Package ‘keyholder’

March 11, 2023

Title  Store Data About Rows
Version  0.1.7
Description  Tools for keeping track of information, named
``keys'', about rows of data frame like objects. This is done by
creating special attribute ``keys'' which is updated after every change
in rows (subsetting, ordering, etc.). This package is designed to
work tightly with ‘dplyr’ package.
License  MIT + file LICENSE
https://github.com/echasnovski/keyholder/
BugReports  https://github.com/echasnovski/keyholder/issues/
Depends  R (>= 3.4.0)
Imports  dplyr (>= 0.7.0), rlang (>= 0.1), tibble, utils
Suggests  covr, knitr, rmarkdown, testthat
VignetteBuilder  knitr
Encoding  UTF-8
RoxygenNote  7.2.3
NeedsCompilation  no
Author  Evgeni Chasnovski [aut, cre]
<https://orcid.org/0000-0002-1617-4019>
Maintainer  Evgeni Chasnovski <evgeni.chasnovski@gmail.com>
Repository  CRAN
Date/Publication  2023-03-11 18:00:02 UTC

R topics documented:

keyholder-package .................................................. 2
key-by-scoped ...................................................... 3
keyed-df ............................................................. 4
keyed-df-one-tbl ................................................... 5
keyholder-package

keyed-df-two-tbl ........................................ 6
keyholder-id .................................................. 7
keyholder-scoped ............................................. 8
keyholder-supported-funs ................................. 9
keys-get ....................................................... 9
keys-manipulate ............................................. 10
keys-set ....................................................... 11
remove-keys-scoped ......................................... 13
rename-keys-scoped ......................................... 14
restore-keys-scoped ......................................... 14

Index 16

keyholder-package  keyholder: Store Data About Rows

Description

keyholder offers a set of tools for storing information about rows of data frame like objects. The common use cases are:

- Track rows of data frame without changing it.
- Store columns for future restoring in data frame.
- Hide columns for convenient use of dplyr’s *_if scoped variants of verbs.

Details

To learn more about keyholder:

- Browse vignettes with browseVignettes(package = "keyholder").
- Look how to set keys.
- Look at the list of supported functions.

Author(s)

Maintainer: Evgeni Chasnovski <evgeni.chasnovski@gmail.com> (ORCID)

See Also

Useful links:

- https://echasnovski.github.io/keyholder/
- https://github.com/echasnovski/keyholder/
- Report bugs at https://github.com/echasnovski/keyholder/issues/
### Description

These functions perform keying by selection of variables using corresponding **scoped variant** of `select`. Appropriate data frame is selected with scoped function first, and then it is assigned as keys.

### Usage

- `key_by_all(.tbl, .funs = list(), ..., .add = FALSE, .exclude = FALSE)`
- `key_by_if(.tbl, .predicate, .funs = list(), ..., .add = FALSE, .exclude = FALSE)`
- `key_by_at(.tbl, .vars, .funs = list(), ..., .add = FALSE, .exclude = FALSE)`

### Arguments

- `.tbl` Reference data frame.
- `.funs` Parameter for **scoped** functions.
- `...` Parameter for **scoped** functions.
- `.add` Whether to add keys to (possibly) existing ones. If **FALSE** keys will be overridden.
- `.exclude` Whether to exclude key variables from `.tbl`.
- `.predicate` Parameter for **scoped** functions.
- `.vars` Parameter for **scoped** functions.

### See Also

- *Not scoped key_by()*

### Examples

```r
mtcars %>% key_by_all(.funs = toupper)
mtcars %>% key_by_if(rlang::is_integerish, toupper)
mtcars %>% key_by_at(c("vs", "am"), toupper)
```
## keyed-df

**Keyed object**

### Description

Utility functions for keyed objects which are implemented with class `keyed_df`. Keyed object should be a data frame which inherits from `keyed_df` and contains a data frame of keys in attribute 'keys'.

### Usage

```r
is_keyed_df(.tbl)

is.keyed_df(.tbl)

## S3 method for class 'keyed_df'
print(x, ...)

## S3 method for class 'keyed_df'
x[i, j, ...]
```

### Arguments

- `.tbl` Object to check.
- `x` Object to print or extract elements.
- `...` Further arguments passed to or from other methods.
- `i, j` Arguments for `[`.

### Examples

```r
is_keyed_df(mtcars)

mtcars %>% key_by(vs) %>% is_keyed_df

# Not valid keyed_df
df <- mtcars
class(df) <- c("keyed_df", "data.frame")
is_keyed_df(df)
```
One-table verbs from dplyr for keyed_df

Description

Defined methods for dplyr generic single table functions. Most of them preserve 'keyed_df' class and 'keys' attribute (excluding summarise with scoped variants, distinct and do which remove them). Also these methods modify rows in keys according to the rows modification in reference data frame (if any).

Usage

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

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

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

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

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

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

## S3 method for class 'keyed_df'
ungroup(x, ...)

## S3 method for class 'keyed_df'
rowwise(data, ...)

## S3 method for class 'keyed_df'
distinct(.data, ..., .keep_all = FALSE)

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

## S3 method for class 'keyed_df'
arrange(.data, ..., .by_group = FALSE)

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

### Arguments

- `.data`, `data`, `x`: A keyed object.
- `...`: Appropriate arguments for functions.
- `.keep_all`: Parameter for `dplyr::distinct`.
- `.by_group`: Parameter for `dplyr::arrange`.

### Details

- `dplyr::transmute()` is supported implicitly with `dplyr::mutate()` support.
- `dplyr::rowwise()` is not supposed to be generic in `dplyr`. Use `rowwise.keyed_df` directly.

All scoped variants of present functions are also supported.

### See Also

- Two-table verbs

### Examples

```r
mtcars %>% key_by(vs, am) %>% dplyr::mutate(gear = 1)
```

---

**Description**

Defined methods for `dplyr` generic `join` functions. All of them preserve 'keyed_df' class and 'keys' attribute of the first argument. Also these methods modify rows in keys according to the rows modification in first argument (if any).

### Usage

```r
## S3 method for class 'keyed_df'
inner_join(x, y, by = NULL, copy = FALSE,  
suffix = c(".x", ",.y"), ...)  
```
right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)  
  ## S3 method for class 'keyed_df'
full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...)  
  ## S3 method for class 'keyed_df'
semi_join(x, y, by = NULL, copy = FALSE, ...)  
  ## S3 method for class 'keyed_df'
anti_join(x, y, by = NULL, copy = FALSE, ...)

Arguments

x, y, by, copy, suffix, ...
  Parameters for join functions.

See Also

One-table verbs

Examples

```r
dplyr::band_members %>% key_by(band) %>%
  dplyr::semi_join(dplyr::band_instruments, by = "name") %>%
  keys()
```

---

keyholder-id  Add id column and key

Description

Functions for creating id column and key.

Usage

use_id(.tbl)

compute_id_name(x)

add_id(.tbl)

key_by_id(.tbl, .add = FALSE, .exclude = FALSE)
keyholder-scoped

Arguments

- `.tbl` Reference data frame.
- `x` Character vector of names.
- `.add, .exclude` Parameters for `key_by()`.  

Details

- `use_id()` assigns as keys a tibble with column `.id` and row numbers of `.tbl` as values.
- `compute_id_name()` computes the name which is different from every element in `x` by the following algorithm: if `.id` is not present in `x` it is returned; if taken - `.id1` is checked; if taken - `.id11` is checked and so on.
- `add_id()` creates a column with unique name (computed with `compute_id_name()`) and row numbers as values (grouping is ignored). After that puts it as first column.
- `key_by_id()` is similar to `add_id()`: it creates a column with unique name and row numbers as values (grouping is ignored) and calls `key_by()` function to use this column as key. If `.add` is FALSE unique name is computed based on `.tbl` column names; if TRUE then based on `.tbl` and its keys column names.

Examples

```r
  mtcars %>% use_id()
  mtcars %>% add_id()
  mtcars %>% key_by_id(.exclude = TRUE)
```

keyholder-scoped Operate on a selection of keys

Description

*keyholder* offers *scoped* variants of the following functions:

- `key_by()`. See `key_by_all()`.
- `remove_keys()`. See `remove_keys_all()`.
- `restore_keys()`. See `restore_keys_all()`.
- `rename_keys()`. See `rename_keys_all()`.

Arguments

- `.funs` Parameter for *scoped* functions.
- `.vars` Parameter for *scoped* functions.
- `.predicate` Parameter for *scoped* functions.
- `...` Parameter for *scoped* functions.
keyholder-supported-funs

See Also

- Not scoped manipulation functions
- Not scoped key_by()

keyholder-supported-funs

Supported functions

Description

keyholder supports the following functions:

- Base subsetting with `[`.
- `dplyr` one table verbs.
- `dplyr` two table verbs.

keys-get

Get keys

Description

Functions for getting information about keys.

Usage

`keys(.tbl)`

`raw_keys(.tbl)`

`has_keys(.tbl)`

Arguments

`.tbl` Reference data frame.

Value

`keys()` always returns a tibble of keys. In case of no keys it returns a tibble with number of rows as in `.tbl` and zero columns. `raw_keys()` is just a wrapper for `attr(.tbl, "keys")`. To know whether `.tbl` has keys use `has_keys()`.

See Also

- Set keys, Manipulate keys
### Examples

```r
keys(mtcars)
raw_keys(mtcars)
has_keys(mtcars)
df <- key_by(mtcars, vs, am)
keys(df)
has_keys(df)
```

### Description

Functions to manipulate keys.

### Usage

- `remove_keys(.tbl, ..., .unkey = FALSE)`
- `restore_keys(.tbl, ..., .remove = FALSE, .unkey = FALSE)`
- `pull_key(.tbl, var)`
- `rename_keys(.tbl, ...)`

### Arguments

- `.tbl` Reference data frame.
- `...` Variables to be used for operations defined in similar fashion as in `dplyr::select()`.
- `.unkey` Whether to `unkey` .tbl in case there are no keys left.
- `.remove` Whether to remove keys after restoring.
- `var` Parameter for `dplyr::pull()`.

### Details

- `remove_keys()` removes keys defined with `...`.
- `restore_keys()` transfers keys defined with `...` into `.tbl` and removes them from `keys` if `.remove == TRUE`. If `.tbl` is grouped the following happens:
  - If restored keys don’t contain grouping variables then groups don’t change;
• If restored keys contain grouping variables then result will be regrouped based on restored values. In other words restoring keys beats 'not-modifying' grouping variables rule. It is made according to the ideology of keys: they contain information about rows and by restoring you want it to be available.

pull_key() extracts one specified column from keys with dplyr::pull().
rename_keys() renames columns in keys using dplyr::rename().

See Also
Get keys, Set keys
Scoped functions

Examples

```r
df <- mtcars %>% dplyr::as_tibble() %>%
    key_by(vs, am, .exclude = TRUE)
df %>% remove_keys(vs)

df %>% remove_keys(dplyr::everything())

df %>% remove_keys(dplyr::everything(), .unkey = TRUE)

df %>% restore_keys(vs)

df %>% restore_keys(vs, .remove = TRUE)

df %>% restore_keys(dplyr::everything(), .remove = TRUE)

df %>% restore_keys(dplyr::everything(), .remove = TRUE, .unkey = TRUE)

# Restoring on grouped data frame
df_grouped <- df %>% dplyr::mutate(vs = 1) %>% dplyr::group_by(vs)
df_grouped %>% restore_keys(dplyr::everything())

# Pulling
df %>% pull_key(vs)

# Renaming
df %>% rename_keys(Vs = vs)
```

| keys-set | Set keys |
Description

Key is a vector which goal is to provide information about rows in reference data frame. Its length should always be equal to number of rows in data frame. Keys are stored as tibble in attribute "keys" and so one data frame can have multiple keys. Data frame with keys is implemented as class keyed_df.

Usage

```
keys(.tbl) <- value
assign_keys(.tbl, value)
key_by(.tbl, ..., .add = FALSE, .exclude = FALSE)
unkey(.tbl)
```

Arguments

- `.tbl` Reference data frame.
- `value` Values of keys (converted to tibble).
- `...` Variables to be used as keys defined in similar fashion as in `dplyr::select()`.
- `.add` Whether to add keys to (possibly) existing ones. If FALSE keys will be overridden.
- `.exclude` Whether to exclude key variables from `.tbl`.

Details

key_by ignores grouping when creating keys. Also if .add == TRUE and names of some added keys match the names of existing keys the new ones will override the old ones.

Value for keys<- should not be NULL because it is converted to tibble with zero rows. To remove keys use `unkey()`, `remove_keys()` or `restore_keys()`. `assign_keys` is a more suitable for piping wrapper for keys<-.

See Also

- Get keys, Manipulate keys
- Scoped key_by()

Examples

```
df <- dplyr::as_tibble(mtcars)

# Value is converted to tibble
keys(df) <- 1:nrow(df)

# This will throw an error
## Not run:
keys(df) <- 1:10
```
## End(Not run)

# Use 'vs' and 'am' as keys
\[
\text{df} \%\% \text{key_by}(\text{vs, am})
\]
\[
\text{df} \%\% \text{key_by}(\text{vs, am, .exclude = TRUE})
\]
\[
\text{df} \%\% \text{key_by}(\text{vs}) \%\% \text{key_by}(\text{am, .add = TRUE, .exclude = TRUE})
\]

# Override keys
\[
\text{df} \%\% \text{key_by}(\text{vs, am}) \%\% \text{dplyr::mutate}(\text{vs = 1}) \%\%
\text{key_by}(\text{gear, vs, .add = TRUE})
\]

# Use select helpers
\[
\text{df} \%\% \text{key_by}(\text{dplyr::one_of(c("vs", "am")))}
\]
\[
\text{df} \%\% \text{key_by}(\text{dplyr::everything()})
\]

---

### remove-keys-scoped

Remove selection of keys

---

**Description**

These functions remove selection of keys using corresponding scoped variant of `select`. `.funs` argument is removed because of its redundancy.

**Usage**

- `remove_keys_all(.tbl, ..., .unkey = FALSE)`
- `remove_keys_if(.tbl, .predicate, ..., .unkey = FALSE)`
- `remove_keys_at(.tbl, .vars, ..., .unkey = FALSE)`

**Arguments**

- `.tbl`: Reference data frame.
- `...`: Parameter for scoped functions.
- `.unkey`: Whether to `unkey()` .tbl in case there are no keys left.
- `.predicate`: Parameter for scoped functions.
- `.vars`: Parameter for scoped functions.
Examples

df <- mtcars %>% dplyr::as_tibble() %>% key_by(vs, am, disp)
df %>% remove_keys_all()

df %>% remove_keys_all(.unkey = TRUE)

df %>% remove_keys_if(rlang::is_integerish)

df %>% remove_keys_at(c("vs", "am"))

Description

These functions rename selection of keys using corresponding scoped variant of rename.

Usage

rename_keys_all(.tbl, .funs = list(), ...)
rename_keys_if(.tbl, .predicate, .funs = list(), ...)
rename_keys_at(.tbl, .vars, .funs = list(), ...)

Arguments

.tbl Reference data frame.
.funs Parameter for scoped functions.
... Parameter for scoped functions.
.predicate Parameter for scoped functions.
.vars Parameter for scoped functions.

Description

These functions restore selection of keys using corresponding scoped variant of select. .funs argument can be used to rename some keys (without touching actual keys) before restoring.
Usage

restore_keys_all(.tbl, .funs = list(), ..., .remove = FALSE, .unkey = FALSE)

restore_keys_if(.tbl, .predicate, .funs = list(), ..., .remove = FALSE, .unkey = FALSE)

restore_keys_at(.tbl, .vars, .funs = list(), ..., .remove = FALSE, .unkey = FALSE)

Arguments

.tbl Reference data frame.
.funs Parameter for scoped functions.
... Parameter for scoped functions.
.remove Whether to remove keys after restoring.
.unkey Whether to unkey(.tbl) in case there are no keys left.
.predicate Parameter for scoped functions.
.vars Parameter for scoped functions.

Examples

df <- mtcars %>% dplyr::as_tibble() %>% key_by(vs, am, disp)
# Just restore all keys
df %>% restore_keys_all()

# Restore all keys with renaming and without touching actual keys
df %>% restore_keys_all(.funs = toupper)

# Restore with renaming and removing
df %>%
    restore_keys_all(.funs = toupper, .remove = TRUE)

# Restore with renaming, removing and unkeying
df %>%
    restore_keys_all(.funs = toupper, .remove = TRUE, .unkey = TRUE)

# Restore with renaming keys satisfying the predicate
df %>%
    restore_keys_if(rlang::is_integerish, .funs = toupper)

# Restore with renaming specified keys
df %>%
    restore_keys_at(c("vs", "disp"), .funs = toupper)
Index

[4, 9
[4] keyed_df (keyed-df), 4
add_id (keyholder-id), 7
anti_join.keyed_df (keyed-df-two-tbl), 6
arrange.keyed_df (keyed-df-one-tbl), 5
assign_keys (keys-set), 11
assigns, 8
compute_id_name (keyholder-id), 7
distinct.keyed_df (keyed-df-one-tbl), 5
do.keyed_df (keyed-df-one-tbl), 5
dplyr, 2, 5, 6
dplyr::arrange, 6
dplyr::distinct, 6
dplyr::mutate(), 6
dplyr::pull(), 10, 11
dplyr::rename(), 11
dplyr::rowwise(), 6
dplyr::select(), 10, 12
dplyr::transmute(), 6
filter.keyed_df (keyed-df-one-tbl), 5
full_join.keyed_df (keyed-df-two-tbl), 6
Get keys, 11, 12
group_by.keyed_df (keyed-df-one-tbl), 5
has_keys (keys-get), 9
inner_join.keyed_df (keyed-df-two-tbl), 6
is.keyed_df (keyed-df), 4
is_keyed_df (keyed-df), 4
join, 6, 7
key-by-scoped, 3
key_by (keys-set), 11
key_by(), 8
key_by_all (key-by-scoped), 3
key_by_all(), 8
key_by_at (key-by-scoped), 3
key_by_id (keyholder-id), 7
key_by_if (key-by-scoped), 3
keyed-df, 4
keyed-df-one-tbl, 5
keyed-df-two-tbl, 6
keyed_df, 12
keyholder, 8
keyholder (keyholder-package), 2
keyholder-id, 7
keyholder-package, 2
keyholder-scoped, 8
keyholder-supported-funs, 9
keys, 4, 10
keys (keys-get), 9
keys-get, 9
keys-manipulate, 10
keys-set, 11
keys<- (keys-set), 11
left_join.keyed_df (keyed-df-two-tbl), 6
Manipulate keys, 9, 12
mutate.keyed_df (keyed-df-one-tbl), 5
Not scoped key_by(), 3, 9
Not scoped manipulation functions, 9
one table verbs, 9
One-table verbs, 7
print.keyed_df (keyed-df), 4
pull_key (keys-manipulate), 10
raw_keys (keys-get), 9
remove-keys-scoped, 13
remove_keys (keys-manipulate), 10
remove_keys(), 8, 12
remove_keys_all (remove-keys-scoped), 13
remove_keys_all(), 8
remove_keys_at (remove-keys-scoped), 13
remove_keys_if (remove-keys-scoped), 13
rename, 14
rename-keys-scoped, 14
rename.keyed_df (keyed-df-one-tbl), 5
rename_keys (keys-manipulate), 10
rename_keys(), 8
rename_keys_all (rename-keys-scoped), 14
rename_keys_all(), 8
rename_keys_at (rename-keys-scoped), 14
rename_keys_if (rename-keys-scoped), 14
restore-keys-scoped, 14
restore_keys (keys-manipulate), 10
restore_keys(), 8, 12
restore_keys_all (restore-keys-scoped),
14
restore_keys_all(), 8
restore_keys_at (restore-keys-scoped),
14
restore_keys_if (restore-keys-scoped),
14
right_join.keyed_df (keyed-df-two-tbl),
6
rowwise.keyed_df (keyed-df-one-tbl), 5
scoped, 3, 6, 8, 13–15
Scoped functions, 11
Scoped key_by(), 12
scoped variant, 3, 13, 14
select, 3, 13, 14
select.keyed_df (keyed-df-one-tbl), 5
semi_join.keyed_df (keyed-df-two-tbl), 6
Set keys, 9, 11
set keys, 2
slice.keyed_df (keyed-df-one-tbl), 5
summarise.keyed_df (keyed-df-one-tbl), 5
supported functions, 2
tibble, 9, 12
transmute.keyed_df (keyed-df-one-tbl), 5
two table verbs, 9
Two-table verbs, 6
ungroup.keyed_df (keyed-df-one-tbl), 5
unkey (keys-set), 11
unkey(), 10, 13, 15
use_id (keyholder-id), 7