Package 'disk.frame'

July 7, 2020

Type Package

Title Larger-than-RAM Disk-Based Data Manipulation Framework

Version 0.3.7

Date 2020-07-07

Maintainer Dai ZJ <zhuojia.dai@gmail.com>

Description A disk-based data manipulation tool for working with large-than-RAM datasets. Aims to lower the barrier-to-entry for manipulating large datasets by adhering closely to popular and familiar data manipulation paradigms like 'dplyr' verbs and 'data.table' syntax.

License MIT + file LICENSE

Imports Rcpp (>= 0.12.13), glue (>= 1.3.1), rlang (>= 0.4.0), furrr (>= 0.1.0), future.apply (>= 1.3.0), fs (>= 1.3.1), jsonlite (>= 1.6), pryr (>= 0.1.4), stringr (>= 1.4.0), fst (>= 0.8.0), globals (>= 0.12.4), future (>= 1.14.0), data.table (>= 1.12.2), crayon (>= 1.3.4), bigreadr (>= 0.2.0), bit64, benchmarkme

Depends R (>= 3.4), dplyr (>= 1.0.0), purrr (>= 0.3.2)

Suggests testthat (>= 2.1.0), nycflights13, magrittr, shiny, LaF, readr, rstudioapi, biglm, biglmm, speedglm, broom, ggplot2, covr

LinkingTo Rcpp

RoxygenNote 7.1.0

Encoding UTF-8

URL https://diskframe.com

BugReports https://github.com/xiaodaigh/disk.frame/issues

NeedsCompilation yes

Author Dai ZJ [aut, cre], Jacky Poon [ctb]

Repository CRAN

Date/Publication 2020-07-07 13:10:03 UTC

R topics documented:

---

add_chunk *Add a chunk to the disk.frame*

Description

If no chunk_id is specified, then the chunk is added at the end as the largest numbered file, "n.fst".

Usage

```
add_chunk(df, chunk, chunk_id = NULL, full.names = FALSE, ...)
```

Arguments

df	the disk.frame to add a chunk to
chunk	a data.frame to be added as a chunk
chunk_id	a numeric number indicating the id of the chunk. If NULL it will be set to the largest chunk_id + 1
full.names	whether the chunk_id name match should be to the full file path not just the file name
...	Passed in the write_fst. E.g. compress

Details

The function is the preferred way to add a chunk to a disk.frame. It performs checks on the types to make sure that the new chunk doesn't have different types to the disk.frame.

Value

disk.frame

Examples

```
# create a disk.frame
df_path = file.path(tempdir(), "tmp_add_chunk")
diskf = disk.frame(df_path)

# add a chunk to diskf
add_chunk(diskf, cars)
add_chunk(diskf, cars)

nchunks(diskf) # 2

df2 = disk.frame(file.path(tempdir(), "tmp_add_chunk2"))

# add chunks by specifying the chunk_id number; this is especially useful if
# you wish to add multiple chunk in parralel

add_chunk(df2, data.frame(chunk=1), 1)
add_chunk(df2, data.frame(chunk=2), 3)

nchunks(df2) # 2

dir(attr(df2, "path", exact=TRUE))
# [1] "1.fst" "3.fst"

# clean up
delete(diskf)
delete(df2)
```

---

anti_join.disk.frame *Performs join/merge for disk.frames*

Description

Performs join/merge for disk.frames

Usage

```
## S3 method for class 'disk.frame'
anti_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_anti_join"),
  merge_by_chunk_id = FALSE,
  overwrite = TRUE,
  .progress = FALSE
)
```

```
## S3 method for class 'disk.frame'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_full_join"),
  overwrite = TRUE,
  merge_by_chunk_id,
  .progress = FALSE
)

## S3 method for class 'disk.frame'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_inner_join"),
  merge_by_chunk_id = NULL,
  overwrite = TRUE,
  .progress = FALSE
)

## S3 method for class 'disk.frame'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_left_join"),
  merge_by_chunk_id = FALSE,
  overwrite = TRUE,
  .progress = FALSE
)

## S3 method for class 'disk.frame'
semi_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_semi_join"),
```

```
  merge_by_chunk_id = FALSE,
  overwrite = TRUE,
  .progress = FALSE
)
```

Arguments

x	a disk.frame
y	a data.frame or disk.frame. If data.frame then returns lazily; if disk.frame it performs the join eagerly and return a disk.frame
by	join by
copy	same as dplyr::anti_join
...	same as dplyr's joins
outdir	output directory for disk.frame
merge_by_chunk_id	the merge is performed by chunk id
overwrite	overwrite output directory
.progress	Show progress or not. Defaults to FALSE

Value

disk.frame or data.frame/data.table

Examples

```
df.df = as.disk.frame(data.frame(x = 1:3, y = 4:6), overwrite = TRUE)
df2.df = as.disk.frame(data.frame(x = 1:2, z = 10:11), overwrite = TRUE)

anti_joined.df = anti_join(df.df, df2.df)

anti_joined.df %>% collect

anti_joined.data.frame = anti_join(df.df, data.frame(x = 1:2, z = 10:11))

# clean up
delete(df.df)
delete(df2.df)
delete(anti_joined.df)
cars.df = as.disk.frame(cars)

join.df = full_join(cars.df, cars.df, merge_by_chunk_id = TRUE)

# clean up cars.df
delete(cars.df)
delete(join.df)
cars.df = as.disk.frame(cars)

join.df = inner_join(cars.df, cars.df, merge_by_chunk_id = TRUE)
```

```
# clean up cars.df
delete(cars.df)
delete(join.df)
cars.df = as.disk.frame(cars)

join.df = left_join(cars.df, cars.df)

# clean up cars.df
delete(cars.df)
delete(join.df)
cars.df = as.disk.frame(cars)

join.df = semi_join(cars.df, cars.df)

# clean up cars.df
delete(cars.df)
delete(join.df)
```

`as.data.frame.disk.frame` *Convert disk.frame to data.frame by collecting all chunks*

Description

Convert disk.frame to data.frame by collecting all chunks

Usage

```
## S3 method for class 'disk.frame'
as.data.frame(x, row.names, optional, ...)
```

Arguments

`x`	a disk.frame
`row.names`	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's base package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional). See also the make.names argument of the matrix method.
`...`	additional arguments to be passed to or from methods.

Examples

```
cars.df = as.disk.frame(cars)
as.data.frame(cars.df)

# clean up
delete(cars.df)
```

`as.data.table.disk.frame` *Convert disk.frame to data.table by collecting all chunks*

Description

Convert disk.frame to data.table by collecting all chunks

Usage

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

Arguments

`x`	a disk.frame
`keep.rownames`	passed to as.data.table
`...`	passed to as.data.table

Examples

```
library(data.table)
cars.df = as.disk.frame(cars)
as.data.table(cars.df)

# clean up
delete(cars.df)
```

`as.disk.frame` *Make a data.frame into a disk.frame*

Description

Make a data.frame into a disk.frame

Usage

```
as.disk.frame(
  df,
  outdir = tempfile(fileext = ".df"),
  nchunks = recommend_nchunks(df),
  overwrite = FALSE,
  shardby = NULL,
  compress = 50,
  ...
)
```

Arguments

df	a disk.frame
outdir	the output directory
nchunks	number of chunks
overwrite	if TRUE the outdir will be overwritten, if FALSE it will throw an error if the directory is not empty
shardby	The shardkey
compress	the compression level 0-100; 100 is highest
...	passed to output_disk.frame

Examples

```
# write to temporary location
cars.df = as.disk.frame(cars)

# specify a different path in the temporary folder, you are free to choose a different folder
cars_new_location.df = as.disk.frame(cars, outdir = file.path(tempdir(), "some_path.df"))

# specify a different number of chunks
# this writes to tempdir() by default
cars_chunks.df = as.disk.frame(cars, nchunks = 4, overwrite = TRUE)

# clean up
delete(cars.df)
delete(cars_new_location.df)
delete(cars_chunks.df)
```

---

chunk_summarize *Group by within each disk.frame*

Description

The disk.frame group by operation perform group WITHIN each chunk. This is often used for performance reasons. If the user wishes to perform group-by, they may choose to use the ‘hard_group_by‘ function which is expensive as it reorganizes the chunks by the shard key.

Usage

```
chunk_summarize(.data, ...)

chunk_summarise(.data, ...)

chunk_group_by(.data, ...)

chunk_ungroup(.data, ...)
```

Arguments

.data	a disk.frame
...	passed to dplyr::group_by

See Also

hard_group_by group_by

---

cmap *Apply the same function to all chunks*

Description

Apply the same function to all chunks

'cimap.disk.frame' accepts a two argument function where the first argument is a data.frame and the second is the chunk ID

'lazy' is convenience function to apply '.f' to every chunk

'delayed' is an alias for lazy and is consistent with the naming in Dask and Dagger.jl

Usage

```
cmap(.x, .f, ...)

## S3 method for class 'disk.frame'
cmap(
  .x,
  .f,
  ...,
  outdir = NULL,
  keep = NULL,
  chunks = nchunks(.x),
  compress = 50,
  lazy = TRUE,
  overwrite = FALSE,
  vars_and_pkgs = future::getGlobalsAndPackages(.f, envir = parent.frame()),
```

```
  .progress = TRUE
)

cmap_dfr(.x, .f, ..., .id = NULL)

## S3 method for class 'disk.frame'
cmap_dfr(.x, .f, ..., .id = NULL, use.names = fill, fill = FALSE, idcol = NULL)

cimap(.x, .f, ...)

## S3 method for class 'disk.frame'
cimap(
  .x,
  .f,
  outdir = NULL,
  keep = NULL,
  chunks = nchunks(.x),
  compress = 50,
  lazy = TRUE,
  overwrite = FALSE,
  ...
)

cimap_dfr(.x, .f, ..., .id = NULL)

## S3 method for class 'disk.frame'
cimap_dfr(
  .x,
  .f,
  ...,
  .id = NULL,
  use.names = fill,
  fill = FALSE,
  idcol = NULL
)

lazy(.x, .f, ...)

## S3 method for class 'disk.frame'
lazy(.x, .f, ...)

delayed(.x, .f, ...)

chunk_lapply(...)

map(.x, .f, ...)

## S3 method for class 'disk.frame'
```

```
map(...)

## Default S3 method:
map(.x, .f, ...)

imap_dfr(.x, .f, ..., .id = NULL)

## S3 method for class 'disk.frame'
imap_dfr(...)

## Default S3 method:
imap_dfr(.x, .f, ..., .id = NULL)

imap(.x, .f, ...)

## Default S3 method:
imap(.x, .f, ...)

## S3 method for class 'disk.frame'
map_dfr(...)

## Default S3 method:
map_dfr(.x, .f, ..., .id = NULL)
```

Arguments

.x	a disk.frame
.f	a function to apply to each of the chunks
...	for compatibility with ‘purrr::map‘
outdir	the output directory
keep	the columns to keep from the input
chunks	The number of chunks to output
compress	0-100 fst compression ratio
lazy	if TRUE then do this lazily
overwrite	if TRUE removes any existing chunks in the data
vars_and_pkgs	variables and packages to send to a background session. This is typically automatically detected
.progress	A logical, for whether or not to print a progress bar for multiprocess, multisession, and multicore plans. From furrr
.id	not used
use.names	for cmap_dfr’s call to data.table::rbindlist. See data.table::rbindlist
fill	for cmap_dfr’s call to data.table::rbindlist. See data.table::rbindlist
idcol	for cmap_dfr’s call to data.table::rbindlist. See data.table::rbindlist

Examples

```
cars.df = as.disk.frame(cars)

# return the first row of each chunk lazily
#
cars2 = cmap(cars.df, function(chunk) {
 chunk[,1]
})

collect(cars2)

# same as above but using purrr
cars2 = cmap(cars.df, ~.x[1,])

collect(cars2)

# return the first row of each chunk eagerly as list
cmap(cars.df, ~.x[1,], lazy = FALSE)

# return the first row of each chunk eagerly as data.table/data.frame by row-binding
cmap_dfr(cars.df, ~.x[1,])

# lazy and delayed are just an aliases for cmap(..., lazy = TRUE)
collect(lazy(cars.df, ~.x[1,]))
collect(delayed(cars.df, ~.x[1,]))

# clean up cars.df
delete(cars.df)
cars.df = as.disk.frame(cars)

# .x is the chunk and .y is the ID as an integer

# lazy = TRUE support is not available at the moment
cimap(cars.df, ~.x[, id := .y], lazy = FALSE)

cimap_dfr(cars.df, ~.x[, id := .y])

# clean up cars.df
delete(cars.df)
```

---

cmap2 *'cmap2' a function to two disk.frames*

Description

Perform a function on both disk.frames .x and .y, each chunk of .x and .y gets run by .f(x.chunk, y.chunk)

Usage

```
cmap2(.x, .y, .f, ...)

map2(.x, .y, .f, ...)

map_by_chunk_id(.x, .y, .f, ..., outdir)
```

Arguments

.x	a disk.frame
.y	a disk.frame
.f	a function to be called on each chunk of x and y matched by chunk_id
...	not used
outdir	output directory

Examples

```
cars.df = as.disk.frame(cars)

cars2.df = cmap2(cars.df, cars.df, ~data.table::rbindlist(list(.x, .y)))
collect(cars2.df)

# clean up cars.df
delete(cars.df)
delete(cars2.df)
```

---

collect.disk.frame *Bring the disk.frame into R*

Description

Bring the disk.frame into RAM by loading the data and running all lazy operations as data.table/data.frame or as a list

Bring the disk.frame into RAM by loading the data and running all lazy operations as data.table/data.frame or as a list

Usage

```
## S3 method for class 'disk.frame'
collect(x, ..., parallel = !is.null(attr(x, "lazyfn")))

collect_list(x, simplify = FALSE, parallel = !is.null(attr(x, "lazyfn")))

## S3 method for class 'summarized_disk.frame'
collect(x, ..., parallel = !is.null(attr(x, "lazyfn")))
```

Arguments

x	a disk.frame
...	not used
parallel	if TRUE the collection is performed in parallel. By default if there are delayed/lazy steps then it will be parallel, otherwise it will not be in parallel. This is because parallel requires transferring data from background R session to the current R session and if there is no computation then it's better to avoid transferring data between session, hence parallel = FALSE is a better choice
simplify	Should the result be simplified to array

Value

collect return a data.frame/data.table

collect_list returns a list

collect return a data.frame/data.table

Examples

```
cars.df = as.disk.frame(cars)
# use collect to bring the data into RAM as a data.table/data.frame
collect(cars.df)

# clean up
delete(cars.df)
cars.df = as.disk.frame(cars)

# returns the result as a list
collect_list(cmap(cars.df, ~1))

# clean up
delete(cars.df)
cars.df = as.disk.frame(cars)
# use collect to bring the data into RAM as a data.table/data.frame
collect(cars.df)

# clean up
delete(cars.df)
```

---

colnames *Return the column names of the disk.frame*

Description

The returned column names are from the source. So if you have lazy operations then the colnames here does not reflects the results of those operations. To obtain the correct names try `names(collect(get_chunk(df,1)))`

Usage

```
colnames(x, ...)

## S3 method for class 'disk.frame'
names(x, ...)

## S3 method for class 'disk.frame'
colnames(x, ...)

## Default S3 method:
colnames(x, ...)
```

Arguments

x	a disk.frame
...	not used

---

compute.disk.frame *Compute without writing*

Description

Perform the computation; same as calling cmap without .f and lazy = FALSE

Usage

```
## S3 method for class 'disk.frame'
compute(
  x,
  name,
  outdir = tempfile("tmp_df_", fileext = ".df"),
  overwrite = TRUE,
  ...
)
```

Arguments

x	a disk.frame
name	Not used. Kept for compatibility with dplyr
outdir	the output directory
overwrite	whether to overwrite or not
...	Not used. Kept for dplyr compatibility

Examples

```
cars.df = as.disk.frame(cars)
cars.df2 = cars.df %>% cmap(~.x)
# the computation is performed and the data is now stored elsewhere
cars.df3 = compute(cars.df2)

# clean up
delete(cars.df)
delete(cars.df3)
```

`create_chunk_mapper` *Create function that applies to each chunk if disk.frame*

Description

A function to make it easier to create functions like `filter`

Usage

```
create_chunk_mapper(chunk_fn, warning_msg = NULL, as.data.frame = TRUE)
```

Arguments

`chunk_fn`	The dplyr function to create a mapper for
`warning_msg`	The warning message to display when invoking the mapper
`as.data.frame`	force the input chunk of a data.frame; needed for dtplyr

Examples

```
filter = create_chunk_mapper(dplyr::filter)

#' example: creating a function that keeps only the first and last n row
first_and_last <- function(chunk, n, ...) {
  nr = nrow(chunk)
  print(nr-n+1:nr)
  chunk[c(1:n, (nr-n+1):nr), ]
}

#' create the function for use with disk.frame
first_and_last_df = create_chunk_mapper(first_and_last)

mtcars.df = as.disk.frame(mtcars)

#' the operation is lazy
lazy_mtcars.df = mtcars.df %>%
  first_and_last_df(2)
```

```
#' bring into R
collect(lazy_mtcars.df)

#' clean up
delete(mtcars.df)
```

`create_dplyr_mapper` *Kept for backwards-compatibility to be removed in 0.3*

Description

Kept for backwards-compatibility to be removed in 0.3

Usage

```
create_dplyr_mapper()
```

`csv_to_disk.frame` *Convert CSV file(s) to disk.frame format*

Description

Convert CSV file(s) to disk.frame format

Usage

```
csv_to_disk.frame(
  infile,
  outdir = tempfile(fileext = ".df"),
  inmapfn = base::I,
  nchunks = recommend_nchunks(sum(file.size(infile))),
  in_chunk_size = NULL,
  shardby = NULL,
  compress = 50,
  overwrite = TRUE,
  header = TRUE,
  .progress = TRUE,
  backend = c("data.table", "readr", "LaF"),
  chunk_reader = c("bigreadr", "data.table", "readr", "readLines"),
  ...
)
```

Arguments

infile	The input CSV file or files
outdir	The directory to output the disk.frame to
inmapfn	A function to be applied to the chunk read in from CSV before the chunk is being written out. Commonly used to perform simple transformations. Defaults to the identity function (ie. no transformation)
nchunks	Number of chunks to output
in_chunk_size	When reading in the file, how many lines to read in at once. This is different to nchunks which controls how many chunks are output
shardby	The column(s) to shard the data by. For example suppose 'shardby = c("col1","col2")' then every row where the values 'col1' and 'col2' are the same will end up in the same chunk; this will allow merging by 'col1' and 'col2' to be more efficient
compress	For fst backends it's a number between 0 and 100 where 100 is the highest compression ratio.
overwrite	Whether to overwrite the existing directory
header	Whether the files have header. Defaults to TRUE
.progress	A logical, for whether or not to print a progress bar for multiprocess, multisession, and multicore plans. From furrr
backend	The CSV reader backend to choose: "data.table" or "readr". disk.frame does not have its own CSV reader. It uses either data.table::fread or readr::read_delimited. It is worth noting that data.table::fread does not detect dates and all dates are imported as strings, and you are encouraged to use fasttime to convert the strings to date. You can use the 'inmapfn' to do that. However, if you want automatic date detection, then backend="readr" may suit your needs. However, readr is often slower than data.table, hence data.table is chosen as the default.
chunk_reader	Even if you choose a backend there can still be multiple strategies on how to approach the CSV reads. For example, data.table::fread tries to mmap the whole file which can cause the whole read process to fail. In that case we can change the chunk_reader to "readLines" which uses the readLines function to read chunk by chunk and still use data.table::fread to process the chunks. There are currently no strategies for readr backend, except the default one.
...	passed to data.table::fread, disk.frame::as.disk.frame, disk.frame::shard

See Also

Other ingesting data: zip_to_disk.frame()

Examples

```
tmpfile = tempfile()
write.csv(cars, tmpfile)
tmpdf = tempfile(fileext = ".df")
df = csv_to_disk.frame(tmpfile, outdir = tmpdf, overwrite = TRUE)

# clean up
fs::file_delete(tmpfile)
delete(df)
```

delete *Delete a disk.frame*

Description

Delete a disk.frame

Usage

```
delete(df)
```

Arguments

df	a disk.frame

Examples

```
cars.df = as.disk.frame(cars)
delete(cars.df)
```

dfglm *Fit generalized linear models (glm) with disk.frame*

Description

Fits GLMs using ‘speedglm‘ or ‘biglm‘. The return object will be exactly as those return by those functions. This is a convenience wrapper

Usage

```
dfglm(formula, data, ..., glm_backend = c("biglm", "speedglm", "biglmm"))
```

Arguments

formula	A model formula
data	See Details below. Method dispatch is on this argument
...	Additional arguments
glm_backend	Which package to use for fitting GLMs. The default is "biglm", which has known issues with factor level if different levels are present in different chunks. The "speedglm" option is more robust, but does not implement ‘predict‘ which makes prediction and implementation impossible.

Details

The data argument may be a function, a data frame, or a SQLiteConnection or RODBC connection object.

When it is a function the function must take a single argument reset. When this argument is FALSE it returns a data frame with the next chunk of data or NULL if no more data are available. Whenreset=TRUE it indicates that the data should be reread from the beginning by subsequent calls. The chunks need not be the same size or in the same order when the data are reread, but the same data must be provided in total. The bigglm.data.frame method gives an example of how such a function might be written, another is in the Examples below.

The model formula must not contain any data-dependent terms, as these will not be consistent when updated. Factors are permitted, but the levels of the factor must be the same across all data chunks (empty factor levels are ok). Offsets are allowed (since version 0.8).

The SQLiteConnection and RODBC methods loads only the variables needed for the model, not the whole table. The code in the SQLiteConnection method should work for other DBI connections, but I do not have any of these to check it with.

Value

An object of class bigglm

References

Algorithm AS274 Applied Statistics (1992) Vol.41, No. 2

See Also

Other Machine Learning (ML): make_glm_streaming_fn()

Examples

```
cars.df = as.disk.frame(cars)
m = dfglm(dist ~ speed, data = cars.df)

# can use normal R functions
# Only works in version > R 3.6
majorv = as.integer(version$major)
minorv = as.integer(strsplit(version$minor, ".", fixed=TRUE)[[1]][1])
if(((majorv == 3) & (minorv >= 6)) | (majorv > 3)) {
  summary(m)
  predict(m, get_chunk(cars.df, 1))
  predict(m, collect(cars.df))
  # can use broom to tidy up the returned info
  broom::tidy(m)
}

# clean up
delete(cars.df)
```

df_ram_size *Get the size of RAM in gigabytes*

Description

Get the size of RAM in gigabytes

Usage

```
df_ram_size()
```

Value

integer of RAM in gigabyte (GB)

Examples

```
# returns the RAM size in gigabyte (GB)
df_ram_size()
```

disk.frame *Create a disk.frame from a folder*

Description

Create a disk.frame from a folder

Usage

```
disk.frame(path, backend = "fst")
```

Arguments

path	The path to store the output file or to a directory
backend	The only available backend is fst at the moment

Examples

```
path = file.path(tempdir(),"cars")
as.disk.frame(cars, outdir=path, overwrite = TRUE, nchunks = 2)
df = disk.frame(path)
head(df)
nchunks(df)
# clean up
delete(df)
```

evalparseglue *Helper function to evalparse some ‘glue::glue‘ string*

Description

Helper function to evalparse some ‘glue::glue‘ string

Usage

```
evalparseglue(code, env = parent.frame())
```

Arguments

code	the code in character(string) format to evaluate
env	the environment in which to evaluate the code

foverlaps.disk.frame *Apply data.table's foverlaps to the disk.frame*

Description

EXPERIMENTAL

Usage

```
foverlaps.disk.frame(
  df1,
  df2,
  by.x = if (identical(shardkey(df1)$shardkey, "")) shardkey(df1)$shardkey else
    shardkey(df2)$shardkey,
  by.y = shardkey(df2)$shardkey,
  ...,
  outdir = tempfile("df_foverlaps_tmp", fileext = ".df"),
  merge_by_chunk_id = FALSE,
  compress = 50,
  overwrite = TRUE
)
```

Arguments

df1	A disk.frame
df2	A disk.frame or a data.frame
by.x	character/string vector. by.x used in foverlaps
by.y	character/string vector. by.x used in foverlaps

...	passed to data.table::foverlaps and disk.frame::cmap.disk.frame
outdir	The output directory of the disk.frame
merge_by_chunk_id	If TRUE then the merges will happen for chunks in df1 and df2 with the same chunk id which speed up processing. Otherwise every chunk of df1 is merged with every chunk of df2. Ignored with df2 is not a disk.frame
compress	The compression ratio for fst
overwrite	overwrite existing directory

Examples

```
library(data.table)

## simple example:
x = as.disk.frame(data.table(start=c(5,31,22,16), end=c(8,50,25,18), val2 = 7:10))
y = as.disk.frame(data.table(start=c(10, 20, 30), end=c(15, 35, 45), val1 = 1:3))
byxy = c("val1", "start", "end")
xy.df = foverlaps.disk.frame(
  x, y, by.x = byxy, by.y = byxy,
  merge_by_chunk_id = TRUE, overwrite = TRUE)

# clean up
delete(x)
delete(y)
delete(xy.df)
```

---

gen_datatable_synthetic

Generate synthetic dataset for testing

Description

Generate synthetic dataset for testing

Usage

```
gen_datatable_synthetic(N = 2e+08, K = 100)
```

Arguments

N	number of rows. Defaults to 200 million
K	controls the number of unique values for id. Some ids will have K distinct values while others have N/K distinct values

get_chunk *Obtain one chunk by chunk id*

Description

Obtain one chunk by chunk id

Usage

```
get_chunk(...)

## S3 method for class 'disk.frame'
get_chunk(df, n, keep = NULL, full.names = FALSE, ...)
```

Arguments

...	passed to fst::read_fst or whichever read function is used in the backend
df	a disk.frame
n	the chunk id. If numeric then matches by number, if character then returns the chunk with the same name as n
keep	the columns to keep
full.names	whether n is the full path to the chunks or just a relative path file name. Ignored if n is numeric

Examples

```
cars.df = as.disk.frame(cars, nchunks = 2)
get_chunk(cars.df, 1)
get_chunk(cars.df, 2)
get_chunk(cars.df, 1, keep = "speed")

# if full.names = TRUE then the full path to the chunk need to be provided
get_chunk(cars.df, file.path(attr(cars.df, "path"), "1.fst"), full.names = TRUE)

# clean up cars.df
delete(cars.df)
```

get_chunk_ids *Get the chunk IDs and files names*

Description

Get the chunk IDs and files names

Usage

```
get_chunk_ids(df, ..., full.names = FALSE, strip_extension = TRUE)
```

Arguments

df	a disk.frame
...	passed to list.files
full.names	If TRUE returns the full path to the file, Defaults to FALSE
strip_extension	If TRUE then the file extension in the chunk_id is removed. Defaults to TRUE

Examples

```
cars.df = as.disk.frame(cars)

# return the integer-string chunk IDs
get_chunk_ids(cars.df)

# return the file name chunk IDs
get_chunk_ids(cars.df, full.names = TRUE)

# return the file name chunk IDs with file extension
get_chunk_ids(cars.df, strip_extension = FALSE)

# clean up cars.df
delete(cars.df)
```

---

groups.disk.frame *The shard keys of the disk.frame*

Description

The shard keys of the disk.frame

Usage

```
## S3 method for class 'disk.frame'
groups(x)
```

Arguments

x	a disk.frame

Value

character

hard_arrange *Perform a hard arrange*

Description

A hard_arrange is a sort by that also reorganizes the chunks to ensure that every unique grouping of ‘by“ is in the same chunk. Or in other words, every row that share the same ‘by‘ value will end up in the same chunk.

Usage

```
hard_arrange(df, ..., add = FALSE, .drop = FALSE)

## S3 method for class 'data.frame'
hard_arrange(df, ...)

## S3 method for class 'disk.frame'
hard_arrange(
  df,
  ...,
  outdir = tempfile("tmp_disk_frame_hard_arrange"),
  nchunks = disk.frame::nchunks(df),
  overwrite = TRUE
)
```

Arguments

df	a disk.frame
...	grouping variables
add	same as dplyr::arrange
.drop	same as dplyr::arrange
outdir	the output directory
nchunks	The number of chunks in the output. Defaults = nchunks.disk.frame(df)
overwrite	overwrite the out put directory

Examples

```
iris.df = as.disk.frame(iris, nchunks = 2)

# arrange iris.df by specifies and ensure rows with the same specifies are in the same chunk
iris_hard.df = hard_arrange(iris.df, Species)

get_chunk(iris_hard.df, 1)
get_chunk(iris_hard.df, 2)

# clean up cars.df
```

```
delete(iris.df)
delete(iris_hard.df)
```

---

hard_group_by *Perform a hard group*

Description

A hard_group_by is a group by that also reorganizes the chunks to ensure that every unique grouping of ‘by‘‘ is in the same chunk. Or in other words, every row that share the same ‘by‘ value will end up in the same chunk.

Usage

```
hard_group_by(df, ..., .add = FALSE, .drop = FALSE)

## S3 method for class 'data.frame'
hard_group_by(df, ..., .add = FALSE, .drop = FALSE)

## S3 method for class 'disk.frame'
hard_group_by(
  df,
  ...,
  outdir = tempfile("tmp_disk_frame_hard_group_by"),
  nchunks = disk.frame::nchunks(df),
  overwrite = TRUE,
  shardby_function = "hash",
  sort_splits = NULL,
  desc_vars = NULL,
  sort_split_sample_size = 100
)
```

Arguments

df	a disk.frame
...	grouping variables
.add	same as dplyr::group_by
.drop	same as dplyr::group_by
outdir	the output directory
nchunks	The number of chunks in the output. Defaults = nchunks.disk.frame(df)
overwrite	overwrite the out put directory
shardby_function	splitting of chunks: "hash" for hash function or "sort" for semi-sorted chunks
sort_splits	for the "sort" shardby function, a dataframe with the split values.
desc_vars	for the "sort" shardby function, the variables to sort descending.

sort_split_sample_size
: for the "sort" shardby function, if sort_splits is null, the number of rows to sample per chunk for random splits.

Examples

```
iris.df = as.disk.frame(iris, nchunks = 2)

# group_by iris.df by specifies and ensure rows with the same specifies are in the same chunk
iris_hard.df = hard_group_by(iris.df, Species)

get_chunk(iris_hard.df, 1)
get_chunk(iris_hard.df, 2)

# clean up cars.df
delete(iris.df)
delete(iris_hard.df)
```

---

head.disk.frame *Head and tail of the disk.frame*

Description

Head and tail of the disk.frame

Usage

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

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

Arguments

x	a disk.frame
n	number of rows to include
...	passed to base::head or base::tail

Examples

```
cars.df = as.disk.frame(cars)
head(cars.df)
tail(cars.df)

# clean up
delete(cars.df)
```

is_disk.frame *Checks if a folder is a disk.frame*

Description

Checks if a folder is a disk.frame

Usage

```
is_disk.frame(df)
```

Arguments

df	a disk.frame or directory to check

Examples

```
cars.df = as.disk.frame(cars)

is_disk.frame(cars) # FALSE
is_disk.frame(cars.df) # TRUE

# clean up cars.df
delete(cars.df)
```

make_glm_streaming_fn *A streaming function for speedglm*

Description

Define a function that can be used to feed data into speedglm and biglm

Usage

```
make_glm_streaming_fn(data, verbose = FALSE)
```

Arguments

data	a disk.frame
verbose	Whether to print the status of data loading. Default to FALSE

Value

return a function, fn, that can be used as the data argument in biglm::bigglm or speedglm::shglm

See Also

Other Machine Learning (ML): dfglm()

Examples

```
cars.df = as.disk.frame(cars)
streamacq = make_glm_streaming_fn(cars.df, verbose = FALSE)

majorv = as.integer(version$major)
minorv = as.integer(strsplit(version$minor, ".", fixed=TRUE)[[1]][1])
if(((majorv == 3) & (minorv >= 6)) | (majorv > 3)) {
  m = biglm::bigglm(dist ~ speed, data = streamacq)
  summary(m)
  predict(m, get_chunk(cars.df, 1))
  predict(m, collect(cars.df, 1))
} else {
  m = speedglm::shglm(dist ~ speed, data = streamacq)
}
```

---

merge.disk.frame *Merge function for disk.frames*

Description

Merge function for disk.frames

Usage

```
## S3 method for class 'disk.frame'
merge(
  x,
  y,
  by,
  outdir = tempfile(fileext = ".df"),
  ...,
  merge_by_chunk_id = FALSE,
  overwrite = FALSE
)
```

Arguments

x	a disk.frame
y	a disk.frame or data.frame
by	the merge by keys
outdir	The output directory for the disk.frame
...	passed to merge and cmap.disk.frame

merge_by_chunk_id
: if TRUE then only chunks in df1 and df2 with the same chunk id will get merged

overwrite
: overwrite the outdir or not

Examples

```
b = as.disk.frame(data.frame(a = 51:150, b = 1:100))
d = as.disk.frame(data.frame(a = 151:250, b = 1:100))
bd.df = merge(b, d, by = "b", merge_by_chunk_id = TRUE)

# clean up cars.df
delete(b)
delete(d)
delete(bd.df)
```

---

move_to *Move or copy a disk.frame to another location*

Description

Move or copy a disk.frame to another location

Usage

```
move_to(df, outdir, ..., copy = FALSE)

copy_df_to(df, outdir, ...)
```

Arguments

df	The disk.frame
outdir	The new location
...	NOT USED
copy	Merely copy and not move

Value

a disk.frame

Examples

```
cars.df = as.disk.frame(cars)

cars_copy.df = copy_df_to(cars.df, outdir = tempfile(fileext=".df"))

cars2.df = move_to(cars.df, outdir = tempfile(fileext=".df"))

# clean up
delete(cars_copy.df)
delete(cars2.df)
```

nchunks *Returns the number of chunks in a disk.frame*

Description

Returns the number of chunks in a disk.frame

Usage

```
nchunks(df, ...)

nchunk(df, ...)

## S3 method for class 'disk.frame'
nchunk(df, ...)

## S3 method for class 'disk.frame'
nchunks(df, skip.ready.check = FALSE, ...)
```

Arguments

df	a disk.frame
...	not used
skip.ready.check	NOT implemented

Examples

```
cars.df = as.disk.frame(cars)

# return the number of chunks
nchunks(cars.df)
nchunk(cars.df)

# clean up cars.df
delete(cars.df)
```

nrow *Number of rows or columns*

Description

Number of rows or columns

Usage

```
nrow(df, ...)

## S3 method for class 'disk.frame'
nrow(df, ...)

ncol(df)

## S3 method for class 'disk.frame'
ncol(df)
```

Arguments

df	a disk.frame
...	passed to base::nrow

Examples

```
cars.df = as.disk.frame(cars)

# return total number of column and rows
ncol(cars.df)
nrow(cars.df)

# clean up cars.df
delete(cars.df)
```

---

overwrite_check *Check if the outdir exists or not*

Description

If the overwrite is TRUE then the folder will be deleted, otherwise the folder will be created.

Usage

```
overwrite_check(outdir, overwrite)
```

Arguments

outdir	the output directory
overwrite	TRUE or FALSE if ‘outdir“ exists and overwrite = FALSE then throw an error

Examples

```
tf = tempfile()
overwrite_check(tf, overwrite = FALSE)
overwrite_check(tf, overwrite = TRUE)

# clean up
fs::dir_delete(tf)
```

`print.disk.frame` *Print disk.frame*

Description

a new print method for disk.frame

Usage

```
## S3 method for class 'disk.frame'
print(x, ...)
```

Arguments

`x`	disk.frame
`...`	not used

`pull.disk.frame` *Pull a column from table similar to 'dplyr::pull'.*

Description

Pull a column from table similar to 'dplyr::pull'.

Usage

```
## S3 method for class 'disk.frame'
pull(.data, var = -1, name = NULL, ...)
```

Arguments

`.data`	The disk.frame
`var`	can be an positive or negative integer or a character/string. See dplyr::pull documentation
`name`	See dplyr::pull documentation
`...`	Not used, kept for compatibility with 'dplyr::pull'

`rbindlist.disk.frame` *rbindlist disk.frames together*

Description

rbindlist disk.frames together

Usage

```
rbindlist.disk.frame(
  df_list,
  outdir = tempfile(fileext = ".df"),
  by_chunk_id = TRUE,
  parallel = TRUE,
  compress = 50,
  overwrite = TRUE,
  .progress = TRUE
)
```

Arguments

`df_list`	A list of disk.frames
`outdir`	Output directory of the row-bound disk.frames
`by_chunk_id`	If TRUE then only the chunks with the same chunk IDs will be bound
`parallel`	if TRUE then bind multiple disk.frame simultaneously, Defaults to TRUE
`compress`	0-100, 100 being the highest compression rate.
`overwrite`	overwrite the output directory
`.progress`	A logical, for whether or not to print a progress bar for multiprocess, multisession, and multicore plans. From furrr

Examples

```
cars.df = as.disk.frame(cars)

# row-bind two disk.frames
cars2.df = rbindlist.disk.frame(list(cars.df, cars.df))

# clean up cars.df
delete(cars.df)
delete(cars2.df)
```

rechunk *Increase or decrease the number of chunks in the disk.frame*

Description

Increase or decrease the number of chunks in the disk.frame

Usage

```
rechunk(
  df,
  nchunks,
  outdir = attr(df, "path", exact = TRUE),
  shardby = NULL,
  overwrite = TRUE,
  shardby_function = "hash",
  sort_splits = NULL,
  desc_vars = NULL
)
```

Arguments

df	the disk.frame to rechunk
nchunks	number of chunks
outdir	the output directory
shardby	the shardkeys
overwrite	overwrite the output directory
shardby_function	splitting of chunks: "hash" for hash function or "sort" for semi-sorted chunks
sort_splits	for the "sort" shardby function, a dataframe with the split values.
desc_vars	for the "sort" shardby function, the variables to sort descending.

Examples

```
# create a disk.frame with 2 chunks in tempdir()
cars.df = as.disk.frame(cars, nchunks = 2)

# re-chunking cars.df to 3 chunks, done "in-place" to the same folder as cars.df
rechunk(cars.df, 3)

new_path = tempfile(fileext = ".df")
# re-chunking cars.df to 4 chunks, shard by speed, and done "out-of-place" to a new directory
cars2.df = rechunk(cars.df, 4, outdir=new_path, shardby = "speed")

# clean up cars.df
delete(cars.df)
delete(cars2.df)
```

recommend_nchunks *Recommend number of chunks based on input size*

Description

Computes the recommended number of chunks to break a data.frame into. It can accept filesizes in bytes (as integer) or a data.frame

Usage

```
recommend_nchunks(
  df,
  type = "csv",
  minchunks = data.table::getDTthreads(),
  conservatism = 8,
  ram_size = df_ram_size()
)
```

Arguments

df	a disk.frame or the file size in bytes of a CSV file holding the data
type	only = "csv" is supported. It indicates the file type corresponding to file size 'df'
minchunks	the minimum number of chunks. Defaults to the number of CPU cores (without hyper-threading)
conservatism	a multiplier to the recommended number of chunks. The more chunks the smaller the chunk size and more likely that each chunk can fit into RAM
ram_size	The amount of RAM available which is usually computed. Except on RStudio with R3.6+

Examples

```
# recommend nchunks based on data.frame
recommend_nchunks(cars)

# recommend nchunks based on file size ONLY CSV is implemented at the moment
recommend_nchunks(1024^3)
```

remove_chunk *Removes a chunk from the disk.frame*

Description

Removes a chunk from the disk.frame

Usage

```
remove_chunk(df, chunk_id, full.names = FALSE)
```

Arguments

df	a disk.frame
chunk_id	the chunk ID of the chunk to remove. If it's a number then return number.fst
full.names	TRUE or FALSE. Defaults to FALSE. If true then chunk_id is the full path to the chunk otherwise it's the relative path

Examples

```
# TODO add these to tests
cars.df = as.disk.frame(cars, nchunks = 4)

# removes 3rd chunk
remove_chunk(cars.df, 3)
nchunks(cars.df) # 3

# removes 4th chunk
remove_chunk(cars.df, "4.fst")
nchunks(cars.df) # 3

# removes 2nd chunk
remove_chunk(cars.df, file.path(attr(cars.df, "path", exact=TRUE), "2.fst"), full.names = TRUE)
nchunks(cars.df) # 1

# clean up cars.df
delete(cars.df)
```

---

sample_frac.disk.frame

Sample n rows from a disk.frame

Description

Sample n rows from a disk.frame

Usage

```
## S3 method for class 'disk.frame'
sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = NULL, ...)
```

Arguments

tbl	A data.frame.
size	<tidy-select> For sample_n(), the number of rows to select. For sample_frac(), the fraction of rows to select. If tbl is grouped, size applies to each group.
replace	Sample with or without replacement?
weight	<tidy-select> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
.env	DEPRECATED.
...	ignored

Examples

```
cars.df = as.disk.frame(cars)

collect(sample_frac(cars.df, 0.5))

# clean up cars.df
delete(cars.df)
```

---

select.disk.frame *The dplyr verbs implemented for disk.frame*

Description

Please see the dplyr document for their usage. Please note ‘chunk_arrange‘ performs the actions within each chunk

Usage

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

## S3 method for class 'disk.frame'
rename(.data, ...)

## S3 method for class 'disk.frame'
filter(.data, ...)

## S3 method for class 'disk.frame'
mutate(.data, ...)

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

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

chunk_arrange(.data, ...)

tally.disk.frame(.data, ...)

count.disk.frame(.data, ...)

add_count.disk.frame(.data, ...)

add_tally.disk.frame(.data, ...)

## S3 method for class 'disk.frame'
do(.data, ...)

## S3 method for class 'disk.frame'
distinct(...)

chunk_distinct(.data, ...)

## S3 method for class 'disk.frame'
glimpse(.data, ...)
```

Arguments

.data	a disk.frame
...	Same as the dplyr functions

Examples

```
library(dplyr)
cars.df = as.disk.frame(cars)
mult = 2

# use all any of the supported dplyr
cars2 = cars.df %>%
  select(speed) %>%
  mutate(speed2 = speed * mult) %>%
  filter(speed < 50) %>%
  rename(speed1 = speed) %>%
  collect

# clean up cars.df
delete(cars.df)
```

`setup_disk.frame` *Set up disk.frame environment*

Description

Set up disk.frame environment

Usage

```
setup_disk.frame(
  workers = data.table::getDTthreads(),
  future_backend = future::multisession,
  ...,
  gui = FALSE
)
```

Arguments

`workers`	the number of workers (background R processes in the
`future_backend`	which future backend to use for parallelization
`...`	passed to 'future::plan'
`gui`	Whether to use a Graphical User Interface (GUI) for selecting the options. Defaults to FALSE

Examples

```
if (interactive()) {
  # setup disk.frame to use multiple workers these may use more than two
  # cores, and is therefore not allowed on CRAN. Hence it's set to run only in
  # interactive session
  setup_disk.frame()

  # use a Shiny GUI to adjust settings
  # only run in interactive()
  setup_disk.frame(gui = TRUE)
}

# set the number workers to 2
setup_disk.frame(2)

# if you do not wish to use multiple workers you can set it to sequential
setup_disk.frame(future_backend=future::sequential)
```

shard *Shard a data.frame/data.table or disk.frame into chunk and saves it into a disk.frame*

Description

Shard a data.frame/data.table or disk.frame into chunk and saves it into a disk.frame

'distribute' is an alias for 'shard'

Usage

```
shard(
  df,
  shardby,
  outdir = tempfile(fileext = ".df"),
  ...,
  nchunks = recommend_nchunks(df),
  overwrite = FALSE,
  shardby_function = "hash",
  sort_splits = NULL,
  desc_vars = NULL
)

distribute(...)
```

Arguments

df	A data.frame/data.table or disk.frame. If disk.frame, then rechunk(df, ...) is run
shardby	The column(s) to shard the data by.
outdir	The output directory of the disk.frame
...	not used
nchunks	The number of chunks
overwrite	If TRUE then the chunks are overwritten
shardby_function	splitting of chunks: "hash" for hash function or "sort" for semi-sorted chunks
sort_splits	If shardby_function is "sort", the split values for sharding
desc_vars	for the "sort" shardby function, the variables to sort descending.

Examples

```
# shard the cars data.frame by speed so that rows with the same speed are in the same chunk
iris.df = shard(iris, "Species")

# clean up cars.df
delete(iris.df)
```

shardkey *Returns the shardkey (not implemented yet)*

Description

Returns the shardkey (not implemented yet)

Usage

```
shardkey(df)
```

Arguments

df	a disk.frame

shardkey_equal *Compare two disk.frame shardkeys*

Description

Compare two disk.frame shardkeys

Usage

```
shardkey_equal(sk1, sk2)
```

Arguments

sk1	shardkey1
sk2	shardkey2

show_ceremony *Show the code to setup disk.frame*

Description

Show the code to setup disk.frame

Usage

```
show_ceremony()

ceremony_text()

show_boilerplate()

insert_ceremony()
```

srckeep *Keep only the variables from the input listed in selections*

Description

Keep only the variables from the input listed in selections

Usage

```
srckeep(diskf, selections, ...)

srckeepchunks(diskf, chunks, ...)
```

Arguments

diskf	a disk.frame
selections	The list of variables to keep from the input source
...	not yet used
chunks	The chunks to load

Examples

```
cars.df = as.disk.frame(cars)

# when loading cars's chunks into RAM, load only the column speed
collect(srckeep(cars.df, "speed"))

# clean up cars.df
delete(cars.df)
```

summarise.grouped_disk.frame *A function to parse the summarize function*

Description

The disk.frame group by operation perform group WITHIN each chunk. This is often used for performance reasons. If the user wishes to perform group-by, they may choose to use the ‘hard_group_by‘ function which is expensive as it reorganizes the chunks by the shard key.

Usage

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

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

## S3 method for class 'disk.frame'
group_by(.data, ..., add = FALSE, .drop = dplyr::group_by_drop_default(.data))

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

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

Arguments

.data	a disk.frame
...	same as the dplyr::group_by
add	from dplyr
.drop	from dplyr

See Also

hard_group_by

---

tbl_vars.disk.frame *Column names for RStudio auto-complete*

Description

Returns the names of the columns. Needed for RStudio to complete variable names

Usage

```
## S3 method for class 'disk.frame'
tbl_vars(x)

## S3 method for class 'disk.frame'
group_vars(x)
```

Arguments

x	a disk.frame

var_df.chunk_agg.disk.frame
One Stage function

Description

One Stage function

mean chunk_agg

mean collected_agg

Usage

```
var_df.chunk_agg.disk.frame(x, na.rm = FALSE)

var_df.collected_agg.disk.frame(listx)

sd_df.chunk_agg.disk.frame(x, na.rm = FALSE)

sd_df.collected_agg.disk.frame(listx)

mean_df.chunk_agg.disk.frame(x, na.rm = FALSE, ...)

mean_df.collected_agg.disk.frame(listx)

sum_df.chunk_agg.disk.frame(x, ...)

sum_df.collected_agg.disk.frame(listx, ...)

min_df.chunk_agg.disk.frame(x, ...)

min_df.collected_agg.disk.frame(listx, ...)

max_df.chunk_agg.disk.frame(x, ...)

max_df.collected_agg.disk.frame(listx, ...)

median_df.chunk_agg.disk.frame(x, ...)

median_df.collected_agg.disk.frame(listx, ...)

n_df.chunk_agg.disk.frame(...)

n_df.collected_agg.disk.frame(listx, ...)

length_df.chunk_agg.disk.frame(x, ...)
```

```
length_df.collected_agg.disk.frame(listx, ...)

any_df.chunk_agg.disk.frame(x, ...)

any_df.collected_agg.disk.frame(listx, ...)

all_df.chunk_agg.disk.frame(x, ...)

all_df.collected_agg.disk.frame(listx, ...)

n_distinct_df.chunk_agg.disk.frame(x, na.rm = FALSE, ...)

n_distinct_df.collected_agg.disk.frame(listx, ...)

quantile_df.chunk_agg.disk.frame(x, ...)

quantile_df.collected_agg.disk.frame(listx, ...)

IQR_df.chunk_agg.disk.frame(x, na.rm = FALSE, ...)

IQR_df.collected_agg.disk.frame(listx, ...)
```

Arguments

x	the input
na.rm	Remove NAs. TRUE of FALSE
listx	a list
...	additional options

`write_disk.frame` *Write disk.frame to disk*

Description

Write a data.frame/disk.frame to a disk.frame location. If df is a data.frame then using the as.disk.frame function is recommended for most cases

Usage

```
write_disk.frame(
  df,
  outdir = tempfile(fileext = ".df"),
  nchunks = ifelse("disk.frame" %in% class(df), nchunks.disk.frame(df),
    recommend_nchunks(df)),
  overwrite = FALSE,
  shardby = NULL,
```

```
  compress = 50,
  shardby_function = "hash",
  sort_splits = NULL,
  desc_vars = NULL,
  ...
)

output_disk.frame(...)
```

Arguments

df	a disk.frame
outdir	output directory for the disk.frame
nchunks	number of chunks
overwrite	overwrite output directory
shardby	the columns to shard by
compress	compression ratio for fst files
shardby_function	splitting of chunks: "hash" for hash function or "sort" for semi-sorted chunks
sort_splits	for the "sort" shardby function, a dataframe with the split values.
desc_vars	for the "sort" shardby function, the variables to sort descending.
...	passed to cmap.disk.frame

Examples

```
cars.df = as.disk.frame(cars)

# write out a lazy disk.frame to disk
cars2.df = write_disk.frame(cmap(cars.df, ~.x[1,]), overwrite = TRUE)
collect(cars2.df)

# clean up cars.df
delete(cars.df)
delete(cars2.df)
```

---

zip_to_disk.frame *'zip_to_disk.frame' is used to read and convert every CSV file within the zip file to disk.frame format*

Description

'zip_to_disk.frame' is used to read and convert every CSV file within the zip file to disk.frame format

Usage

```
zip_to_disk.frame(
  zipfile,
  outdir,
  ...,
  validation.check = FALSE,
  overwrite = TRUE
)
```

Arguments

zipfile	The zipfile
outdir	The output directory for disk.frame
...	passed to fread
validation.check	should the function perform a check at the end to check for validity of output. It can detect issues with conversion
overwrite	overwrite output directory

Value

a list of disk.frame

See Also

Other ingesting data: csv_to_disk.frame()

Examples

```
# create a zip file containing a csv
csvfile = tempfile(fileext = ".csv")
write.csv(cars, csvfile)
zipfile = tempfile(fileext = ".zip")
zip(zipfile, csvfile)

# read every file and convert it to a disk.frame
zip.df = zip_to_disk.frame(zipfile, tempfile(fileext = ".df"))

# there is only one csv file so it return a list of one disk.frame
zip.df[[1]]

# clean up
unlink(csvfile)
unlink(zipfile)
delete(zip.df[[1]])
```

`[.disk.frame` *[interface for disk.frame using fst backend*

Description

[interface for disk.frame using fst backend

Usage

```
## S3 method for class 'disk.frame'

  df[
  ...,
  keep = NULL,
  rbind = TRUE,
  use.names = TRUE,
  fill = FALSE,
  idcol = NULL
]
```

Arguments

df	a disk.frame
...	same as data.table
keep	the columns to srckeep
rbind	Whether to rbind the chunks. Defaults to TRUE
use.names	Same as in data.table::rbindlist
fill	Same as in data.table::rbindlist
idcol	Same as in data.table::rbindlist

Examples

```
cars.df = as.disk.frame(cars)
speed_limit = 50
cars.df[speed < speed_limit ,.N, cut(dist, pretty(dist))]

# clean up
delete(cars.df)
```

Index