Package 'cdata'

October 8, 2018

Type Package

Title Fluid Data Transformations

Version 1.0.2

Date 2018-10-08

URL https://github.com/WinVector/cdata/,
https://winvector.github.io/cdata/

Maintainer John Mount <jmount@win-vector.com>

BugReports https://github.com/WinVector/cdata/issues

Description Supplies higher-order fluid data transform operators that include pivot and anti-pivot as special cases.
The methodology is describe in 'Zumel', 2018, ``Fluid data reshaping with 'cdata'", <http://winvector.github.io/FluidData/FluidDataReshapingWithCdata.html> , doi:10.5281/zenodo.117329
Based on the 'DBI' database interface.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

Depends R (>= 3.4.0)

Imports wrapr (>= 1.6.3), rquery (>= 1.1.0), stats,

Suggests DBI, RSQLite, testthat, knitr, rmarkdown, data.table (>= 1.11.4), rqdatatable (>= 1.1.1)

VignetteBuilder knitr

ByteCompile true

NeedsCompilation no

Author John Mount [aut, cre],
Nina Zumel [aut],
Win-Vector LLC [cph]

Repository CRAN

Date/Publication 2018-10-08 16:10:03 UTC

R topics documented:

`blocks_to_rowrecs` *Map sets rows to columns (takes a* `data.frame`*).*

Description

Transform data facts from rows into additional columns using controlTable.

Usage

```
blocks_to_rowrecs(tallTable, keyColumns, controlTable, ...,
  columnsToCopy = NULL, checkNames = TRUE, checkKeys = TRUE,
  strict = FALSE, use_data_table = FALSE)

## Default S3 method:
blocks_to_rowrecs(tallTable, keyColumns, controlTable,
  ..., columnsToCopy = NULL, checkNames = TRUE, checkKeys = TRUE,
  strict = FALSE, use_data_table = TRUE)

## S3 method for class 'relop'
blocks_to_rowrecs(tallTable, keyColumns, controlTable, ...,
  columnsToCopy = NULL, checkNames = TRUE, strict = FALSE,
  use_data_table = TRUE,
  tmp_name_source = wrapr::mk_tmp_name_source("bltrr"),
  temporary = TRUE)
```

Arguments

tallTable	data.frame containing data to be mapped (in-memory data.frame).
keyColumns	character vector of column defining row groups
controlTable	table specifying mapping (local data frame)
...	force later arguments to be by name.
columnsToCopy	character, extra columns to copy.
checkNames	logical, if TRUE check names.
checkKeys	logical, if TRUE check keyColumns uniquely identify blocks (required).
strict	logical, if TRUE check control table name forms
use_data_table	logical if TRUE try to use data.table for the pivots.
tmp_name_source	a tempNameGenerator from cdata::mk_tmp_name_source()
temporary	logical, if TRUE make result temporary.

Details

This is using the theory of "fluid data"n (https://github.com/WinVector/cdata), which includes the principle that each data cell has coordinates independent of the storage details and storage detail dependent coordinates (usually row-id, column-id, and group-id) can be re-derived at will (the other principle is that there may not be "one true preferred data shape" and many re-shapings of data may be needed to match data to different algorithms and methods).

The controlTable defines the names of each data element in the two notations: the notation of the tall table (which is row oriented) and the notation of the wide table (which is column oriented). controlTable[, 1] (the group label) cross colnames(controlTable) (the column labels) are names of data cells in the long form. controlTable[, 2:ncol(controlTable)] (column labels) are names of data cells in the wide form. To get behavior similar to tidyr::gather/spread one builds the control table by running an appropriate query over the data.

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html and here https://github.com/WinVector/cdata.

Value

wide table built by mapping key-grouped tallTable rows to one row per group

See Also

build_pivot_control, rowrecs_to_blocks_q

Examples

```
# pivot example
d <- data.frame(meas = c('AUC', 'R2'),
                val = c(0.6, 0.2))

cT <- build_pivot_control(d,
```

```
                            columnToTakeKeysFrom= 'meas',
                            columnToTakeValuesFrom= 'val')
  blocks_to_rowrecs(d,
                    keyColumns = NULL,
                    controlTable = cT)


d <- data.frame(meas = c('AUC', 'R2'),
                val = c(0.6, 0.2))
cT <- build_pivot_control(
  d,
  columnToTakeKeysFrom= 'meas',
  columnToTakeValuesFrom= 'val')

ops <- rquery::local_td(d) %.>%
  blocks_to_rowrecs(.,
                    keyColumns = NULL,
                    controlTable = cT)
cat(format(ops))

if(requireNamespace("rqdatatable", quietly = TRUE)) {
  library("rqdatatable")
  d %.>%
    ops %.>%
    print(.)
}

if(requireNamespace("RSQLite", quietly = TRUE)) {
  db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  DBI::dbWriteTable(db,
                    'd',
                    d,
                    overwrite = TRUE,
                    temporary = TRUE)
  db %.>%
    ops %.>%
    print(.)
  DBI::dbDisconnect(db)
}
```

`blocks_to_rowrecs_q` *Map sets rows to columns (query based, take name of table).*

Description

Transform data facts from rows into additional columns using SQL and controlTable.

Usage

```
blocks_to_rowrecs_q(tallTable, keyColumns, controlTable, my_db, ...,
  columnsToCopy = NULL,
  tempNameGenerator = mk_tmp_name_source("mvtcq"), strict = FALSE,
  checkNames = TRUE, showQuery = FALSE, defaultValue = NULL,
  dropDups = FALSE, temporary = FALSE, resultName = NULL)
```

Arguments

tallTable	name of table containing data to be mapped (db/Spark data)
keyColumns	character list of column defining row groups
controlTable	table specifying mapping (local data frame)
my_db	db handle
...	force later arguments to be by name.
columnsToCopy	character list of column names to copy
tempNameGenerator	a tempNameGenerator from cdata::mk_tmp_name_source()
strict	logical, if TRUE check control table name forms
checkNames	logical, if TRUE check names
showQuery	if TRUE print query
defaultValue	if not NULL literal to use for non-match values.
dropDups	logical if TRUE suppress duplicate columns (duplicate determined by name, not content).
temporary	logical, if TRUE make result temporary.
resultName	character, name for result table.

Details

This is using the theory of "fluid data"n (https://github.com/WinVector/cdata), which includes the principle that each data cell has coordinates independent of the storage details and storage detail dependent coordinates (usually row-id, column-id, and group-id) can be re-derived at will (the other principle is that there may not be "one true preferred data shape" and many re-shapings of data may be needed to match data to different algorithms and methods).

The controlTable defines the names of each data element in the two notations: the notation of the tall table (which is row oriented) and the notation of the wide table (which is column oriented). controlTable[, 1] (the group label) cross colnames(controlTable) (the column labels) are names of data cells in the long form. controlTable[, 2:ncol(controlTable)] (column labels) are names of data cells in the wide form. To get behavior similar to tidyr::gather/spread one builds the control table by running an appropriate query over the data.

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html and here https://github.com/WinVector/cdata.

Value

wide table built by mapping key-grouped tallTable rows to one row per group

See Also

rowrecs_to_blocks_q, build_pivot_control_q, blocks_to_rowrecs

Examples

```
if (requireNamespace("DBI", quietly = TRUE) &&
  requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  # pivot example
  d <- data.frame(meas = c('AUC', 'R2'), val = c(0.6, 0.2))
  DBI::dbWriteTable(my_db,
                    'd',
                    d,
                    temporary = TRUE)
  cT <- build_pivot_control_q('d',
                              columnToTakeKeysFrom= 'meas',
                              columnToTakeValuesFrom= 'val',
                              my_db = my_db)
  tab <- blocks_to_rowrecs_q('d',
                             keyColumns = NULL,
                             controlTable = cT,
                             my_db = my_db)
  qlook(my_db, tab)
  DBI::dbDisconnect(my_db)
}
```

`build_pivot_control` *Build a blocks_to_rowrecs()/rowrecs_to_blocks() control table that specifies a pivot from a* `data.frame`.

Description

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html.

Usage

```
build_pivot_control(table, columnToTakeKeysFrom, columnToTakeValuesFrom,
  ..., prefix = columnToTakeKeysFrom, sep = NULL)

## Default S3 method:
build_pivot_control(table, columnToTakeKeysFrom,
  columnToTakeValuesFrom, ..., prefix = columnToTakeKeysFrom,
  sep = NULL)

## S3 method for class 'relop'
```

```
build_pivot_control(table, columnToTakeKeysFrom,
  columnToTakeValuesFrom, ..., prefix = columnToTakeKeysFrom,
  sep = NULL, tmp_name_source = wrapr::mk_tmp_name_source("bpc"),
  temporary = FALSE)
```

Arguments

table	data.frame to scan for new column names (in-memory data.frame).
columnToTakeKeysFrom	character name of column build new column names from.
columnToTakeValuesFrom	character name of column to get values from.
...	not used, force later args to be by name
prefix	column name prefix (only used when sep is not NULL)
sep	separator to build complex column names.
tmp_name_source	a tempNameGenerator from cdata::mk_tmp_name_source()
temporary	logical, if TRUE use temporary tables

Value

control table

See Also

blocks_to_rowrecs, build_pivot_control_q

Examples

```
  d <- data.frame(measType = c("wt", "ht"),
                  measValue = c(150, 6),
                  stringsAsFactors = FALSE)
  build_pivot_control(d,
                      'measType', 'measValue',
                      sep = '_')


d <- data.frame(measType = c("wt", "ht"),
                measValue = c(150, 6),
                stringsAsFactors = FALSE)

ops <- rquery::local_td(d) %.>%
  build_pivot_control(.,
                      'measType', 'measValue',
                      sep = '_')
cat(format(ops))

if(requireNamespace("rqdatatable", quietly = TRUE)) {
```

```
  library("rqdatatable")
  d %.>%
    ops %.>%
    print(.)
}

if(requireNamespace("RSQLite", quietly = TRUE)) {
  db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  DBI::dbWriteTable(db,
                    'd',
                    d,
                    overwrite = TRUE,
                    temporary = TRUE)
  db %.>%
    ops %.>%
    print(.)
  DBI::dbDisconnect(db)
}
```

`build_pivot_control_q` *Build a blocks_to_rowrecs_q() control table that specifies a pivot (query based, takes name of table).*

Description

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html.

Usage

```
build_pivot_control_q(tableName, columnToTakeKeysFrom,
  columnToTakeValuesFrom, my_db, ..., prefix = columnToTakeKeysFrom,
  sep = NULL)
```

Arguments

`tableName`	Name of table to scan for new column names.
`columnToTakeKeysFrom`	character name of column build new column names from.
`columnToTakeValuesFrom`	character name of column to get values from.
`my_db`	db handle
`...`	not used, force later args to be by name
`prefix`	column name prefix (only used when sep is not NULL)
`sep`	separator to build complex column names.

Value

control table

See Also

blocks_to_rowrecs_q, build_pivot_control_q

Examples

```
if (requireNamespace("DBI", quietly = TRUE) &&
  requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  d <- data.frame(measType = c("wt", "ht"),
                  measValue = c(150, 6),
                  stringsAsFactors = FALSE)
  DBI::dbWriteTable(my_db,
                    'd',
                    d,
                    overwrite = TRUE,
                    temporary = TRUE)
  build_pivot_control_q('d', 'measType', 'measValue',
                        my_db = my_db,
                        sep = '_') %.>%
     print(.)
  DBI::dbDisconnect(my_db)
}
```

---

build_unpivot_control *Build a rowrecs_to_blocks() control table that specifies a un-pivot (or "shred").*

Description

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html and here https://github.com/WinVector/cdata.

Usage

```
build_unpivot_control(nameForNewKeyColumn, nameForNewValueColumn,
  columnsToTakeFrom, ...)
```

Arguments

nameForNewKeyColumn	character name of column to write new keys in.
nameForNewValueColumn	character name of column to write new values in.

columnsToTakeFrom
: character array names of columns to take values from.

... : not used, force later args to be by name

Value

control table

See Also

rowrecs_to_blocks_q, rowrecs_to_blocks

Examples

```
build_unpivot_control("measurmentType", "measurmentValue", c("c1", "c2"))
```

---

cdata *cdata: Fluid Data Transformations.*

Description

Supplies implementations of higher order "fluid data" transforms. These' transforms move data between rows and columns, are controlled by a graphical transformation specification, and have pivot and un-pivot as special cases. Large scale implementation is based on 'DBI', so should be usable on 'DBI' compliant data sources (include large systems such as 'PostgreSQL' and 'Spark'). Convenience adapters are provided for in-memory 'data.frame's. A theory of fluid data transforms can be found in the following articles: http://winvector.github.io/FluidData/FluidDataReshapingWithCdata.html, https://github.com/WinVector/cdata and https://winvector.github.io/FluidData/FluidData.html.

---

checkColsFormUniqueKeys *Check that a set of columns form unique keys.*

Description

For local data.frame only.

Usage

```
checkColsFormUniqueKeys(data, keyColNames)
```

Arguments

data	data.frame to work with.
keyColNames	character array of column names to check.

Value

logical TRUE if the rows of data are unique addressable by the columns named in keyColNames.

Examples

```
d <- data.frame(key = c('a','a', 'b'), k2 = c(1 ,2, 2))
checkColsFormUniqueKeys(d, 'key') # should be FALSE
checkColsFormUniqueKeys(d, c('key', 'k2')) # should be TRUE
```

---

map_fields *Map field values from one column into new derived columns (takes a* data.frame*).*

Description

Map field values from one column into new derived columns (takes a data.frame).

Usage

```
map_fields(d, cname, m)
```

Arguments

d	name of table to re-map.
cname	name of column to re-map.
m	name of table of data describing the mapping (cname column is source, derived columns are destinations).

Value

re-mapped table

Examples

```
d <- data.frame(what = c("acc", "loss",
                         "val_acc", "val_loss"),
                score = c(0.8, 1.2,
                       0.7, 1.7),
                stringsAsFactors = FALSE)
m <- data.frame(what = c("acc", "loss",
                         "val_acc", "val_loss"),
                measure = c("accuracy", "log-loss",
                            "accuracy", "log-loss"),
                dataset = c("train", "train", "validation", "validation"),
                stringsAsFactors = FALSE)
map_fields(d, 'what', m)
```

`map_fields_q` *Map field values from one column into new derived columns (query based, takes name of table).*

Description

Map field values from one column into new derived columns (query based, takes name of table).

Usage

```
map_fields_q(dname, cname, mname, my_db, rname)
```

Arguments

`dname`	name of table to re-map.
`cname`	name of column to re-map.
`mname`	name of table of data describing the mapping (cname column is source, derived columns are destinations).
`my_db`	DBI database handle.
`rname`	name of result table.

Value

re-mapped table

Examples

```
if (requireNamespace("DBI", quietly = TRUE) &&
  requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- DBI::dbConnect(RSQLite::SQLite(),
                          ":memory:")
  DBI::dbWriteTable(
    my_db,
    'd',
    data.frame(what = c("acc", "loss",
                        "val_acc", "val_loss"),
               score = c(0.8, 1.2,
                         0.7, 1.7),
               stringsAsFactors = FALSE),
    overwrite = TRUE,
    temporary = TRUE)
  DBI::dbWriteTable(
    my_db,
    'm',
    data.frame(what = c("acc", "loss",
                        "val_acc", "val_loss"),
               measure = c("accuracy", "log-loss",
                           "accuracy", "log-loss"),
               dataset = c("train", "train", "validation", "validation"),
               stringsAsFactors = FALSE),
    overwrite = TRUE,
    temporary = TRUE)

  map_fields_q('d', 'what', 'm', my_db, "dm")
  cdata::qlook(my_db, 'dm')
  DBI::dbDisconnect(my_db)
}
```

`pivot_to_rowrecs` *Move values from rows to columns (pivot).*

Description

This is a convenience notation for `blocks_to_rowrecs`. For a tutorial please try `vignette('RowsAndColumns', package='`

Usage

```
pivot_to_rowrecs(data, columnToTakeKeysFrom, columnToTakeValuesFrom,
  rowKeyColumns, ..., sep = NULL, checkNames = TRUE,
  checkKeys = TRUE, strict = FALSE, use_data_table = FALSE)
```

Arguments

data	data.frame to work with (must be local, for remote please try moveValuesToColumns*).
columnToTakeKeysFrom	character name of column build new column names from.
columnToTakeValuesFrom	character name of column to get values from.
rowKeyColumns	character array names columns that should be table keys.
...	force later arguments to bind by name.
sep	character if not null build more detailed column names.
checkNames	logical, if TRUE check names.
checkKeys	logical, if TRUE check keyColumns uniquely identify blocks (required).
strict	logical, if TRUE check control table name forms
use_data_table	logical if TRUE try to use data.table for the pivots.

Value

new data.frame with values moved to columns.

See Also

unpivot_to_blocks, blocks_to_rowrecs

Examples

```
d <- data.frame(meas= c('AUC', 'R2'), val= c(0.6, 0.2))
pivot_to_rowrecs(d,
                 columnToTakeKeysFrom= 'meas',
                 columnToTakeValuesFrom= 'val',
                 rowKeyColumns= c()) %.>%
   print(.)
```

---

qlook *Quick look at remote data*

Description

Quick look at remote data

Usage

```
qlook(my_db, tableName, displayRows = 10, countRows = TRUE)
```

Arguments

my_db	DBI database handle
tableName	name of table to look at
displayRows	number of rows to sample
countRows	logical, if TRUE return row count.

Value

str view of data

Examples

```
if ( requireNamespace("DBI", quietly = TRUE) &&
  requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  DBI::dbWriteTable(my_db,
                    'd',
                    data.frame(AUC = 0.6, R2 = 0.2),
                    overwrite = TRUE,
                    temporary = TRUE)
  qlook(my_db, 'd')
  DBI::dbDisconnect(my_db)
}
```

---

rowrecs_to_blocks *Map a set of columns to rows (takes a* data.frame*).*

Description

Transform data facts from columns into additional rows controlTable.

Usage

```
rowrecs_to_blocks(wideTable, controlTable, ..., checkNames = TRUE,
  checkKeys = FALSE, strict = FALSE, columnsToCopy = NULL,
  use_data_table = FALSE)

## Default S3 method:
rowrecs_to_blocks(wideTable, controlTable, ...,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  columnsToCopy = NULL, use_data_table = FALSE)

## S3 method for class 'relop'
rowrecs_to_blocks(wideTable, controlTable, ...,
  checkNames = TRUE, strict = FALSE, columnsToCopy = NULL,
```

```
  use_data_table = TRUE,
  tmp_name_source = wrapr::mk_tmp_name_source("rrtbl"),
  temporary = TRUE)
```

Arguments

wideTable	data.frame containing data to be mapped (in-memory data.frame).
controlTable	table specifying mapping (local data frame).
...	force later arguments to be by name.
checkNames	logical, if TRUE check names.
checkKeys	logical, if TRUE check columnsToCopy form row keys (not a requirement, unless you want to be able to invert the operation).
strict	logical, if TRUE check control table name forms.
columnsToCopy	character array of column names to copy.
use_data_table	logical if TRUE try to use data.table for the un-pivots.
tmp_name_source	a tempNameGenerator from cdata::mk_tmp_name_source()
temporary	logical, if TRUE make result temporary.

Details

This is using the theory of "fluid data" (https://github.com/WinVector/cdata), which includes the principle that each data cell has coordinates independent of the storage details and storage detail dependent coordinates (usually row-id, column-id, and group-id) can be re-derived at will (the other principle is that there may not be "one true preferred data shape" and many re-shapings of data may be needed to match data to different algorithms and methods).

The controlTable defines the names of each data element in the two notations: the notation of the tall table (which is row oriented) and the notation of the wide table (which is column oriented). controlTable[, 1] (the group label) cross colnames(controlTable) (the column labels) are names of data cells in the long form. controlTable[, 2:ncol(controlTable)] (column labels) are names of data cells in the wide form. To get behavior similar to tidyr::gather/spread one builds the control table by running an appropriate query over the data.

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html and here https://github.com/WinVector/cdata.

Value

long table built by mapping wideTable to one row per group

See Also

build_unpivot_control, blocks_to_rowrecs_q

Examples

```
  # un-pivot example
  d <- data.frame(AUC = 0.6, R2 = 0.2)
  cT <- build_unpivot_control(nameForNewKeyColumn= 'meas',
                              nameForNewValueColumn= 'val',
                              columnsToTakeFrom= c('AUC', 'R2'))
  rowrecs_to_blocks(d, cT)



d <- data.frame(AUC = 0.6, R2 = 0.2)
cT <- build_unpivot_control(
  nameForNewKeyColumn= 'meas',
  nameForNewValueColumn= 'val',
  columnsToTakeFrom= c('AUC', 'R2'))

ops <- rquery::local_td(d) %.>%
  rowrecs_to_blocks(., cT)
cat(format(ops))

if(requireNamespace("rqdatatable", quietly = TRUE)) {
  library("rqdatatable")
  d %.>%
    ops %.>%
    print(.)
}

if(requireNamespace("RSQLite", quietly = TRUE)) {
  db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  DBI::dbWriteTable(db,
                    'd',
                    d,
                    overwrite = TRUE,
                    temporary = TRUE)
  db %.>%
    ops %.>%
    print(.)
  DBI::dbDisconnect(db)
}
```

`rowrecs_to_blocks_q` *Map a set of columns to rows (query based, take name of table).*

Description

Transform data facts from columns into additional rows using SQL and controlTable.

Usage

```
rowrecs_to_blocks_q(wideTable, controlTable, my_db, ...,
  columnsToCopy = NULL,
  tempNameGenerator = mk_tmp_name_source("mvtrq"), strict = FALSE,
  checkNames = TRUE, showQuery = FALSE, defaultValue = NULL,
  temporary = FALSE, resultName = NULL)
```

Arguments

wideTable	name of table containing data to be mapped (db/Spark data)
controlTable	table specifying mapping (local data frame)
my_db	db handle
...	force later arguments to be by name.
columnsToCopy	character array of column names to copy
tempNameGenerator	a tempNameGenerator from cdata::mk_tmp_name_source()
strict	logical, if TRUE check control table name forms
checkNames	logical, if TRUE check names
showQuery	if TRUE print query
defaultValue	if not NULL literal to use for non-match values.
temporary	logical, if TRUE make result temporary.
resultName	character, name for result table.

Details

This is using the theory of "fluid data"n (https://github.com/WinVector/cdata), which includes the principle that each data cell has coordinates independent of the storage details and storage detail dependent coordinates (usually row-id, column-id, and group-id) can be re-derived at will (the other principle is that there may not be "one true preferred data shape" and many re-shapings of data may be needed to match data to different algorithms and methods).

The controlTable defines the names of each data element in the two notations: the notation of the tall table (which is row oriented) and the notation of the wide table (which is column oriented). controlTable[, 1] (the group label) cross colnames(controlTable) (the column labels) are names of data cells in the long form. controlTable[, 2:ncol(controlTable)] (column labels) are names of data cells in the wide form. To get behavior similar to tidyr::gather/spread one builds the control table by running an appropriate query over the data.

Some discussion and examples can be found here: https://winvector.github.io/FluidData/FluidData.html and here https://github.com/WinVector/cdata.

Value

long table built by mapping wideTable to one row per group

See Also

build_unpivot_control, blocks_to_rowrecs_q, rowrecs_to_blocks

Examples

```
if (requireNamespace("DBI", quietly = TRUE) &&
  requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")

  # un-pivot example
  d <- data.frame(AUC = 0.6, R2 = 0.2)
  DBI::dbWriteTable(my_db,
                    'd',
                    d,
                    overwrite = TRUE,
                    temporary = TRUE)
  cT <- build_unpivot_control(nameForNewKeyColumn= 'meas',
                              nameForNewValueColumn= 'val',
                              columnsToTakeFrom= c('AUC', 'R2'))
  tab <- rowrecs_to_blocks_q('d', cT, my_db = my_db)
  qlook(my_db, tab)
  DBI::dbDisconnect(my_db)
}
```

---

unpivot_to_blocks *Move values from columns to rows (anti-pivot, or "shred").*

Description

This is a convenience notation for rowrecs_to_blocks. For a tutorial please try vignette('RowsAndColumns', package='

Usage

```
unpivot_to_blocks(data, nameForNewKeyColumn, nameForNewValueColumn,
  columnsToTakeFrom, ..., nameForNewClassColumn = NULL,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  use_data_table = FALSE)

## Default S3 method:
unpivot_to_blocks(data, nameForNewKeyColumn,
  nameForNewValueColumn, columnsToTakeFrom, ...,
  nameForNewClassColumn = NULL, checkNames = TRUE, checkKeys = FALSE,
  strict = FALSE, use_data_table = FALSE)

## S3 method for class 'relop'
unpivot_to_blocks(data, nameForNewKeyColumn,
  nameForNewValueColumn, columnsToTakeFrom, ...,
  nameForNewClassColumn = NULL,
  tmp_name_source = wrapr::mk_tmp_name_source("upb"), temporary = TRUE)
```

Arguments

data	data.frame to work with.
nameForNewKeyColumn	character name of column to write new keys in.
nameForNewValueColumn	character name of column to write new values in.
columnsToTakeFrom	character array names of columns to take values from.
...	force later arguments to bind by name.
nameForNewClassColumn	optional name to land original cell classes to.
checkNames	logical, if TRUE check names.
checkKeys	logical, if TRUE check columnsToCopy form row keys (not a requirement, unless you want to be able to invert the operation).
strict	logical, if TRUE check control table name forms.
use_data_table	logical if TRUE try to use data.table for the un-pivots.
tmp_name_source	a tempNameGenerator from cdata::mk_tmp_name_source()
temporary	logical, if TRUE make result temporary.

Value

new data.frame with values moved to rows.

See Also

pivot_to_rowrecs, rowrecs_to_blocks

Examples

```
  d <- data.frame(AUC= 0.6, R2= 0.2)
  unpivot_to_blocks(d,
                    nameForNewKeyColumn= 'meas',
                    nameForNewValueColumn= 'val',
                    columnsToTakeFrom= c('AUC', 'R2')) %.>%
     print(.)


d <- data.frame(AUC= 0.6, R2= 0.2)
ops <- rquery::local_td(d) %.>%
  unpivot_to_blocks(
    .,
    nameForNewKeyColumn= 'meas',
    nameForNewValueColumn= 'val',
    columnsToTakeFrom= c('AUC', 'R2'))
cat(format(ops))
```

```
if(requireNamespace("rqdatatable", quietly = TRUE)) {
  library("rqdatatable")
  d %.>%
    ops %.>%
    print(.)
}

if(requireNamespace("RSQLite", quietly = TRUE)) {
  db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
  DBI::dbWriteTable(db,
                    'd',
                    d,
                    overwrite = TRUE,
                    temporary = TRUE)
  db %.>%
    ops %.>%
    print(.)
  DBI::dbDisconnect(db)
}
```

Index