# Package ‘cdata’

August 20, 2023

**Type** Package  
**Title** Fluid Data Transformations  
**Version** 1.2.1  
**Date** 2023-08-19  
**Maintainer** John Mount <jmount@win-vector.com>  
**BugReports** [https://github.com/WinVector/cdata/issues](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'”, [https://winvector.github.io/FluidData/FluidDataReshapingWithCdata.html](https://winvector.github.io/FluidData/FluidDataReshapingWithCdata.html), [DOI:10.5281/zenodo.1173299]. This package introduces the idea of explicit control table specification of data transforms. Works on in-memory data or on remote data using 'rquery' and 'SQL' database interfaces.  
**License** GPL-2 | GPL-3  
**Encoding** UTF-8  
**RoxygenNote** 7.2.3  
**Depends** R (>= 3.4.0), wrapr (>= 2.0.9)  
**Imports** rquery (>= 1.4.9), rqdatatable (>= 1.3.2), methods, stats  
**Suggests** DBI, RSQLite, knitr, rmarkdown, yaml, tinytest  
**VignetteBuilder** knitr  
**ByteCompile** true  
**NeedsCompilation** no  
**Author** John Mount [aut, cre], Nina Zumel [aut], Win-Vector LLC [cph]  
**Repository** CRAN  
**Date/Publication** 2023-08-20 00:02:32 UTC
R topics documented:

cdata-package ................................................................. 2
blocks_to_rowrecs ............................................................ 3
blocks_to_rowrecs_spec .................................................... 6
build_pivot_control ........................................................... 7
build_unpivot_control ....................................................... 9
convert_cdata_spec_to_yaml ............................................... 10
convert_records ............................................................. 11
convert_yaml_to_cdata_spec ............................................... 13
layout_by ................................................................. 14
layout_by.blocks_to_rowrecs_spec ........................................ 15
layout_by.cdata_general_transform_spec ................................ 16
layout_by.rowrecs_to_blocks_spec ....................................... 16
layout_specification ...................................................... 17
map_fields ................................................................. 19
map_fields_q ............................................................... 20
pivot_to_rowrecs ........................................................... 21
rowrecs_to_blocks ......................................................... 23
rowrecs_to_blocks_spec ................................................... 25
unpivot_to_blocks .......................................................... 27
%//% .................................................................. 30
%**% .................................................................. 31

Index ................................................................. 33

cdata-package 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 '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 aslo adapted from 'cdata' by 'tidyr'). A theory of fluid data transforms can be found in the following articles: https://winvector.github.io/FluidData/FluidDataReshapingWithCdata.html, https://github.com/WinVector/cdata and https://winvector.github.io/FluidData/FluidData.html.

Author(s)

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

Authors:

• Nina Zumel <nzumel@win-vector.com>

Other contributors:

• Win-Vector LLC [copyright holder]
blocks_to_rowrecs

See Also
Useful links:

- https://github.com/WinVector/cdata/
- https://winvector.github.io/cdata/

blocks_to_rowrecs  Map data records from block records to row records.

Description
Map data records from block records (which each record may be more than one row) to row records (where each record is a single row).

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,
  allow_rqdatatable = FALSE
)

## Default S3 method:
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("btrd"),
  temporary = TRUE,
  allow_rqdatatable = FALSE
)
blocks_to_rowrecs

## 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,
  allow_rqdatatable = FALSE
)

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 tempNameGenerator from cdata::mk_tmp_name_source()
temporary logical, if TRUE use temporary tables
allow_rqdatatable logical, if TRUE allow rqdatatable shortcutting on simple conversions.

Details

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

```r
# 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(d,
                   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(.)
}
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,
  allow_rqdatatable = 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.
allow_rqdatatable logical, if TRUE allow rqdatatable shortcutting on simple conversions.

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,
## 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 cdata::mk_tmp_name_source()
- **temporary**: logical, if TRUE use temporary tables

### Value

control table

### See Also

blocks_to_rowrecs
Examples

```r
library('dplyr')

# Build a pivot control table
build_pivot_control(d,
                      'measType', 'measValue',
                      sep = '_')

# Build a pivot control table using other data
build_pivot_control(d,
                      'measType', 'measValue',
                      sep = '_')
```

```r
# Define a local td for our pivot control
ops <- rquery::local_td(d) %.>%
   build_pivot_control(.,
                         'measType', 'measValue',
                         sep = '_')

# Format and print the operations
cat(format(ops))
```

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

```r
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 rowreps_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

```r
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

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

---

**convert_cdata_spec_to_yaml**

Convert a `layout_specification`, `blocks_to_rowrecs_spec`, or `rowrecs_to_blocks_spec` to a simple object.

Description

Convert a `layout_specification`, `blocks_to_rowrecs_spec`, or `rowrecs_to_blocks_spec` to a simple object.

Usage

```r
convert_cdata_spec_to_yaml(spec)
```
**convert_records**

**Arguments**

- **spec**  
  a layout_specification, blocks_to_rowrecs_spec, or rowrecs_to_blocks_spec

**Value**

- a simple object suitable for YAML serialization

---

**convert_records**  
*General transform from arbitrary record shape to arbitrary record shape.*

**Description**

General transform from arbitrary record shape to arbitrary record shape.

**Usage**

```r
convert_records(
  table,
  incoming_shape = NULL,
  outgoing_shape = NULL,
  ...,
  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,
  allow_rqdatatable_in = FALSE,
  allow_rqdatatable_out = FALSE
)
```

**Arguments**

- **table**  
  data.frame or relop.
- **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

allow_rqdatatable_in logical, if TRUE allow rqdatatable shortcutting on simple conversions.

allow_rqdatatable_out logical, if TRUE allow rqdatatable shortcutting on simple conversions.

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(
```
convert_yaml_to_cdata_spec

Read a cdata record transform from a simple object (such as is imported from YAML).

Description

Read a cdata record transform from a simple object (such as is imported from YAML).

Usage

convert_yaml_to_cdata_spec(obj)

Arguments

obj object to convert

Value

cdata transform specification
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(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)
Description
Use transform spec to layout data.

Usage
## S3 method for class 'cdata_general_transform_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).

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

```r
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)
```

layout_specification

Create a record to record spec.

Description

Create a general record to record transform specification.

Usage

```r
layout_specification(
  incoming_shape = NULL,
  outgoing_shape = NULL,
  ..., 
  recordKeys = character(0),
  incoming_controlTableKeys = colnames(incoming_shape)[[1]],
  outgoing_controlTableKeys = colnames(outgoing_shape)[[1]],
  checkNames = TRUE,
  checkKeys = TRUE,
  strict = FALSE,
  allow_rqdatatable_in = FALSE,
  allow_rqdatatable_out = FALSE
)
```

Arguments

- **incoming_shape**  data.frame, definition of incoming record shape.
- **outgoing_shape**  data.frame, definition of outgoing record shape.
- **...**  not used, force later arguments to bind by name.
- **recordKeys**  vector of columns identifying records.
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.

checkNames
    passed to rowrecs_to_blocks.

checkKeys
    passed to rowrecs_to_blocks.

strict
    passed to rowrecs_to_blocks.

allow_rqdatatable_in
    logical, if TRUE allow rqdatatable shortcutting on simple conversions.

allow_rqdatatable_out
    logical, if TRUE allow rqdatatable shortcutting on simple conversions.

Value

a record specification object

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)

layout <- layout_specification(
    incoming_shape = incoming_shape,
    outgoing_shape = outgoing_shape,
    recordKeys = 'record_id')
map_fields

print(layout)
data %.>% layout
data %.>% layout %.>% .(t(layout))

<table>
<thead>
<tr>
<th>map_fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map field values from one column into new derived columns (takes a data.frame).</td>
</tr>
</tbody>
</table>

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,
  ...,
  d_qualifiers = NULL,
  m_qualifiers = NULL
)

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.
... force later arguments to be by name.
d_qualifiers optional named ordered vector of strings carrying additional db hierarchy terms, such as schema.
m_qualifiers optional named ordered vector of strings carrying additional db hierarchy terms, such as schema.

Value

re-mapped table

Examples

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

Map data records from block records that have one row per measurement value to row records.

Description

Map data records from block records (where each record may be more than one row) to row records (where each record is a single row). Values specified in rowKeyColumns determine which sets of rows build up records and are copied into the result.

Usage

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

pivot_to_rowrecs(
  strict = FALSE,
  allow_rqdatatable = FALSE
)

layout_to_rowrecs(
  data,
  columnToTakeKeysFrom,
  columnToTakeValuesFrom,
  rowKeyColumns,
  ..., sep = NULL,
  checkNames = TRUE,
  checkKeys = TRUE,
  strict = FALSE,
  allow_rqdatatable = 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
allow_rqdatatable logical, if TRUE allow rqdatatable shortcutting on simple conversions.

Value

ew data.frame with values moved to columns.

See Also

unpivot_to_blocks, blocks_to_rowrecs

Examples

d <- data.frame(model_id = c("m1", "m1"), meas = c("AUC", "R2"), val= c(0.6, 0.2))
pivot_to_rowrecs(d,
  columnToTakeKeysFrom= 'meas',
rowrecs_to_blocks  

Map a data records from row records to block records.

Description

Map a data records from row records (records that are exactly single rows) to block records (records
that may be more than one row).

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,
  allow_rqdatatable = FALSE
)

## 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,
  allow_rqdatatable = FALSE
)

## 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_temp_name_source("rrtbl"),
temporary = TRUE,
allow_rqdatatable = FALSE
)

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.
controlTableKeys  character, which column names of the control table are considered to be keys.
columnsToCopy  character array of column names to copy.
tmp_name_source  a tempNameGenerator from cdata::mk_temp_name_source()
temporary  logical, if TRUE use temporary tables
allow_rqdatatable  logical, if TRUE allow rqdatatable shortcutting on simple conversions.

Details

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.

rowrecs_to_blocks.default will change some factor columns to character, and there are issues with time columns with different time zones.

Value

long table built by mapping wideTable to one row per group
See Also

build_unpivot_control, blocks_to_rowrecs

Examples

```r
# 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)
```

```r
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)
}
```
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 = FALSE,
  strict = FALSE,
  allow_rqdatatable = 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.
allow_rqdatatable logical, if TRUE allow rqdatatable shortcutting on simple conversions.

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))

---

unpivot_to_blocks

Map a data records from row records to block records with one record row per columnsToTakeFrom value.

Description

Map a data records from row records (records that are exactly single rows) to block records (records that may be more than one row). All columns not named in columnsToTakeFrom are copied to each record row in the result.

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,
                    allow_rqdatatable = FALSE)

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,
                    allow_rqdatatable = FALSE)
unpivot_to_blocks

nameForNewClassColumn = NULL,
checkNames = TRUE,
checkKeys = FALSE,
strict = FALSE,
tmp_name_source = wrapr::mk_tmp_name_source("upb"),
temporary = TRUE,
allow_rqdatatable = FALSE)

pivot_to_blocks(
  data,
  nameForNewKeyColumn,
  nameForNewValueColumn,
  columnsToTakeFrom,
  ...
)

## Default S3 method:
unpivot_to_blocks(
  data,
  nameForNewKeyColumn,
  nameForNewValueColumn,
  columnsToTakeFrom,
  ...
)

## S3 method for class 'relop'
unpivot_to_blocks(
  data,
  nameForNewKeyColumn,
  nameForNewValueColumn,
  columnsToTakeFrom,
  ...
)
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 cdata::mk_tmp_name_source()
- **temporary**: logical, if TRUE make result temporary.
- **allow_rqdatatable**: logical, if TRUE allow rqdatatable shortcutting on simple conversions.

Value

new data.frame with values moved to rows.

See Also

- **pivot_to_rowrecs**, **rowrecs_to_blocks**

Examples

```r
# Example 1

d <- data.frame(model_name = "m1", AUC = 0.6, R2 = 0.2)
unpivot_to_blocks(d,
  nameForNewKeyColumn= 'meas',
  nameForNewValueColumn= 'val',
  columnsToTakeFrom= c('AUC', 'R2')) %>%
  print(.)

# Example 2

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)
}

---

**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

%**%, 31
%//%, 30

blocks_to_rowrecs, 3, 8, 22, 25
blocks_to_rowrecs_spec, 6
build.pivot.control, 5, 7
build.unpivot.control, 9, 25

cdata(cdata-package), 2
cdata-package, 2
convert.cdata.spec_to.yaml, 10
convert.records, 11
convert.yaml_to.cdata.spec, 13

layout_by, 14
layout_by.blocks_to_rowrecs_spec, 15
layout_by.cdata_general_transform_spec, 16
layout_by.rowrecs_to_blocks_spec, 16
layout_specification, 17
layout_to_blocks(unpivot_to_blocks), 27
layout_to_rowrecs(pivot_to_rowrecs), 21

map_fields, 19
map_fields_q, 20

pivot_to_blocks(unpivot_to_blocks), 27
pivot_to_rowrecs, 21, 29

rowrecs_to_blocks, 5, 10, 23, 29
rowrecs_to_blocks_spec, 25

unpivot_to_blocks, 22, 27