omitted/NULL, in which case a mock join statement is produced. See Details.
show	Whether to print the data.table code for the join to the console. Default is the opposite of do. If x and y are both omitted/NULL, mock join code is displayed.

Details

Details are as for e.g. [fjoin_inner](#) except for remarks about join columns and matching logic, which do not apply.

Value

A data.frame, data.table, (grouped) tibble, sf, or sf-tibble, or else NULL if do is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# data frames
df1 <- data.table::fread(data.table = FALSE, input = "
bread    kcal
Brown    150
White    180
Baguette 250
")

df2 <- data.table::fread(data.table = FALSE, input = "
filling  kcal
Cheese   200
Pâté     160
")

fjoin_cross(df1, df2)
fjoin_cross(df1, df2, order = "right")
```

fjoin_full

Full join

Description

Full join of x and y

Usage

```
fjoin_full(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  on.first = FALSE,
  order = "left",
  select = NULL,
  select.x = NULL,
```

```

select.y = NULL,
indicate = FALSE,
prefix.y = "R.",
both = FALSE,
do = !(is.null(x) && is.null(y)),
show = !do
)

```

Arguments

<code>x, y</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	Whether to allow equality matches between NAs or NaNs. Default FALSE.
<code>mult.x, mult.y</code>	When a row of <code>x</code> (<code>y</code>) has multiple matching rows in <code>y</code> (<code>x</code>), which to accept: "all" (the default), "first", or "last". May be used in combination.
<code>on.first</code>	Whether to place the join columns first in the join result. Default FALSE.
<code>order</code>	Whether the row order of the result should reflect <code>x</code> then <code>y</code> ("left") or <code>y</code> then <code>x</code> ("right"). Default "left".
<code>select, select.x, select.y</code>	Character vectors of columns to be selected from either input if present (<code>select</code>) or specifically from one or other of them (e.g. <code>select.x</code>). NULL (the default) selects all columns. Use "" or NA to select no columns. Join columns are always selected. See Details.
<code>indicate</code>	Whether to add a column ".join" at the front of the result, with values 1L if from <code>x</code> only, 2L if from <code>y</code> only, and 3L if joined from both tables (c.f. <code>_merge</code> in Stata). Default FALSE.
<code>prefix.y</code>	A prefix to attach to column names in <code>y</code> that are the same as a column name in <code>x</code> . Default "R. ".
<code>both</code>	Whether to include <code>y</code> 's equality join column(s) separately in the output, instead of combining them with <code>x</code> 's. Default FALSE. Note that non-equality join columns from <code>x</code> are always included separately.
<code>do</code>	Whether to execute the join. If FALSE, <code>show</code> is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless <code>x</code> and <code>y</code> are both omitted/NULL, in which case a mock join statement is produced. See Details.
<code>show</code>	Whether to print the data.table code for the join to the console. Default is the opposite of <code>do</code> . If <code>x</code> and <code>y</code> are both omitted/NULL, mock join code is displayed.

Details

Input and output class: Each input can be any object with class `data.frame`, or a plain list of same-length vectors.

The output class depends on `x` as follows:

- a `data.table` if `x` is a pure `data.table`
- a tibble if it is a tibble (and a grouped tibble if it has class `grouped_df`)
- an `sf` if it is an `sf` with its active geometry selected in the output
- a plain `data.frame` in all other cases

The following attributes are carried through and refreshed: `data.table` key, tibble groups, `sf` `agr` (and `bbox` etc. of all individual `sfc`-class columns regardless of output class). See below for specifics.

Specifying join conditions with `on`: `on` is a required argument. For a natural join (a join by equality on all same-named column pairs), you must specify `on = NA`; you can't just omit `on` as in other packages. This is to prevent a natural join being specified by mistake, which may then go unnoticed.

Using `select`, `select.x`, and `select.y`: Used on its own, `select` keeps the join columns plus the specified non-join columns from both inputs if present.

If `select.x` is provided (and similarly for `select.y`) then:

- if `select` is also specified, non-join columns of `x` named in either `select` or `select.x` are included
- if `select` is not specified, only non-join columns named in `select.x` are included from `x`. Thus e.g. `select.x = ""` excludes all of `x`'s non-join columns.

Non-existent column names are ignored without warning.

Column order: When `select` is specified but `select.x` and `select.y` are not, the output consists of all join columns followed by the selected non-join columns from either input in the order given in `select`.

In all other cases:

- columns from `x` come before columns from `y`
- within each group of columns, non-join columns are in the order given by `select.x/select.y`, or in their original data order if no selection is provided
- if `on.first` is `TRUE`, join columns from both inputs are moved to the front of the overall output.

Using `mult.x` and `mult.y`: See the Examples for an application of using `mult.x` and `mult.y` together. Note that `mult.y` is applied after `mult.x` except with `order = "right"`.

Displaying code and 'mock joins': The option of displaying the join code with `show = TRUE` or by passing null inputs is aimed at **data.table** users wanting to use the package as a cookbook of recipes for adaptation. If `x` and `y` are both `NULL`, template code is displayed based on join column names implied by `on`, plus sample non-join column names. `select` arguments are ignored in this case.

The code displayed is for the join operation after casting the inputs as `data.tables` if necessary, and before casting the result as a tibble and/or `sf` if applicable. Note that **fjoin** departs from the usual `j = list()` idiom in order to avoid a deep copy of the output made by `as.data.table.list`. (Likewise, internally it takes only shallow copies of columns when casting inputs or outputs to different classes.)

tibble groups: If `x` is a grouped tibble (class `grouped_df`), the output is grouped by the grouping columns that are selected in the result.

data.table keys: If the output is a `data.table`, it inherits a key as follows:

- `fjoin_inner` or `fjoin_left` with `order = "left"` (default): `x`'s key if present
- `fjoin_inner` or `fjoin_right` with `order = "right"`: `y`'s key if present

If not all of the key columns are selected in the result, the leading subset is used.

sf objects and sfc-class columns: Joins between two `sf` objects are supported. The active geometry and relation-to-geometry attribute `agr` are determined by `x`. All `sfc-class` columns in the output are refreshed after joining (using `sf::st_sfc()` with `recompute_bbox = TRUE`); this is true regardless of whether or not the inputs and output are `sfs`.

Value

A `data.frame`, `data.table`, (grouped) `tibble`, `sf`, or `sf-tibble`, or else `NULL` if `do` is `FALSE`. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# True joins (inner/left/right/full): basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
  country pop_m
Australia 27.2
  Brazil 212.0
  Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
  country forest_pc
  Brazil 59.1
  Chad 3.2
  Denmark 15.8
")

# -----
# `indicate = TRUE` adds a front column ".join" indicating whether a row is
# from `x` only (1L), from `y` only (2L), or joined from both (3L)

fjoin_full(x, y, on = "country", indicate = TRUE)
fjoin_left(x, y, on = "country", indicate = TRUE)
fjoin_right(x, y, on = "country", indicate = TRUE)
fjoin_inner(x, y, on = "country", indicate = TRUE)

# -----
# Core options and arguments (in a 1:1 equality join with fjoin_full())
```

```

# -----

# data frames
dfQ <- data.table::fread(data.table = FALSE, quote = "'", input = "
id quantity          notes other_cols
  2      5             ''          ...
  1      6             ''          ...
  3      7             ''          ...
NA      8 'oranges (not listed)'    ...
")

dfP <- data.table::fread(data.table = FALSE, input = "
id  item price other_cols
NA  apples  10     ...
  3 bananas  20     ...
  2 cherries 30     ...
  1  dates  40     ...
")

# -----

# (1) basic syntax
# cf. dplyr: full_join(dfQ, dfP, join_by(id), na.matches = "never")
fjoin_full(dfQ, dfP, on = "id")

# (an aside) equality matches on NA if you insist
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity", "notes"), match.na = TRUE)

# (2) join-select in one line
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity"))

# equivalent operation in dplyr
# x <- dfQ |> select(id, quantity)
# y <- dfP |> select(id, item, price)
# full_join(x, y, join_by(id), na.matches = "never") |>
#   select(id, item, price, quantity)

# (3) indicator column (in Stata since 1984)
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE
)

# (4) order rows by y then x
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,

```

```

    order = "right"
  )

# (5) display code instead
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,
  order = "right",
  do = FALSE
)

# -----
# M:M inequality join reduced to 1:1 using `mult.x` and `mult.y`
# -----

# data.table (`mult`) and dplyr (`multiple`) have options for reducing the
# cardinality on one side of the join from many ("all") to one ("first" or
# "last"). fjoin (`mult.x`, `mult.y`) permits this on either side of the
# join, or on both sides at once.

# This example (using `fjoin_left()`) shows an application to temporally
# ordered data frames of "events" and "reactions".

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
    1      10
    2      20
    3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
    1         30
    2         50
    3         60
")

# -----

# (1) for each event, all subsequent reactions (M:M)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
)

# (2) for each event, the next reaction (1:M)
fjoin_left(
  events,
  reactions,

```

```

    on = c("event_ts < reaction_ts"),
    mult.x = "first"
  )

# (3) for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

# -----
# Natural join
# -----
fjoin_inner(x, y, on = NA) # note `NA` not `NULL`/omitted
try(fjoin_inner(x, y)) # to prevent accidental natural joins

# -----
# Mock join (code "ghostwriter" for data.table users)
# -----
fjoin_inner(on = c("id"))

```

fjoin_inner

Inner join

Description

Inner join of x and y

Usage

```

fjoin_inner(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  order = "left",
  select = NULL,
  select.x = NULL,
  select.y = NULL,
  indicate = FALSE,
  prefix.y = "R.",
  on.first = FALSE,
  both = FALSE,

```

```

do = !(is.null(x) && is.null(y)),
show = !do
)

```

Arguments

<code>x, y</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	Whether to allow equality matches between NAs or NaNs. Default FALSE.
<code>mult.x, mult.y</code>	When a row of x (y) has multiple matching rows in y (x), which to accept: "all" (the default), "first", or "last". May be used in combination.
<code>order</code>	Whether the row order of the result should reflect x then y ("left") or y then x ("right"). Default "left".
<code>select, select.x, select.y</code>	Character vectors of columns to be selected from either input if present (select) or specifically from one or other of them (e.g. select.x). NULL (the default) selects all columns. Use "" or NA to select no columns. Join columns are always selected. See Details.
<code>indicate</code>	Whether to add a column ".join" at the front of the result, with values 1L if from x only, 2L if from y only, and 3L if joined from both tables (c.f. _merge in Stata). Default FALSE.
<code>prefix.y</code>	A prefix to attach to column names in y that are the same as a column name in x. Default "R. ".
<code>on.first</code>	Whether to place the join columns first in the join result. Default FALSE.
<code>both</code>	Whether to include y's equality join column(s) separately in the output, instead of combining them with x's. Default FALSE. Note that non-equality join columns from x are always included separately.
<code>do</code>	Whether to execute the join. If FALSE, show is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless x and y are both omitted/NULL, in which case a mock join statement is produced. See Details.
<code>show</code>	Whether to print the data.table code for the join to the console. Default is the opposite of do. If x and y are both omitted/NULL, mock join code is displayed.

Details

Input and output class: Each input can be any object with class `data.frame`, or a plain list of same-length vectors.

The output class depends on x as follows:

- a `data.table` if x is a pure `data.table`
- a `tibble` if it is a `tibble` (and a grouped `tibble` if it has class `grouped_df`)
- an `sf` if it is an `sf` with its active geometry selected in the output

- a plain `data.frame` in all other cases

The following attributes are carried through and refreshed: `data.table` key, tibble groups, `sf` agr (and `bbox` etc. of all individual `sfc`-class columns regardless of output class). See below for specifics.

Specifying join conditions with `on`: `on` is a required argument. For a natural join (a join by equality on all same-named column pairs), you must specify `on = NA`; you can't just omit `on` as in other packages. This is to prevent a natural join being specified by mistake, which may then go unnoticed.

Using `select`, `select.x`, and `select.y`: Used on its own, `select` keeps the join columns plus the specified non-join columns from both inputs if present.

If `select.x` is provided (and similarly for `select.y`) then:

- if `select` is also specified, non-join columns of `x` named in either `select` or `select.x` are included
- if `select` is not specified, only non-join columns named in `select.x` are included from `x`. Thus e.g. `select.x = ""` excludes all of `x`'s non-join columns.

Non-existent column names are ignored without warning.

Column order: When `select` is specified but `select.x` and `select.y` are not, the output consists of all join columns followed by the selected non-join columns from either input in the order given in `select`.

In all other cases:

- columns from `x` come before columns from `y`
- within each group of columns, non-join columns are in the order given by `select.x/select.y`, or in their original data order if no selection is provided
- if `on.first` is `TRUE`, join columns from both inputs are moved to the front of the overall output.

Using `mult.x` and `mult.y`: See the Examples for an application of using `mult.x` and `mult.y` together. Note that `mult.y` is applied after `mult.x` except with `order = "right"`.

Displaying code and 'mock joins': The option of displaying the join code with `show = TRUE` or by passing null inputs is aimed at `data.table` users wanting to use the package as a cookbook of recipes for adaptation. If `x` and `y` are both `NULL`, template code is displayed based on join column names implied by `on`, plus sample non-join column names. `select` arguments are ignored in this case.

The code displayed is for the join operation after casting the inputs as `data.tables` if necessary, and before casting the result as a tibble and/or `sf` if applicable. Note that `fjoin` departs from the usual `j = list()` idiom in order to avoid a deep copy of the output made by `as.data.table.list`. (Likewise, internally it takes only shallow copies of columns when casting inputs or outputs to different classes.)

tibble groups: If `x` is a grouped tibble (class `grouped_df`), the output is grouped by the grouping columns that are selected in the result.

data.table keys: If the output is a `data.table`, it inherits a key as follows:

- `fjoin_inner` or `fjoin_left` with `order = "left"` (default): `x`'s key if present
- `fjoin_inner` or `fjoin_right` with `order = "right"`: `y`'s key if present

If not all of the key columns are selected in the result, the leading subset is used.

sf objects and sfc-class columns: Joins between two `sf` objects are supported. The active geometry and relation-to-geometry attribute `agr` are determined by `x`. All `sfc`-class columns in the output are refreshed after joining (using `sf::st_sfc()` with `recompute_bbox = TRUE`); this is true regardless of whether or not the inputs and output are `sfs`.

Value

A `data.frame`, `data.table`, (grouped) `tibble`, `sf`, or `sf-tibble`, or else `NULL` if `do` is `FALSE`. See `Details`.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# True joins (inner/left/right/full): basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
  country pop_m
Australia 27.2
  Brazil 212.0
  Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
  country forest_pc
  Brazil 59.1
  Chad 3.2
  Denmark 15.8
")

# -----
# `indicate = TRUE` adds a front column ".join" indicating whether a row is
# from `x` only (1L), from `y` only (2L), or joined from both (3L)

fjoin_full(x, y, on = "country", indicate = TRUE)
fjoin_left(x, y, on = "country", indicate = TRUE)
fjoin_right(x, y, on = "country", indicate = TRUE)
fjoin_inner(x, y, on = "country", indicate = TRUE)

# -----
# Core options and arguments (in a 1:1 equality join with fjoin_full())
# -----
```

```

# data frames
dfQ <- data.table::fread(data.table = FALSE, quote = "'", input = "
id quantity          notes other_cols
  2      5            ''          ...
  1      6            ''          ...
  3      7            ''          ...
NA      8 'oranges (not listed)'    ...
")

dfP <- data.table::fread(data.table = FALSE, input = "
id  item price other_cols
NA  apples  10    ...
  3 bananas  20    ...
  2 cherries 30    ...
  1  dates  40    ...
")

# -----

# (1) basic syntax
# cf. dplyr: full_join(dfQ, dfP, join_by(id), na.matches = "never")
fjoin_full(dfQ, dfP, on = "id")

# (an aside) equality matches on NA if you insist
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity", "notes"), match.na = TRUE)

# (2) join-select in one line
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity"))

# equivalent operation in dplyr
# x <- dfQ |> select(id, quantity)
# y <- dfP |> select(id, item, price)
# full_join(x, y, join_by(id), na.matches = "never") |>
#   select(id, item, price, quantity)

# (3) indicator column (in Stata since 1984)
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE
)

# (4) order rows by y then x
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,
  order = "right"
)

```

```

# (5) display code instead
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,
  order = "right",
  do = FALSE
)

# -----
# M:M inequality join reduced to 1:1 using `mult.x` and `mult.y`
# -----

# data.table (`mult`) and dplyr (`multiple`) have options for reducing the
# cardinality on one side of the join from many ("all") to one ("first" or
# "last"). fjoin (`mult.x`, `mult.y`) permits this on either side of the
# join, or on both sides at once.

# This example (using `fjoin_left()`) shows an application to temporally
# ordered data frames of "events" and "reactions".

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
  1      10
  2      20
  3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
  1          30
  2          50
  3          60
")

# -----

# (1) for each event, all subsequent reactions (M:M)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
)

# (2) for each event, the next reaction (1:M)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first"
)

```

```

)

# (3) for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

# -----
# Natural join
# -----
fjoin_inner(x, y, on = NA) # note `NA` not `NULL`/omitted
try(fjoin_inner(x, y)) # to prevent accidental natural joins

# -----
# Mock join (code "ghostwriter" for data.table users)
# -----
fjoin_inner(on = c("id"))

```

fjoin_left

Left join

Description

Left join of x and y

Usage

```

fjoin_left(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  order = "left",
  select = NULL,
  select.x = NULL,
  select.y = NULL,
  indicate = FALSE,
  prefix.y = "R.",
  on.first = FALSE,
  both = FALSE,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

```

Arguments

<code>x, y</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	Whether to allow equality matches between NAs or NaNs. Default FALSE.
<code>mult.x, mult.y</code>	When a row of <code>x</code> (<code>y</code>) has multiple matching rows in <code>y</code> (<code>x</code>), which to accept: "all" (the default), "first", or "last". May be used in combination.
<code>order</code>	Whether the row order of the result should reflect <code>x</code> then <code>y</code> ("left") or <code>y</code> then <code>x</code> ("right"). Default "left".
<code>select, select.x, select.y</code>	Character vectors of columns to be selected from either input if present (<code>select</code>) or specifically from one or other of them (e.g. <code>select.x</code>). NULL (the default) selects all columns. Use "" or NA to select no columns. Join columns are always selected. See Details.
<code>indicate</code>	Whether to add a column ".join" at the front of the result, with values 1L if from <code>x</code> only, 2L if from <code>y</code> only, and 3L if joined from both tables (c.f. <code>_merge</code> in Stata). Default FALSE.
<code>prefix.y</code>	A prefix to attach to column names in <code>y</code> that are the same as a column name in <code>x</code> . Default "R. ".
<code>on.first</code>	Whether to place the join columns first in the join result. Default FALSE.
<code>both</code>	Whether to include <code>y</code> 's equality join column(s) separately in the output, instead of combining them with <code>x</code> 's. Default FALSE. Note that non-equality join columns from <code>x</code> are always included separately.
<code>do</code>	Whether to execute the join. If FALSE, <code>show</code> is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless <code>x</code> and <code>y</code> are both omitted/NULL, in which case a mock join statement is produced. See Details.
<code>show</code>	Whether to print the data.table code for the join to the console. Default is the opposite of <code>do</code> . If <code>x</code> and <code>y</code> are both omitted/NULL, mock join code is displayed.

Details

Input and output class: Each input can be any object with class `data.frame`, or a plain list of same-length vectors.

The output class depends on `x` as follows:

- a `data.table` if `x` is a pure `data.table`
- a `tibble` if it is a `tibble` (and a grouped `tibble` if it has class `grouped_df`)
- an `sf` if it is an `sf` with its active geometry selected in the output
- a plain `data.frame` in all other cases

The following attributes are carried through and refreshed: `data.table` key, `tibble` groups, `sf` agr (and `bbox` etc. of all individual `sfc`-class columns regardless of output class). See below for specifics.

Specifying join conditions with on: `on` is a required argument. For a natural join (a join by equality on all same-named column pairs), you must specify `on = NA`; you can't just omit `on` as in other packages. This is to prevent a natural join being specified by mistake, which may then go unnoticed.

Using `select`, `select.x`, and `select.y`: Used on its own, `select` keeps the join columns plus the specified non-join columns from both inputs if present.

If `select.x` is provided (and similarly for `select.y`) then:

- if `select` is also specified, non-join columns of `x` named in either `select` or `select.x` are included
- if `select` is not specified, only non-join columns named in `select.x` are included from `x`. Thus e.g. `select.x = ""` excludes all of `x`'s non-join columns.

Non-existent column names are ignored without warning.

Column order: When `select` is specified but `select.x` and `select.y` are not, the output consists of all join columns followed by the selected non-join columns from either input in the order given in `select`.

In all other cases:

- columns from `x` come before columns from `y`
- within each group of columns, non-join columns are in the order given by `select.x/select.y`, or in their original data order if no selection is provided
- if `on.first` is `TRUE`, join columns from both inputs are moved to the front of the overall output.

Using `mult.x` and `mult.y`: See the Examples for an application of using `mult.x` and `mult.y` together. Note that `mult.y` is applied after `mult.x` except with `order = "right"`.

Displaying code and 'mock joins': The option of displaying the join code with `show = TRUE` or by passing null inputs is aimed at `data.table` users wanting to use the package as a cookbook of recipes for adaptation. If `x` and `y` are both `NULL`, template code is displayed based on join column names implied by `on`, plus sample non-join column names. `select` arguments are ignored in this case.

The code displayed is for the join operation after casting the inputs as `data.tables` if necessary, and before casting the result as a `tibble` and/or `sf` if applicable. Note that `fjoin` departs from the usual `j = list()` idiom in order to avoid a deep copy of the output made by `as.data.table.list`. (Likewise, internally it takes only shallow copies of columns when casting inputs or outputs to different classes.)

tibble groups: If `x` is a grouped `tibble` (class `grouped_df`), the output is grouped by the grouping columns that are selected in the result.

data.table keys: If the output is a `data.table`, it inherits a key as follows:

- `fjoin_inner` or `fjoin_left` with `order = "left"` (default): `x`'s key if present
- `fjoin_inner` or `fjoin_right` with `order = "right"`: `y`'s key if present

If not all of the key columns are selected in the result, the leading subset is used.

sf objects and sfc-class columns: Joins between two `sf` objects are supported. The active geometry and relation-to-geometry attribute `agr` are determined by `x`. All `sfc-class` columns in the output are refreshed after joining (using `sf::st_sfc()` with `recompute_bbox = TRUE`); this is true regardless of whether or not the inputs and output are `sfs`.

Value

A data.frame, data.table, (grouped) tibble, sf, or sf-tibble, or else NULL if do is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# True joins (inner/left/right/full): basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
  country pop_m
Australia 27.2
  Brazil 212.0
  Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
  country forest_pc
  Brazil 59.1
  Chad 3.2
  Denmark 15.8
")

# -----
# `indicate = TRUE` adds a front column ".join" indicating whether a row is
# from `x` only (1L), from `y` only (2L), or joined from both (3L)

fjoin_full(x, y, on = "country", indicate = TRUE)
fjoin_left(x, y, on = "country", indicate = TRUE)
fjoin_right(x, y, on = "country", indicate = TRUE)
fjoin_inner(x, y, on = "country", indicate = TRUE)

# -----
# Core options and arguments (in a 1:1 equality join with fjoin_full())
# -----

# data frames
dfQ <- data.table::fread(data.table = FALSE, quote = "'", input = "
id quantity          notes other_cols
  2      5              ''           ...
  1      6              ''           ...
  3      7              ''           ...
NA      8 'oranges (not listed)'      ...
")

dfP <- data.table::fread(data.table = FALSE, input = "
```

```

id    item price other_cols
NA   apples  10      ...
 3   bananas  20      ...
 2   cherries 30      ...
 1    dates  40      ...
")

# -----

# (1) basic syntax
# cf. dplyr: full_join(dfQ, dfP, join_by(id), na.matches = "never")
fjoin_full(dfQ, dfP, on = "id")

# (an aside) equality matches on NA if you insist
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity", "notes"), match.na = TRUE)

# (2) join-select in one line
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity"))

# equivalent operation in dplyr
# x <- dfQ |> select(id, quantity)
# y <- dfP |> select(id, item, price)
# full_join(x, y, join_by(id), na.matches = "never") |>
#   select(id, item, price, quantity)

# (3) indicator column (in Stata since 1984)
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE
)

# (4) order rows by y then x
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,
  order = "right"
)

# (5) display code instead
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,
  order = "right",
  do = FALSE
)

```

```

)

# -----
# M:M inequality join reduced to 1:1 using `mult.x` and `mult.y`
# -----

# data.table (`mult`) and dplyr (`multiple`) have options for reducing the
# cardinality on one side of the join from many ("all") to one ("first" or
# "last"). fjoin (`mult.x`, `mult.y`) permits this on either side of the
# join, or on both sides at once.

# This example (using `fjoin_left()`) shows an application to temporally
# ordered data frames of "events" and "reactions".

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
      1      10
      2      20
      3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
          1          30
          2          50
          3          60
")

# -----

# (1) for each event, all subsequent reactions (M:M)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
)

# (2) for each event, the next reaction (1:M)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first"
)

# (3) for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

```

```

# -----
# Natural join
# -----
fjoin_inner(x, y, on = NA) # note `NA` not `NULL`/omitted
try(fjoin_inner(x, y)) # to prevent accidental natural joins

# -----
# Mock join (code "ghostwriter" for data.table users)
# -----
fjoin_inner(on = c("id"))

```

fjoin_left_anti	<i>Left anti-join</i>
-----------------	-----------------------

Description

The anti-join of x in a join of x and y, i.e. the rows of x that do not join. The alias `fjoin_anti` can be used instead.

Usage

```

fjoin_left_anti(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  select = NULL,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

fjoin_anti(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  select = NULL,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

```

Arguments

<code>x, y</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	Whether to allow equality matches between NAs or NaNs. Default FALSE.
<code>mult.x, mult.y</code>	When a row of <code>x</code> (<code>y</code>) has multiple matching rows in <code>y</code> (<code>x</code>), which to accept: "all" (the default), "first", or "last". May be used in combination.
<code>select</code>	Character vector of non-join columns to be selected from <code>x</code> . NULL (the default) selects all columns. Join columns are always selected.
<code>do</code>	Whether to execute the join. If FALSE, <code>show</code> is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless <code>x</code> and <code>y</code> are both omitted/NULL, in which case a mock join statement is produced. See Details.
<code>show</code>	Whether to print the data.table code for the join to the console. Default is the opposite of <code>do</code> . If <code>x</code> and <code>y</code> are both omitted/NULL, mock join code is displayed.

Details

Details are as for [fjoin_inner](#) except for arguments controlling the order and prefixing of output columns, which do not apply. Output class is determined by `x`.

Value

A data.frame, data.table, (grouped) tibble, sf, or sf-tibble, or else NULL if `do` is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# Semi- and anti-joins: basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
country pop_m
Australia 27.2
Brazil 212.0
Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
country forest_pc
```

```

Brazil      59.1
Chad        3.2
Denmark     15.8
")

# full join with `indicate = TRUE` for comparison
fjoin_full(x, y, on = "country", indicate = TRUE)

fjoin_semi(x, y, on = "country")
fjoin_anti(x, y, on = "country")
fjoin_right_semi(x, y, on = "country")
fjoin_right_anti(x, y, on = "country")

# -----
# `mult.x` and `mult.y` support
# -----

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
      1      10
      2      20
      3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
          1          30
          2          50
          3          60
")

# -----

# for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_full(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last",
  indicate = TRUE
)

fjoin_semi(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

fjoin_anti(
  events,

```

```

    reactions,
    on = c("event_ts < reaction_ts"),
    mult.x = "first",
    mult.y = "last"
  )

# -----
# Natural join
# -----
fjoin_semi(x, y, on = NA)
fjoin_anti(x, y, on = NA)

# -----
# Mock join
# -----
fjoin_semi(on="id")
fjoin_semi(on=c("id", "date"))
fjoin_semi(on=c("id"), mult.y = "last")

```

fjoin_left_semi	<i>Left semi-join</i>
-----------------	-----------------------

Description

The semi-join of x in a join of x and y , i.e. the rows of x that join at least once. The alias `fjoin_semi` can be used instead.

Usage

```

fjoin_left_semi(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  select = NULL,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

fjoin_semi(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",

```

```

  mult.y = "all",
  select = NULL,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

```

Arguments

<code>x, y</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	Whether to allow equality matches between NAs or NaNs. Default FALSE.
<code>mult.x, mult.y</code>	When a row of <code>x</code> (<code>y</code>) has multiple matching rows in <code>y</code> (<code>x</code>), which to accept: "all" (the default), "first", or "last". May be used in combination.
<code>select</code>	Character vector of non-join columns to be selected from <code>x</code> . NULL (the default) selects all columns. Join columns are always selected.
<code>do</code>	Whether to execute the join. If FALSE, <code>show</code> is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless <code>x</code> and <code>y</code> are both omitted/NULL, in which case a mock join statement is produced. See Details.
<code>show</code>	Whether to print the data.table code for the join to the console. Default is the opposite of <code>do</code> . If <code>x</code> and <code>y</code> are both omitted/NULL, mock join code is displayed.

Details

Details are as for e.g. [fjoin_inner](#) except for arguments controlling the order and prefixing of output columns, which do not apply. Output class is determined by `x`.

Value

A data.frame, data.table, (grouped) tibble, sf, or sf-tibble, or else NULL if `do` is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```

# -----
# Semi- and anti-joins: basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
country  pop_m

```

```

Australia 27.2
Brazil    212.0
Chad      3.0
")

y <- data.table::fread(data.table = FALSE, input = "
country forest_pc
Brazil    59.1
Chad      3.2
Denmark   15.8
")

# full join with `indicate = TRUE` for comparison
fjoin_full(x, y, on = "country", indicate = TRUE)

fjoin_semi(x, y, on = "country")
fjoin_anti(x, y, on = "country")
fjoin_right_semi(x, y, on = "country")
fjoin_right_anti(x, y, on = "country")

# -----
# `mult.x` and `mult.y` support
# -----

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
      1      10
      2      20
      3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
          1          30
          2          50
          3          60
")

# -----

# for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_full(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last",
  indicate = TRUE
)

fjoin_semi(
  events,
  reactions,

```

```

    on = c("event_ts < reaction_ts"),
    mult.x = "first",
    mult.y = "last"
  )

  fjoin_anti(
    events,
    reactions,
    on = c("event_ts < reaction_ts"),
    mult.x = "first",
    mult.y = "last"
  )

# -----
# Natural join
# -----
fjoin_semi(x, y, on = NA)
fjoin_anti(x, y, on = NA)

# -----
# Mock join
# -----
fjoin_semi(on="id")
fjoin_semi(on=c("id", "date"))
fjoin_semi(on=c("id"), mult.y = "last")

```

fjoin_right

Right join

Description

Right join of x and y

Usage

```

fjoin_right(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  indicate = FALSE,
  order = "left",
  select = NULL,
  select.x = NULL,
  select.y = NULL,
  prefix.y = "R.",

```

```

  on.first = FALSE,
  both = FALSE,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

```

Arguments

<code>x, y</code>	data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.
<code>on</code>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
<code>match.na</code>	Whether to allow equality matches between NAs or NaNs. Default FALSE.
<code>mult.x, mult.y</code>	When a row of <code>x</code> (<code>y</code>) has multiple matching rows in <code>y</code> (<code>x</code>), which to accept: "all" (the default), "first", or "last". May be used in combination.
<code>indicate</code>	Whether to add a column ".join" at the front of the result, with values 1L if from <code>x</code> only, 2L if from <code>y</code> only, and 3L if joined from both tables (c.f. <code>_merge</code> in Stata). Default FALSE.
<code>order</code>	Whether the row order of the result should reflect <code>x</code> then <code>y</code> ("left") or <code>y</code> then <code>x</code> ("right"). Default "left".
<code>select, select.x, select.y</code>	Character vectors of columns to be selected from either input if present (<code>select</code>) or specifically from one or other of them (e.g. <code>select.x</code>). NULL (the default) selects all columns. Use "" or NA to select no columns. Join columns are always selected. See Details.
<code>prefix.y</code>	A prefix to attach to column names in <code>y</code> that are the same as a column name in <code>x</code> . Default "R. ".
<code>on.first</code>	Whether to place the join columns first in the join result. Default FALSE.
<code>both</code>	Whether to include <code>y</code> 's equality join column(s) separately in the output, instead of combining them with <code>x</code> 's. Default FALSE. Note that non-equality join columns from <code>x</code> are always included separately.
<code>do</code>	Whether to execute the join. If FALSE, <code>show</code> is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless <code>x</code> and <code>y</code> are both omitted/NULL, in which case a mock join statement is produced. See Details.
<code>show</code>	Whether to print the data.table code for the join to the console. Default is the opposite of <code>do</code> . If <code>x</code> and <code>y</code> are both omitted/NULL, mock join code is displayed.

Details

Input and output class: Each input can be any object with class `data.frame`, or a plain list of same-length vectors.

The output class depends on `x` as follows:

- a `data.table` if `x` is a pure `data.table`

- a tibble if it is a tibble (and a grouped tibble if it has class `grouped_df`)
- an `sf` if it is an `sf` with its active geometry selected in the output
- a plain `data.frame` in all other cases

The following attributes are carried through and refreshed: `data.table` key, tibble groups, `sf` `agr` (and `bbox` etc. of all individual `sfc`-class columns regardless of output class). See below for specifics.

Specifying join conditions with `on`: `on` is a required argument. For a natural join (a join by equality on all same-named column pairs), you must specify `on = NA`; you can't just omit `on` as in other packages. This is to prevent a natural join being specified by mistake, which may then go unnoticed.

Using `select`, `select.x`, and `select.y`: Used on its own, `select` keeps the join columns plus the specified non-join columns from both inputs if present.

If `select.x` is provided (and similarly for `select.y`) then:

- if `select` is also specified, non-join columns of `x` named in either `select` or `select.x` are included
- if `select` is not specified, only non-join columns named in `select.x` are included from `x`. Thus e.g. `select.x = ""` excludes all of `x`'s non-join columns.

Non-existent column names are ignored without warning.

Column order: When `select` is specified but `select.x` and `select.y` are not, the output consists of all join columns followed by the selected non-join columns from either input in the order given in `select`.

In all other cases:

- columns from `x` come before columns from `y`
- within each group of columns, non-join columns are in the order given by `select.x/select.y`, or in their original data order if no selection is provided
- if `on.first` is `TRUE`, join columns from both inputs are moved to the front of the overall output.

Using `mult.x` and `mult.y`: See the Examples for an application of using `mult.x` and `mult.y` together. Note that `mult.y` is applied after `mult.x` except with `order = "right"`.

Displaying code and 'mock joins': The option of displaying the join code with `show = TRUE` or by passing null inputs is aimed at `data.table` users wanting to use the package as a cookbook of recipes for adaptation. If `x` and `y` are both `NULL`, template code is displayed based on join column names implied by `on`, plus sample non-join column names. `select` arguments are ignored in this case.

The code displayed is for the join operation after casting the inputs as `data.tables` if necessary, and before casting the result as a tibble and/or `sf` if applicable. Note that `fjoin` departs from the usual `j = list()` idiom in order to avoid a deep copy of the output made by `as.data.table.list`. (Likewise, internally it takes only shallow copies of columns when casting inputs or outputs to different classes.)

tibble groups: If `x` is a grouped tibble (class `grouped_df`), the output is grouped by the grouping columns that are selected in the result.

data.table keys: If the output is a `data.table`, it inherits a key as follows:

- `fjoin_inner` or `fjoin_left` with `order = "left"` (default): `x`'s key if present
- `fjoin_inner` or `fjoin_right` with `order = "right"`: `y`'s key if present

If not all of the key columns are selected in the result, the leading subset is used.

sf objects and sfc-class columns: Joins between two `sf` objects are supported. The active geometry and relation-to-geometry attribute `agr` are determined by `x`. All `sfc-class` columns in the output are refreshed after joining (using `sf::st_sfc()` with `recompute_bbox = TRUE`); this is true regardless of whether or not the inputs and output are `sfs`.

Value

A `data.frame`, `data.table`, (grouped) `tibble`, `sf`, or `sf-tibble`, or else `NULL` if `do` is `FALSE`. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# True joins (inner/left/right/full): basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
  country pop_m
Australia 27.2
  Brazil 212.0
  Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
  country forest_pc
  Brazil 59.1
  Chad 3.2
  Denmark 15.8
")

# -----
# `indicate = TRUE` adds a front column ".join" indicating whether a row is
# from `x` only (1L), from `y` only (2L), or joined from both (3L)

fjoin_full(x, y, on = "country", indicate = TRUE)
fjoin_left(x, y, on = "country", indicate = TRUE)
fjoin_right(x, y, on = "country", indicate = TRUE)
fjoin_inner(x, y, on = "country", indicate = TRUE)

# -----
# Core options and arguments (in a 1:1 equality join with fjoin_full())
```

```

# -----

# data frames
dfQ <- data.table::fread(data.table = FALSE, quote = "'", input = "
id quantity          notes other_cols
  2      5              ''          ...
  1      6              ''          ...
  3      7              ''          ...
NA      8 'oranges (not listed)'    ...
")

dfP <- data.table::fread(data.table = FALSE, input = "
id  item price other_cols
NA  apples  10     ...
  3 bananas  20     ...
  2 cherries 30     ...
  1  dates  40     ...
")

# -----

# (1) basic syntax
# cf. dplyr: full_join(dfQ, dfP, join_by(id), na.matches = "never")
fjoin_full(dfQ, dfP, on = "id")

# (an aside) equality matches on NA if you insist
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity", "notes"), match.na = TRUE)

# (2) join-select in one line
fjoin_full(dfQ, dfP, on = "id", select = c("item", "price", "quantity"))

# equivalent operation in dplyr
# x <- dfQ |> select(id, quantity)
# y <- dfP |> select(id, item, price)
# full_join(x, y, join_by(id), na.matches = "never") |>
#   select(id, item, price, quantity)

# (3) indicator column (in Stata since 1984)
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE
)

# (4) order rows by y then x
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,

```

```

    order = "right"
  )

# (5) display code instead
fjoin_full(
  dfQ,
  dfP,
  on = "id",
  select = c("item", "price", "quantity"),
  indicate = TRUE,
  order = "right",
  do = FALSE
)

# -----
# M:M inequality join reduced to 1:1 using `mult.x` and `mult.y`
# -----

# data.table (`mult`) and dplyr (`multiple`) have options for reducing the
# cardinality on one side of the join from many ("all") to one ("first" or
# "last"). fjoin (`mult.x`, `mult.y`) permits this on either side of the
# join, or on both sides at once.

# This example (using `fjoin_left()`) shows an application to temporally
# ordered data frames of "events" and "reactions".

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
  1      10
  2      20
  3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
  1          30
  2          50
  3          60
")

# -----

# (1) for each event, all subsequent reactions (M:M)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
)

# (2) for each event, the next reaction (1:M)
fjoin_left(
  events,
  reactions,

```

```

    on = c("event_ts < reaction_ts"),
    mult.x = "first"
  )

# (3) for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_left(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

# -----
# Natural join
# -----
fjoin_inner(x, y, on = NA) # note `NA` not `NULL`/omitted
try(fjoin_inner(x, y)) # to prevent accidental natural joins

# -----
# Mock join (code "ghostwriter" for data.table users)
# -----
fjoin_inner(on = c("id"))

```

fjoin_right_anti *Right anti-join*

Description

The anti-join of *y* in a join of *x* and *y*, i.e. the rows of *y* that do not join.

Usage

```

fjoin_right_anti(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  select = NULL,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

```

Arguments

x, *y* data.frame-like objects (plain, data.table, tibble, sf, list, etc.) or else both omitted for a mock join statement with no data. See Details.

on	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else NA for a natural join (an equality join on all same-named columns).
match.na	Whether to allow equality matches between NAs or NaNs. Default FALSE.
mult.x, mult.y	When a row of x (y) has multiple matching rows in y (x), which to accept: "all" (the default), "first", or "last". May be used in combination.
select	Character vector of columns to be selected from y. NULL (the default) selects all columns. Join columns are always selected.
do	Whether to execute the join. If FALSE, show is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless x and y are both omitted/NULL, in which case a mock join statement is produced. See Details.
show	Whether to print the data.table code for the join to the console. Default is the opposite of do. If x and y are both omitted/NULL, mock join code is displayed.

Details

Details are as for e.g. [fjoin_inner](#) except for arguments controlling the order and prefixing of output columns, which do not apply. Output class is determined by y.

Value

A `data.frame`, `data.table`, (grouped) tibble, `sf`, or `sf-tibble`, or else NULL if do is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# Semi- and anti-joins: basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
country pop_m
Australia 27.2
Brazil 212.0
Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
country forest_pc
Brazil 59.1
Chad 3.2
Denmark 15.8
")
```

```

# full join with `indicate = TRUE` for comparison
fjoin_full(x, y, on = "country", indicate = TRUE)

fjoin_semi(x, y, on = "country")
fjoin_anti(x, y, on = "country")
fjoin_right_semi(x, y, on = "country")
fjoin_right_anti(x, y, on = "country")

# -----
# `mult.x` and `mult.y` support
# -----

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
  1      10
  2      20
  3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
  1          30
  2          50
  3          60
")
# -----

# for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_full(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last",
  indicate = TRUE
)

fjoin_semi(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

fjoin_anti(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

```

```

)

# -----
# Natural join
# -----
fjoin_semi(x, y, on = NA)
fjoin_anti(x, y, on = NA)

# -----
# Mock join
# -----
fjoin_semi(on="id")
fjoin_semi(on=c("id", "date"))
fjoin_semi(on=c("id"), mult.y = "last")

```

fjoin_right_semi	<i>Right semi-join</i>
------------------	------------------------

Description

The semi-join of *y* in a join of *x* and *y*, i.e. the rows of *y* that join at least once.

Usage

```

fjoin_right_semi(
  x = NULL,
  y = NULL,
  on,
  match.na = FALSE,
  mult.x = "all",
  mult.y = "all",
  select = NULL,
  do = !(is.null(x) && is.null(y)),
  show = !do
)

```

Arguments

<i>x</i> , <i>y</i>	<i>data.frame</i> -like objects (plain, <i>data.table</i> , <i>tibble</i> , <i>sf</i> , <i>list</i> , etc.) or else both omitted for a mock join statement with no data. See Details.
<i>on</i>	A character vector of join predicates, e.g. <code>c("id", "col_x == col_y", "date > date", "cost <= budget")</code> , or else <code>NA</code> for a natural join (an equality join on all same-named columns).
<i>match.na</i>	Whether to allow equality matches between NAs or NaNs. Default <code>FALSE</code> .
<i>mult.x</i> , <i>mult.y</i>	When a row of <i>x</i> (<i>y</i>) has multiple matching rows in <i>y</i> (<i>x</i>), which to accept: "all" (the default), "first", or "last". May be used in combination.

select	Character vector of columns to be selected from y. NULL (the default) selects all columns. Join columns are always selected.
do	Whether to execute the join. If FALSE, show is set to TRUE and the data.table code for the join is printed to the console instead. Default is TRUE unless x and y are both omitted/NULL, in which case a mock join statement is produced. See Details.
show	Whether to print the data.table code for the join to the console. Default is the opposite of do. If x and y are both omitted/NULL, mock join code is displayed.

Details

Details are as for e.g. [fjoin_inner](#) except for arguments controlling the order and prefixing of output columns, which do not apply. Output class is determined by y.

Value

A data.frame, data.table, (grouped) tibble, sf, or sf-tibble, or else NULL if do is FALSE. See Details.

See Also

See the package-level documentation [fjoin](#) for related functions.

Examples

```
# -----
# Semi- and anti-joins: basic usage
# -----

# data frames
x <- data.table::fread(data.table = FALSE, input = "
country pop_m
Australia 27.2
Brazil 212.0
Chad 3.0
")

y <- data.table::fread(data.table = FALSE, input = "
country forest_pc
Brazil 59.1
Chad 3.2
Denmark 15.8
")

# full join with `indicate = TRUE` for comparison
fjoin_full(x, y, on = "country", indicate = TRUE)

fjoin_semi(x, y, on = "country")
fjoin_anti(x, y, on = "country")
fjoin_right_semi(x, y, on = "country")
fjoin_right_anti(x, y, on = "country")
```

```

# -----
# `mult.x` and `mult.y` support
# -----

# data frames
events <- data.table::fread(data.table = FALSE, input = "
event_id event_ts
      1      10
      2      20
      3      40
")

reactions <- data.table::fread(data.table = FALSE, input = "
reaction_id reaction_ts
          1          30
          2          50
          3          60
")

# -----

# for each event, the next reaction, provided there was no intervening event (1:1)
fjoin_full(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last",
  indicate = TRUE
)

fjoin_semi(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

fjoin_anti(
  events,
  reactions,
  on = c("event_ts < reaction_ts"),
  mult.x = "first",
  mult.y = "last"
)

# -----
# Natural join
# -----
fjoin_semi(x, y, on = NA)
fjoin_anti(x, y, on = NA)

```

```
# -----  
# Mock join  
# -----  
fjoin_semi(on="id")  
fjoin_semi(on=c("id", "date"))  
fjoin_semi(on=c("id"), mult.y = "last")
```

Index

dtjoin, [2](#), [7](#), [9](#), [11](#)
dtjoin_anti, [6](#)
dtjoin_cross, [8](#)
dtjoin_semi, [10](#)

fjoin, [6](#), [8](#), [9](#), [11](#), [13](#), [16](#), [22](#), [28](#), [32](#), [35](#), [40](#),
[44](#), [47](#)
fjoin_anti (fjoin_left_anti), [31](#)
fjoin_cross, [8](#), [12](#)
fjoin_full, [2](#), [13](#)
fjoin_inner, [2](#), [12](#), [19](#), [32](#), [35](#), [44](#), [47](#)
fjoin_left, [2](#), [25](#)
fjoin_left_anti, [6](#), [31](#)
fjoin_left_semi, [10](#), [34](#)
fjoin_right, [2](#), [37](#)
fjoin_right_anti, [6](#), [43](#)
fjoin_right_semi, [10](#), [46](#)
fjoin_semi (fjoin_left_semi), [34](#)