Package ‘fjoin’

December 11, 2025

Type Package

Title Data Frame Joins Leveraging 'data.table'

Version 0.1.0

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 2025-12-11 13:30:08 UTC

Contents

`dtjoin` *Join data frame-like objects using an extended* `DT[i]`*-style interface to data.table*

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

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

on.first	Whether to place the join columns from both inputs first in the join result. Default FALSE.
i.home	Whether to treat .i as the "home" table and .DT as the "foreign" table for column prefixing and indicate. Default FALSE, i.e. .DT is the "home" table, as in [.data.table.
i.first	Whether to place .i's columns before .DT's in the join result. The default is to use the value of i.home, i.e. bring .i's columns to the front if .i is the "home" table.
prefix	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 "i." if the "foreign" table is .i (i.home is FALSE) and "x." if it is .DT (i.home is TRUE).
i.class	Whether the class of the output should be based on .i instead of .DT. The default follows i.home (default FALSE). See Details for how output class and other attributes are set.
do	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.
show	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.
verbose	(passed to [.data.table) Whether data.table should print information to the console during execution. Default FALSE.
...	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 sfc-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

`.DT`, `.i`	`data.frame`-like objects (plain, `data.table`, tibble, `sf`, `list`, etc.), or else both omitted for a mock join statement with no data.
`on`	A character vector of join predicates, e.g. `c("id", "col_DT == col_i", "date < date", "cost <= budget")`, or else `NA` for a natural join (an equality join on all same-named columns).
`match.na`	If `TRUE`, allow equality matches between `NA`s or `NaN`s. Default `FALSE`.
`mult`	(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"`.
`mult.DT`	Permitted for consistency with `dtjoin` but has no effect on the resulting semi-join.
`nomatch`, `nomatch.DT`	Permitted for consistency with `dtjoin` but have no effect on the resulting semi-join.
`select`	Character vector of columns of `.DT` to be selected. `NULL` (the default) selects all columns. Join columns are always selected.
`do`	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.
`show`	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.
`verbose`	(passed to `[.data.table`) Whether data.table should print information to the console during execution. Default `FALSE`.
`...`	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")
```

---

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

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

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

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

match.na	If TRUE, allow equality matches between NAs or NaNs. Default FALSE.
mult	(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".
mult.DT	Permitted for consistency with dtjoin but has no effect on the resulting semi-join.
nomatch, nomatch.DT	Permitted for consistency with dtjoin but have no effect on the resulting semi-join.
select	Character vector of columns of .DT to be selected. NULL (the default) selects all columns. Join columns are always selected.
do	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.
show	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.
verbose	(passed to [.data.table) Whether data.table should print information to the console during execution. Default FALSE.
...	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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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.
on.first	Whether to place the join columns first in the join result. Default FALSE.
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.
indicate	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.
prefix.y	A prefix to attach to column names in y that are the same as a column name in x. Default "R.".
both	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.
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

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 `sf`s.

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")

# (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)
# ------------------------------------------------------------------------

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

# (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_left(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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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.
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.
indicate	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.
prefix.y	A prefix to attach to column names in y that are the same as a column name in x. Default "R.".
on.first	Whether to place the join columns first in the join result. Default FALSE.
both	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.
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

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 `sf`s.

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")

# (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)
# ------------------------------------------------------------------------

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

# (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_left(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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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.
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.
indicate	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.
prefix.y	A prefix to attach to column names in y that are the same as a column name in x. Default "R.".
on.first	Whether to place the join columns first in the join result. Default FALSE.
both	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.
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

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 `key`s: 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 `sf`s.

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")

# (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)
# ------------------------------------------------------------------------------

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

# (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_left(x, y)) # to prevent accidental natural joins

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

`fjoin_left_anti` *Left anti-join*

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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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 non-join columns to be selected from x. 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 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` *Left semi-join*

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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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 non-join columns to be selected from x. 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 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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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.
indicate	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.
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.".
on.first	Whether to place the join columns first in the join result. Default FALSE.
both	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.
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

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")

# (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)
# ------------------------------------------------------------------------------

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

# (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_left(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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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` *Right semi-join*

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

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. c("id", "col_x == col_y", "date > date", "cost <= budget"), 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")
```

Index