

Package ‘cdata’

April 20, 2019

Type Package

Title Fluid Data Transformations

Version 1.0.9

Date 2019-04-20

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 coordinatized data specification and fluid 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

This package introduces the idea of control table specification of data transforms (later also adapted from 'cdata' by 'tidyr').

Works on in-memory data or on remote data using 'rquery' and 'SQL' database interfaces.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 3.4.0)

Imports wrapr (>= 1.8.6), rquery (>= 1.3.2), methods, stats

Suggests DBI, RSQLite, knitr, rqdatatable (>= 1.1.4), RUnit

VignetteBuilder knitr

ByteCompile true

NeedsCompilation no

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

Repository CRAN

Date/Publication 2019-04-20 15:40:04 UTC

R topics documented:

blocks_to_rowrecs	2
blocks_to_rowrecs_spec	4
build_pivot_control	6
build_unpivot_control	8
cdata	9
convert_records	9
layout_by	11
layout_by.blocks_to_rowrecs_spec	12
layout_by.rowrecs_to_blocks_spec	13
map_fields	14
map_fields_q	15
pivot_to_rowrecs	16
rowrecs_to_blocks	17
rowrecs_to_blocks_spec	19
run_cdata_tests	21
unpivot_to_blocks	22
% %	24
%**%	25

Index **26**

blocks_to_rowrecs	<i>Map sets rows to columns (takes a data.frame).</i>
-------------------	---

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, controlTableKeys = colnames(controlTable)[[1]],
  tmp_name_source = wrapr::mk_tmp_name_source("bltrr"),
  temporary = TRUE)
```

```
## Default S3 method:
```

```
blocks_to_rowrecs(tallTable, keyColumns, controlTable,
  ..., columnsToCopy = NULL, checkNames = TRUE, checkKeys = TRUE,
  strict = FALSE, controlTableKeys = colnames(controlTable)[[1]],
  tmp_name_source = wrapr::mk_tmp_name_source("btrd"),
  temporary = TRUE)
```

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

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

```
strict = FALSE, controlTableKeys = colnames(controlTable)[[1]],
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
controlTableKeys	character, which column names of the control table are considered to be keys.
tmp_name_source	a tmpNameGenerator from cdata::mk_tmp_name_source()
temporary	logical, if TRUE use temporary tables

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](#)

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_spec

Create a block records to row records transform specification.

Description

Create a block records to row records transform specification object that holds the pivot control table, specification of extra row keys, and control table keys.

Usage

```
blocks_to_rowrecs_spec(controlTable, ..., recordKeys = character(0),
  controlTableKeys = colnames(controlTable)[[1]], checkNames = TRUE,
  checkKeys = TRUE, strict = FALSE)
```

Arguments

`controlTable` an all character data frame or cdata pivot control.
`...` not used, force later arguments to bind by name.
`recordKeys` vector of columns identifying records.
`controlTableKeys` vector of keying columns of the controlTable.
`checkNames` passed to `blocks_to_rowrecs`.
`checkKeys` passed to `blocks_to_rowrecs`.
`strict` passed to `blocks_to_rowrecs`.

Value

a record specification object

Examples

```
d <- wrapr::build_frame(
  "id", "measure", "value" |
  1 , "AUC" , 0.7 |
  1 , "R2" , 0.4 |
  2 , "AUC" , 0.8 |
  2 , "R2" , 0.5 )

transform <- blocks_to_rowrecs_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC" , AUC |
    "R2" , R2 ),
  recordKeys = "id")

print(transform)

d %>% transform

inv_transform <- t(transform)
print(inv_transform)
```

```
# identity (in structure)
d %>% transform %>% inv_transform

# identity again (using .() "immediate" notation)
d %>% transform %>% .(t(transform))
```

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,
  tmp_name_source = wrapr::mk_tmp_name_source("bpc"),
  temporary = FALSE)
```

Default S3 method:

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

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 `cddata::mk_tmp_name_source()`
`temporary` logical, if TRUE use temporary tables

Value

control table

See Also

[blocks_to_rowrecs](#)

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_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](#)

Examples

```
build_unpivot_control("measurementType", "measurementValue", c("c1", "c2"))
```

cdata	cdata: <i>Fluid Data Transformations.</i>
-------	---

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 'rquery', so should be usable on 'SQL' compliant data sources (include large systems such as 'PostgreSQL' and 'Spark'). This package introduces the idea of control table specification of data transforms (later also adapted from 'cdata' by 'tidyr'). 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>.

convert_records	<i>General transform from arbitrary record shape to arbitrary record shape.</i>
-----------------	---

Description

General transform from arbitrary record shape to arbitrary record shape.

Usage

```
convert_records(table, incoming_shape, outgoing_shape, ...,
  keyColumns = NULL, columnsToCopy_in = NULL, checkNames = TRUE,
  checkKeys = FALSE, strict = FALSE,
  incoming_controlTableKeys = colnames(incoming_shape)[[1]],
  outgoing_controlTableKeys = colnames(outgoing_shape)[[1]],
  tmp_name_source = wrapr::mk_tmp_name_source("crec"),
  temporary = TRUE)
```

Arguments

table	data.frame or relap.
incoming_shape	data.frame, definition of incoming record shape.
outgoing_shape	data.frame, definition of outgoing record shape.
...	force later arguments to bind by name.
keyColumns	character vector of column defining incoming row groups
columnsToCopy_in	character array of incoming column names to copy.
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.
incoming_controlTableKeys	character, which column names of the incoming control table are considered to be keys.
outgoing_controlTableKeys	character, which column names of the outgoing control table are considered to be keys.
tmp_name_source	a tempNameGenerator from cdata::mk_tmp_name_source()
temporary	logical, if TRUE use temporary tables

Value

processing pipeline or transformed table

Examples

```
incoming_shape <- qchar_frame(
  "row", "col1", "col2", "col3" |
  "row1", v11, v12, v13 |
  "row2", v21, v22, v23 |
  "row3", v31, v32, v33 )

outgoing_shape <- qchar_frame(
  "column", "row1", "row2", "row3" |
  "col1", v11, v21, v31 |
  "col2", v12, v22, v32 |
  "col3", v13, v23, v33 )

data <- build_frame(
  'record_id', 'row', 'col1', 'col2', 'col3' |
  1, 'row1', 1, 2, 3 |
  1, 'row2', 4, 5, 6 |
  1, 'row3', 7, 8, 9 |
  2, 'row1', 11, 12, 13 |
  2, 'row2', 14, 15, 16 |
  2, 'row3', 17, 18, 19 )

print(data)

convert_records(
  data,
  keyColumns = 'record_id',
  incoming_shape = incoming_shape,
  outgoing_shape = outgoing_shape)
```

```
td <- rquery::local_td(data)

ops <- convert_records(
  td,
  keyColumns = 'record_id',
  incoming_shape = incoming_shape,
  outgoing_shape = outgoing_shape)

cat(format(ops))

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

layout_by

Use transform spec to layout data.

Description

Use transform spec to layout data.

Usage

```
layout_by(transform, table)
```

Arguments

transform object of class rowrecs_to_blocks_spec
table data.frame or relop.

Value

re-arranged data or data reference (relop).

Examples

```
d <- wrapr::build_frame(
  "id" , "AUC", "R2" |
  1   , 0.7 , 0.4 |
  2   , 0.8 , 0.5 )
transform <- rowrecs_to_blocks_spec(
  wrapr::qchar_frame(
```

```

      "measure", "value" |
      "AUC"     , AUC    |
      "R2"     , R2     ),
    recordKeys = "id")
print(transform)
layout_by(transform, d)

d <- wrapr::build_frame(
  "id", "measure", "value" |
  1   , "AUC"     , 0.7   |
  1   , "R2"     , 0.4   |
  2   , "AUC"     , 0.8   |
  2   , "R2"     , 0.5   )
transform <- blocks_to_rowrecs_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC"     , AUC    |
    "R2"     , R2     ),
  recordKeys = "id")
print(transform)
layout_by(transform, d)

```

```
layout_by.blocks_to_rowrecs_spec
```

Use transform spec to layout data.

Description

Use transform spec to layout data.

Usage

```
## S3 method for class 'blocks_to_rowrecs_spec'
layout_by(transform, table)
```

Arguments

transform object of class blocks_to_rowrecs_spec.
table data.frame or relop.

Value

re-arranged data or data reference (relop).

Examples

```
d <- wrapr::build_frame(
  "id", "measure", "value" |
  1 , "AUC" , 0.7 |
  1 , "R2" , 0.4 |
  2 , "AUC" , 0.8 |
  2 , "R2" , 0.5 )

transform <- blocks_to_rowrecs_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC" , AUC |
    "R2" , R2 ),
  recordKeys = "id")

print(transform)

layout_by(transform, d)
```

```
layout_by.rowrecs_to_blocks_spec
      Use transform spec to layout data.
```

Description

Use transform spec to layout data.

Usage

```
## S3 method for class 'rowrecs_to_blocks_spec'
layout_by(transform, table)
```

Arguments

```
transform      object of class rowrecs_to_blocks_spec
table          data.frame or relop.
```

Value

re-arranged data or data reference (relop).

Examples

```
d <- wrapr::build_frame(
  "id" , "AUC", "R2" |
  1   , 0.7 , 0.4 |
  2   , 0.8 , 0.5 )

transform <- rowrecs_to_blocks_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC"     , AUC    |
    "R2"      , R2     ),
  recordKeys = "id")

print(transform)
layout_by(transform, d)
```

map_fields	<i>Map field values from one column into new derived columns (takes a data.frame).</i>
------------	--

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

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	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	<i>Move values from rows to columns (pivot).</i>
------------------	--

Description

This is a convenience notation for `blocks_to_rowrecs`. For a tutorial please try <https://winvector.github.io/cdata/articles/blocksrecs.html>.

Usage

```

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

```

```

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

```

Arguments

<code>data</code>	data.frame to work with (must be local, for remote please try <code>moveValuesToColumns*</code>).
<code>columnToTakeKeysFrom</code>	character name of column build new column names from.
<code>columnToTakeValuesFrom</code>	character name of column to get values from.
<code>rowKeyColumns</code>	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

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(.)
```

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,
  controlTableKeys = colnames(controlTable)[[1]], columnsToCopy = NULL,
  tmp_name_source = wrapr::mk_tmp_name_source("rrtbl"),
  temporary = TRUE)
```

Default S3 method:

```
rowrecs_to_blocks(wideTable, controlTable, ...,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  controlTableKeys = colnames(controlTable)[[1]], columnsToCopy = NULL,
  tmp_name_source = wrapr::mk_tmp_name_source("rrtobd"),
  temporary = TRUE)
```

```
## S3 method for class 'relop'
rowrecs_to_blocks(wideTable, controlTable, ...,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  controlTableKeys = colnames(controlTable)[[1]], columnsToCopy = NULL,
  tmp_name_source = wrapr::mk_tmp_name_source("rrtbl"),
  temporary = TRUE)
```

Arguments

<code>wideTable</code>	data.frame containing data to be mapped (in-memory data.frame).
<code>controlTable</code>	table specifying mapping (local data frame).
<code>...</code>	force later arguments to be by name.
<code>checkNames</code>	logical, if TRUE check names.
<code>checkKeys</code>	logical, if TRUE check columnsToCopy form row keys (not a requirement, unless you want to be able to invert the operation).
<code>strict</code>	logical, if TRUE check control table name forms.
<code>controlTableKeys</code>	character, which column names of the control table are considered to be keys.
<code>columnsToCopy</code>	character array of column names to copy.
<code>tmp_name_source</code>	a tempNameGenerator from <code>cdata::mk_tmp_name_source()</code>
<code>temporary</code>	logical, if TRUE use temporary tables

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](#)

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_spec

Create a row records to block records transform specification.

Description

Create a row records to block records transform specification object that holds the pivot control table, specification of extra row keys, and control table keys.

Usage

```
rowrecs_to_blocks_spec(controlTable, ..., recordKeys = character(0),
  controlTableKeys = colnames(controlTable)[[1]], checkNames = TRUE,
  checkKeys = TRUE, strict = FALSE)
```

Arguments

`controlTable` an all character data frame or cdata pivot control.
`...` not used, force later arguments to bind by name.
`recordKeys` vector of columns identifying records.
`controlTableKeys` vector of keying columns of the controlTable.
`checkNames` passed to `rowrecs_to_blocks`.
`checkKeys` passed to `rowrecs_to_blocks`.
`strict` passed to `rowrecs_to_blocks`.

Value

a record specification object

Examples

```
d <- wrapr::build_frame(
  "id"  , "AUC", "R2" |
  1    , 0.7  , 0.4 |
  2    , 0.8  , 0.5 )

transform <- rowrecs_to_blocks_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC"    , AUC    |
    "R2"     , R2     ),
  recordKeys = "id")

print(transform)

d %>% transform

inv_transform <- t(transform)
print(inv_transform)

# identity (in structure)
d %>% transform %>% inv_transform

# identity again (using .() "immediate" notation)
d %>% transform %>% .(t(transform))
```

run_cdata_tests	<i>Run cdata package tests.</i>
-----------------	---------------------------------

Description

Run the tests included with the cdata package (assumes cdata package is attached).

Usage

```
run_cdata_tests(..., verbose = TRUE, package_test_dirs = "unit_tests",
  test_dirs = character(0), stop_on_issue = TRUE,
  stop_if_no_tests = TRUE, require_RUnit_attached = FALSE,
  require_pkg_attached = TRUE, rngKind = "Mersenne-Twister",
  rngNormalKind = "Inversion")
```

Arguments

...	not used, force later arguments to bind by name.
verbose	logical, if TRUE print more.
package_test_dirs	directory names to look for in the installed package.
test_dirs	paths to look for tests in.
stop_on_issue	logical, if TRUE stop after errors or failures.
stop_if_no_tests	logical, if TRUE stop if no tests were found.
require_RUnit_attached	logical, if TRUE require RUnit be attached before testing.
require_pkg_attached	logical, if TRUE require pkg be attached before testing.
rngKind	pseudo-random number generator method name.
rngNormalKind	pseudo-random normal generator method name.

Value

RUnit test results (invisible).

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 <https://winvector.github.io/cdata/articles/blocksrecs.html>.

Usage

```
unpivot_to_blocks(data, nameForNewKeyColumn, nameForNewValueColumn,
  columnsToTakeFrom, ..., nameForNewClassColumn = NULL,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  tmp_name_source = wrapr::mk_tmp_name_source("upb"), temporary = TRUE)
```

```
layout_to_blocks(data, nameForNewKeyColumn, nameForNewValueColumn,
  columnsToTakeFrom, ..., nameForNewClassColumn = NULL,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  tmp_name_source = wrapr::mk_tmp_name_source("upb"), temporary = TRUE)
```

```
pivot_to_blocks(data, nameForNewKeyColumn, nameForNewValueColumn,
  columnsToTakeFrom, ..., nameForNewClassColumn = NULL,
  checkNames = TRUE, checkKeys = FALSE, strict = FALSE,
  tmp_name_source = wrapr::mk_tmp_name_source("upb"), temporary = TRUE)
```

```
## Default S3 method:
```

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

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

```
unpivot_to_blocks(data, nameForNewKeyColumn,
  nameForNewValueColumn, columnsToTakeFrom, ..., checkNames = TRUE,
  checkKeys = FALSE, strict = FALSE, 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.

`tmp_name_source` a `tempNameGenerator` from `cdta::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("rdatatable", quietly = TRUE)) {
  library("rdatatable")
  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)
}

```

 %//%

Factor-out (aggregate/project) block records into row records.

Description

Call `blocks_to_rowrecs()`.

Usage

```
table %//% transform
```

Arguments

```

table          data (data.frame or relop).
transform      a rowrecs_to_blocks_spec.

```

Value

`blocks_to_rowrecs()` result.

Examples

```

d <- wrapr::build_frame(
  "id", "measure", "value" |
  1 , "AUC" , 0.7 |
  1 , "R2" , 0.4 |
  2 , "AUC" , 0.8 |
  2 , "R2" , 0.5 )

transform <- blocks_to_rowrecs_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC" , AUC |
    "R2" , R2 ),
  recordKeys = "id")

d %//% transform

```



```
# identity (in structure)
d %//% transform %**% t(transform)
```

%**%

Multiply/join row records into block records.

Description

Call `rowrecs_to_blocks()`.

Usage

```
table %**% transform
```

Arguments

```
table          data (data.frame or relop).
transform      a rowrecs_to_blocks_spec.
```

Value

`rowrecs_to_blocks()` result.

Examples

```
d <- wrapr::build_frame(
  "id", "AUC", "R2" |
  1   , 0.7  , 0.4 |
  2   , 0.8  , 0.5 )

transform <- rowrecs_to_blocks_spec(
  wrapr::qchar_frame(
    "measure", "value" |
    "AUC"     , AUC    |
    "R2"      , R2     ),
  recordKeys = "id")

d %**% transform

# identity (in structure)
d %**% transform %//% t(transform)
```

Index

[%**%, 25](#)

[%//%, 24](#)

[blocks_to_rowrecs, 2, 7, 17, 18](#)

[blocks_to_rowrecs_spec, 4](#)

[build_pivot_control, 3, 6](#)

[build_unpivot_control, 8, 18](#)

[cdata, 9](#)

[cdata-package \(cdata\), 9](#)

[convert_records, 9](#)

[layout_by, 11](#)

[layout_by.blocks_to_rowrecs_spec, 12](#)

[layout_by.rowrecs_to_blocks_spec, 13](#)

[layout_to_blocks \(unpivot_to_blocks\), 22](#)

[layout_to_rowrecs \(pivot_to_rowrecs\), 16](#)

[map_fields, 14](#)

[map_fields_q, 15](#)

[pivot_to_blocks \(unpivot_to_blocks\), 22](#)

[pivot_to_rowrecs, 16, 23](#)

[rowrecs_to_blocks, 3, 8, 17, 23](#)

[rowrecs_to_blocks_spec, 19](#)

[run_cdata_tests, 21](#)

[unpivot_to_blocks, 17, 22](#)