

# Package ‘replyr’

January 20, 2018

**Type** Package

**Title** Reliable Plying of Remote or Big Data with R

**Version** 0.9.1

**Date** 2018-01-20

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

**URL** <https://github.com/WinVector/replyr/>,  
<https://winvector.github.io/replyr/>

**BugReports** <https://github.com/WinVector/replyr/issues>

**Description** Methods to use 'dplyr' remote data sources ('SQL' databases, 'Spark' 2.0.0 and above) in a reliable ``generic" fashion (generic meaning user code works similarly on all such sources, without needing per-source adaption).

Adds convenience functions to make such tasks more like working with an in-memory R 'data.frame'.

**License** GPL-3

**LazyData** TRUE

**Depends** seplyr (>= 0.5.1), wrapr (>= 1.1.0), cdata (>= 0.5.1)

**Imports** dplyr (>= 0.7.4), dbplyr, DBI, RSQLite

**RoxygenNote** 6.0.1

**Suggests** testthat, knitr, rmarkdown, sparklyr, ggplot2, RPostgreSQL, igraph, DiagrammeR, htmlwidgets, webshot, magick, grid

**VignetteBuilder** knitr

**ByteCompile** true

**NeedsCompilation** no

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

**Repository** CRAN

**Date/Publication** 2018-01-20 22:22:53 UTC

**R topics documented:**

addConstantColumn	3
buildJoinPlan	4
dplyr_src_to_db_handle	5
executeLeftJoinPlan	5
expandColumn	7
gapply	8
inspectDescrAndJoinPlan	10
keysAreUnique	11
key_inspector_all_cols	12
key_inspector_postgresql	13
key_inspector_sqlite	13
makeJoinDiagramSpec	14
renderJoinDiagram	15
replyr	16
replyr_add_ids	16
replyr_apply_f_mapped	17
replyr_arrange	18
replyr_bind_rows	18
replyr_check_ranks	19
replyr_coalesce	20
replyr_colClasses	21
replyr_copy_from	22
replyr_copy_to	23
replyr_dim	24
replyr_drop_table_name	24
replyr_filter	25
replyr_get_src	26
replyr_group_by	26
replyr_hasrows	27
replyr_has_table	28
replyr_inTest	28
replyr_is_local_data	29
replyr_is_MySQL_data	30
replyr_is_Spark_data	30
replyr_list_tables	31
replyr_mapRestrictCols	31
replyr_ncol	33
replyr_nrow	33
replyr_quantile	34
replyr_quantilec	35
replyr_rename	35
replyr_reverseMap	36
replyr_select	37
replyr_split	37
replyr_summary	38
replyr_testCols	40

<i>addConstantColumn</i>	3
replyr_union_all . . . . .	40
replyr_uniqueValues . . . . .	41
tableDescription . . . . .	42
topoSortTables . . . . .	43
%land% . . . . .	44
<b>Index</b>	<b>45</b>

---

<code>addConstantColumn</code>	<i>Add constant to a table.</i>
--------------------------------	---------------------------------

---

### Description

Work around different treatment of character types accross remote data soures when adding a constant column to a table. Deals with issues such as Postgresql requiring a charcater-cast and MySQL not allowing such.

### Usage

```
addConstantColumn(d, colName, val, ...,
  tempNameGenerator = makeTempNameGenerator("replyr_addConstantColumn"))
```

### Arguments

<code>d</code>	data.frame like object to add column to.
<code>colName</code>	character, name of column to add.
<code>val</code>	scalar, value to add.
<code>...</code>	force later arguments to be bound by name.
<code>tempNameGenerator</code>	temp name generator produced by <code>cdata::makeTempNameGenerator</code> , used to record <code>dplyr::compute()</code> effects.

### Value

table with new column added.

### Examples

```
d <- data.frame(x= c(1:3))
addConstantColumn(d, 'newCol', 'newVal')
```

---

buildJoinPlan	<i>Build a join plan</i>
---------------	--------------------------

---

## Description

Please see `vignette('DependencySorting', package = 'replyr')` and `vignette('joinController', package = 'replyr')` for more details.

## Usage

```
buildJoinPlan(tDesc, ..., check = TRUE)
```

## Arguments

tDesc	description of tables from <a href="#">tableDescription</a> (and likely altered by user). Note: no column names must intersect with names of the form <code>table_CLEANEDTABNAME_present</code> .
...	force later arguments to bind by name.
check	logical, if TRUE check the join plan for consistency.

## Value

detailed column join plan (appropriate for editing)

## See Also

[tableDescription](#), [inspectDescrAndJoinPlan](#), [makeJoinDiagramSpec](#), [executeLeftJoinPlan](#)

## Examples

```
d <- data.frame(id=1:3, weight= c(200, 140, 98))
tDesc <- rbind(tableDescription('d1', d),
              tableDescription('d2', d))
tDesc$keys[[1]] <- list(PrimaryKey= 'id')
tDesc$keys[[2]] <- list(PrimaryKey= 'id')
buildJoinPlan(tDesc)
```

---

dplyr\_src\_to\_db\_handle

*get the db handle from a dplyr src*


---

**Description**

Spark2 handles are DBIConnection SQLite are not this distinction is going away post dplyr 0.5.0

**Usage**

```
dplyr_src_to_db_handle(dplyr_src)
```

**Arguments**

dplyr\_src      remote data handle

**Value**

database connection

**Examples**

```
my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
# my_db <- sparklyr::spark_connect(master = "local")
d <- replayr_copy_to(my_db, data.frame(x=c(1,2)), 'd',
  overwrite=TRUE, temporary=TRUE)
dplyr_src_to_db_handle(replayr_get_src('d'))
```

---

executeLeftJoinPlan      *Execute an ordered sequence of left joins.*


---

**Description**

Please see vignette('DependencySorting', package = 'replayr') and vignette('joinController', package = 'replayr') for more details.

**Usage**

```
executeLeftJoinPlan(tDesc, columnJoinPlan, ..., checkColumns = FALSE,
  computeFn = function(x, name) { dplyr::compute(x, name = name) },
  eagerCompute = TRUE, checkColClasses = FALSE, verbose = FALSE,
  dryRun = FALSE,
  tempNameGenerator = makeTempNameGenerator("executeLeftJoinPlan"))
```

**Arguments**

tDesc	description of tables, either a data.frame from <a href="#">tableDescription</a> , or a list mapping from names to handles/frames. Only used to map table names to data.
columnJoinPlan	columns to join, from <a href="#">buildJoinPlan</a> (and likely altered by user). Note: no column names must intersect with names of the form table_CLEANEDTABNAME_present.
...	force later arguments to bind by name.
checkColumns	logical if TRUE confirm column names before starting joins.
computeFn	function to call to try and materialize intermediate results.
eagerCompute	logical if TRUE materialize intermediate results with computeFn.
checkColClasses	logical if true check for exact class name matches
verbose	logical if TRUE print more.
dryRun	logical if TRUE do not perform joins, only print steps.
tempNameGenerator	temp name generator produced by <code>cdata::makeTempNameGenerator</code> , used to record <code>dplyr::compute()</code> effects.

**Value**

joined table

**See Also**

[tableDescription](#), [buildJoinPlan](#), [inspectDescrAndJoinPlan](#), [makeJoinDiagramSpec](#)

TODO: parameterize the implementation provider (right now hard-coded for `dplyr`, but at least also direct SQL is a good extension).

**Examples**

```
# example data
meas1 <- data.frame(id= c(1,2),
                    weight= c(200, 120),
                    height= c(60, 14))
meas2 <- data.frame(pid= c(2,3),
                    weight= c(105, 110),
                    width= 1)

# get the initial description of table defs
tDesc <- rbind(tableDescription('meas1', meas1),
               tableDescription('meas2', meas2))

# declare keys (and give them consistent names)
tDesc$keys[[1]] <- list(PatientID= 'id')
tDesc$keys[[2]] <- list(PatientID= 'pid')

# build the column join plan
columnJoinPlan <- buildJoinPlan(tDesc)

# decide we don't want the width column
```

```

columnJoinPlan$want[columnJoinPlan$resultColumn=='width'] <- FALSE
# double check our plan
if(!is.null(inspectDescrAndJoinPlan(tDesc, columnJoinPlan,
  checkColClasses= TRUE))) {
  stop("bad join plan")
}
# execute the left joins
executeLeftJoinPlan(tDesc, columnJoinPlan,
  checkColClasses= TRUE,
  verbose= TRUE)

# also good
executeLeftJoinPlan(list('meas1'=meas1, 'meas2'=meas2),
  columnJoinPlan,
  checkColClasses= TRUE,
  verbose= TRUE)

```

---

expandColumn

*Expand a column of vectors into one row per value of each vector.*


---

### Description

Similar to `tidyr::unnest` but lands rowids and value ids, and can work on remote data sources. Fairly expensive per-row operation, not suitable for big data.

### Usage

```

expandColumn(data, colName, ..., rowidSource = NULL, rowidDest = NULL,
  idxDest = NULL,
  tempNameGenerator = makeTempNameGenerator("replay_expandColumn"))

```

### Arguments

<code>data</code>	data.frame to work with.
<code>colName</code>	character name of column to expand.
<code>...</code>	force later arguments to be bound by name
<code>rowidSource</code>	optional character name of column to take row indices from (rowidDest must be NULL to use this).
<code>rowidDest</code>	optional character name of column to write row indices to (must not be an existing column name, rowidSource must be NULL to use this).
<code>idxDest</code>	optional character name of column to write value indices to (must not be an existing column name).
<code>tempNameGenerator</code>	temp name generator produced by <code>cdata::makeTempNameGenerator</code> , used to record <code>dplyr::compute()</code> effects.

**Value**

expanded data frame where each value of colName column is in a new row.

**Examples**

```
d <- data.frame(name= c('a','b'))
d$value <- list(c('x','y'),'z')
expandColumn(d, 'value',
             rowidDest= 'origRowId',
             idxDest= 'valueIndex')
```

---

gapply

*grouped ordered apply*


---

**Description**

Partitions from by values in grouping column, applies a generic transform to each group and then binds the groups back together. Only advised for a moderate number of groups and better if grouping column is an index. This is powerfull enough to implement "The Split-Apply-Combine Strategy for Data Analysis" <https://www.jstatsoft.org/article/view/v040i01>

**Usage**

```
gapply(df, gcolumn, f, ..., ocolumn = NULL, decreasing = FALSE,
       partitionMethod = "split", bindrows = TRUE, maxgroups = 100,
       eagerCompute = FALSE, restoreGroup = FALSE,
       tempNameGenerator = makeTempNameGenerator("replyr_gapply"))
```

**Arguments**

df	remote dplyr data item
gcolumn	grouping column
f	transform function or pipeline
...	force later values to be bound by name
ocolumn	ordering column (optional)
decreasing	logical, if TRUE sort in decreasing order by ocolumn
partitionMethod	method to partition the data, one of 'group_by' (depends on f being dplyr compatible), 'split' (only works over local data frames), or 'extract'
bindrows	logical, if TRUE bind the rows back into a data item, else return split list
maxgroups	maximum number of groups to work over (intentionally not enforced if partitionMethod=='group_by')



**eagerCompute** logical, if TRUE call compute on split results  
**restoreGroup** logical, if TRUE restore group column after apply when partitionMethod %in% c('extract', 'split')  
**tempNameGenerator**  
 temp name generator produced by cdata::makeTempNameGenerator, used to record dplyr::compute() effects.

## Details

Note this is a fairly expensive operator, so it only makes sense to use in situations where f itself is fairly complicated and/or expensive.

## Value

transformed frame

## Examples

```

d <- data.frame(
  group = c(1, 1, 2, 2, 2),
  order = c(.1, .2, .3, .4, .5),
  values = c(10, 20, 2, 4, 8)
)

# User supplied window functions. They depend on known column names and
# the data back-end matching function names (as cumsum).
cumulative_sum <- function(d) {
  dplyr::mutate(d, cv = cumsum(values))
}
rank_in_group <- function(d) {
  d %>%
  dplyr::mutate(., constcol = 1) %>%
  dplyr::mutate(., rank = cumsum(constcol)) %>%
  dplyr::select(., -constcol)
}

for (partitionMethod in c('group_by', 'split', 'extract')) {
  print(partitionMethod)
  print('cumulative sum example')
  print(
    gapply(
      d,
      'group',
      cumulative_sum,
      ocolumn = 'order',
      partitionMethod = partitionMethod
    )
  )
  print('ranking example')
  print(
    gapply(
      d,

```

```

        'group',
        rank_in_group,
        ocolumn = 'order',
        partitionMethod = partitionMethod
    )
)
print('ranking example (decreasing)')
print(
    gapply(
        d,
        'group',
        rank_in_group,
        ocolumn = 'order',
        decreasing = TRUE,
        partitionMethod = partitionMethod
    )
)
}

```

---

inspectDescrAndJoinPlan

*check that a join plan is consistent with table descriptions*

---

### Description

Please see vignette('DependencySorting', package = 'replry') and vignette('joinController', package = 'replry') for more details.

### Usage

```
inspectDescrAndJoinPlan(tDesc, columnJoinPlan, ..., checkColClasses = FALSE)
```

### Arguments

tDesc	description of tables, from <a href="#">tableDescription</a> (and likely altered by user).
columnJoinPlan	columns to join, from <a href="#">buildJoinPlan</a> (and likely altered by user). Note: no column names must intersect with names of the form table_CLEANEDTABNAME_present.
...	force later arguments to bind by name.
checkColClasses	logical if true check for exact class name matches

### Value

NULL if okay, else a string

### See Also

[tableDescription](#), [buildJoinPlan](#), [makeJoinDiagramSpec](#), [executeLeftJoinPlan](#)

## Examples

```
# example data
d1 <- data.frame(id= 1:3,
                 weight= c(200, 140, 98),
                 height= c(60, 24, 12))
d2 <- data.frame(pid= 2:3,
                 weight= c(130, 110),
                 width= 1)
# get the initial description of table defs
tDesc <- rbind(tableDescription('d1', d1),
               tableDescription('d2', d2))
# declare keys (and give them consistent names)
tDesc$keys[[1]] <- list(PrimaryKey= 'id')
tDesc$keys[[2]] <- list(PrimaryKey= 'pid')
# build the join plan
columnJoinPlan <- buildJoinPlan(tDesc)
# confirm the plan
inspectDescrAndJoinPlan(tDesc, columnJoinPlan,
                        checkColClasses= TRUE)
# damage the plan
columnJoinPlan$sourceColumn[columnJoinPlan$sourceColumn=='width'] <- 'wd'
# find a problem
inspectDescrAndJoinPlan(tDesc, columnJoinPlan,
                        checkColClasses= TRUE)
```

---

keysAreUnique

*Check uniqueness of rows with respect to keys.*

---

## Description

Can be an expensive operation.

## Usage

```
keysAreUnique(tDesc)
```

## Arguments

tDesc                    description of tables, from [tableDescription](#) (and likely altered by user).

## Value

logical TRUE if keys are unique

## See Also

[tableDescription](#)

## Examples

```
d <- data.frame(x=c(1,1,2,2,3,3), y=c(1,2,1,2,1,2))
tDesc1 <- tableDescription('d1', d)
tDesc2 <- tableDescription('d2', d)
tDesc <- rbind(tDesc1, tDesc2)
tDesc$keys[[2]] <- c(x='x')
keysAreUnique(tDesc)
```

---

key\_inspector\_all\_cols

*Return all columns as guess at preferred primary keys.*

---

## Description

Return all columns as guess at preferred primary keys.

## Usage

```
key_inspector_all_cols(handle)
```

## Arguments

handle            data handle

## Value

map of keys to keys

## See Also

tableDescription

## Examples

```
d <- data.frame(x=1:3, y=NA)
key_inspector_all_cols(d)
```

---

`key_inspector_postgresql`

*Return all primary key columns as guess at preferred primary keys for a PostgreSQL handle.*

---

**Description**

Return all primary key columns as guess at preferred primary keys for a PostgreSQL handle.

**Usage**

```
key_inspector_postgresql(handle)
```

**Arguments**

handle            data handle

**Value**

map of keys to keys

**See Also**

tableDescription

---

`key_inspector_sqlite`    *Return all primary key columns as guess at preferred primary keys for a SQLite handle.*

---

**Description**

Return all primary key columns as guess at preferred primary keys for a SQLite handle.

**Usage**

```
key_inspector_sqlite(handle)
```

**Arguments**

handle            data handle

**Value**

map of keys to keys

**See Also**

tableDescription

---

makeJoinDiagramSpec     *Build a drawable specification of the join diagram*

---

### Description

Please see vignette('DependencySorting', package = 'replyr') and vignette('joinController', package = 'replyr') for more details.

### Usage

```
makeJoinDiagramSpec(columnJoinPlan, ..., groupByKey = TRUE,
  graphOpts = NULL)
```

### Arguments

```
columnJoinPlan  join plan
...             force later arguments to bind by name
groupByKey      logical if true build key-equivalent sub-graphs
graphOpts       options for graphViz
```

### Value

grViz diagram spec

### See Also

[tableDescription](#), [buildJoinPlan](#), [renderJoinDiagram](#), [executeLeftJoinPlan](#)

### Examples

```
# note: employeeanddate is likely built as a cross-product
#       join of an employee table and set of dates of interest
#       before getting to the join controller step. We call
#       such a table "row control" or "experimental design."

my_db <- dplyr::src_sqlite("memory:",
  create = TRUE)
tDesc <- replyr::example_employeeAndDate(my_db)
# fix order by hand, please see replyr::topoSortTables for
# how to automate this.
ord <- match(c('employeeanddate', 'orgtable', 'activity', 'revenue'),
  tDesc$tableName)
tDesc <- tDesc[ord, , drop=FALSE]
columnJoinPlan <- buildJoinPlan(tDesc, check= FALSE)
# unify keys
columnJoinPlan$resultColumn[columnJoinPlan$resultColumn=='id'] <- 'eid'
```

```

# look at plan defects
print(paste('problems:',
           inspectDescrAndJoinPlan(tDesc, columnJoinPlan)))
diagramSpec <- makeJoinDiagramSpec(columnJoinPlan)
# to render as JavaScript:
# DiagrammeR::grViz(diagramSpec)
# or as a PNG:
# renderJoinDiagram(diagramSpec)
#
DBI::dbDisconnect(my_db$con)
my_db <- NULL

```

---

renderJoinDiagram	<i>Render a diagram spec from <a href="#">makeJoinDiagramSpec</a> as a PNG graphics item.</i>
-------------------	---

---

### Description

Requires packages DiagrammeR properly installed to use. Please see `vignette('DependencySorting', package = 'replayr')` and `vignette('joinController', package = 'replayr')` for more details.

### Usage

```
renderJoinDiagram(diagramSpec, ..., pngFileName = NULL, tempDir = tempdir())
```

### Arguments

diagramSpec	diagram specification from <a href="#">makeJoinDiagramSpec</a> .
...	force later arguments to bind by name.
pngFileName	character, if not null where to write PNG
tempDir	character, if not null tempDir to create/use

### Details

This PNG can be smaller than directly including a `grViz` rendering in markdown. Requires all of DiagrammeR, htmlwidgets, webshot, and magick to be installed with all external dependencies properly installed and configured.

### Value

DiagrammeR::grViz result as a PNG

### See Also

[tableDescription](#), [buildJoinPlan](#), [makeJoinDiagramSpec](#), [executeLeftJoinPlan](#)

---

replyr	<i>replyr: Diligent Use of Big Data for R</i>
--------	---

---

### Description

Methods to reliably use 'dplyr' remote data sources in R ('SQL' databases, 'Spark' 2.0.0 and above) in a generic fashion. REmote PLYing of big data for R. Adds convenience functions to make big data tasks more like working with an in-memory R 'data.frame'. Results do depend on which 'dplyr' data service provider used.

### Details

replyr helps with the following:

- Summarizing remote data (via `replyr_summarize`).
- Facilitating writing "source generic" code that works similarly on multiple 'dplyr' data sources.
- Providing big data versions of functions for splitting data, binding rows, pivoting, adding row-ids, ranking, and completing experimental designs.
- Packaging common data manipulation tasks into operators such as the [gapply](#) function.
- Providing support code for common SparklyR tasks, such as tracking temporary handle IDs.

To learn more about replyr, please start with the vignette: `vignette('replyr', 'replyr')`

---

replyr_add_ids	<i>Add unique ids to rows. Note: re-arranges rows in many cases.</i>
----------------	--

---

### Description

Add unique ids to rows. Note: re-arranges rows in many cases.

### Usage

```
replyr_add_ids(df, idColName)
```

### Arguments

df	data.frame object to work with
idColName	name of column to add

### Examples

```
replyr_add_ids(data.frame(x=c('a','b')), 'id')
```



---

replyr\_apply\_f\_mapped *Apply a function to a re-mapped data frame.*

---

## Description

Apply a function to a re-mapped data frame.

## Usage

```
replyr_apply_f_mapped(d, f, nmap, ..., restrictMapIn = FALSE,
  rmap = replyr::replyr_reverseMap(nmap), restrictMapOut = FALSE)
```

## Arguments

d	data.frame to work on
f	function to apply.
nmap	named list mapping with keys specifying new column names, and values as original column names.
...	force later arguments to bind by name
restrictMapIn	logical if TRUE restrict columns when mapping in.
rmap	reverse map (for after f is applied).
restrictMapOut	logical if TRUE restrict columns when mapping out.

## See Also

[let](#), [replyr\\_reverseMap](#), [replyr\\_mapRestrictCols](#)

## Examples

```
# an external function with hard-coded column names
DecreaseRankColumnByOne <- function(d) {
  d$RankColumn <- d$RankColumn - 1
  d
}

# our example data, with different column names
d <- data.frame(Sepal_Length=c(5.8,5.7),
  Sepal_Width=c(4.0,4.4),
  Species='setosa',rank=c(1,2))
print(d)

# map our data to expected column names so we can use function
nmap <- c(GroupColumn='Species',
  ValueColumn='Sepal_Length',
  RankColumn='rank')
```

```
print(nmap)

dF <- replyr_apply_f_mapped(d, DecreaseRankColumnByOne, nmap)
print(dF)
```

---

replyr\_arrange      *arrange by a single column*

---

### Description

arrange by a single column

### Usage

```
replyr_arrange(.data, colname, descending = FALSE)
```

### Arguments

.data	data object to work on
colname	character column name
descending	logical if true sort descending (else sort ascending)

### Examples

```
d <- data.frame(Sepal_Length= c(5.8,5.7),
               Sepal_Width= c(4.0,4.4))
replyr_arrange(d, 'Sepal_Length', descending= TRUE)
```

---

replyr\_bind\_rows      *Bind a list of items by rows (can't use dplyr::bind\_rows or dplyr::combine on remote sources). Columns are intersected.*

---

### Description

Can't set eagerTempRemoval=TRUE on platforms that don't correctly implement `dplyr::compute` (for instance Sparklyr prior to full resolution of <https://github.com/rstudio/sparklyr/issues/721>).

**Usage**

```
replyr_bind_rows(lst, ..., useDplyrLocal = TRUE, useSparkRbind = TRUE,
  useUnionALL = TRUE, eagerTempRemoval = FALSE,
  tempNameGenerator = makeTempNameGenerator("replyr_bind_rows"))
```

**Arguments**

```
lst          list of items to combine, must be all in same dplyr data service
...         force other arguments to be used by name
useDplyrLocal logical if TRUE use dplyr for local data.
useSparkRbind logical if TRUE try to use rbind on Sparklyr data
useUnionALL  logical if TRUE try to use union all binding
eagerTempRemoval
              logical if TRUE remove temps early.
tempNameGenerator
              temp name generator produced by cdata::makeTempNameGenerator, used to
              record dplyr::compute() effects.
```

**Value**

single data item

**Examples**

```
my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
# my_db <- sparklyr::spark_connect(master = "local")
d <- replyr_copy_to(my_db, data.frame(x = 1:2), 'd',
  temporary = TRUE)
# dplyr::bind_rows(list(d, d))
# # Argument 1 must be a data frame or a named atomic vector, not a tbl_dbi/tbl_sql/tbl_lazy/tbl
replyr_bind_rows(list(d, d))
```

---

```
replyr_check_ranks    confirm data has good ranked groups
```

---

**Description**

confirm data has good ranked groups

**Usage**

```
replyr_check_ranks(x, GroupColumnName, ValueColumnName, RankColumnName, ...,
  decreasing = FALSE,
  tempNameGenerator = makeTempNameGenerator("replyr_check_ranks"))
```

**Arguments**

**x** data item to work with  
**GroupColumnName** column to group by  
**ValueColumnName** column determining order  
**RankColumnName** column having proposed rank (function of order)  
**...** force later arguments to bind by name  
**decreasing** if true make order decreasing instead of increasing.  
**tempNameGenerator** temp name generator produced by `cdata::makeTempNameGenerator`, used to record `dplyr::compute()` effects.

**Value**

summary of quality of ranking.

**Examples**

```

d <- data.frame(Sepal_Length=c(5.8,5.7),Sepal_Width=c(4.0,4.4),
                Species='setosa',rank=c(1,2))
replayr_check_ranks(d, 'Species', 'Sepal_Length', 'rank', decreasing=TRUE)
  
```

---

`replayr_coalesce`      *Augment a data frame by adding additional rows.*

---

**Description**

Note: do not count on order of resulting data. Also only added rows are altered by the fill instructions.

**Usage**

```

replayr_coalesce(data, support, ..., fills = NULL, newRowColumn = NULL,
                 copy = TRUE, tempNameGenerator = makeTempNameGenerator("replayr_coalesce"))
  
```

**Arguments**

**data** data.frame data to augment  
**support** data.frame rows of unique key-values into data  
**...** not used, force later arguments to bind by name  
**fills** list default values to fill in columns  
**newRowColumn** character if not null name to use for new row indicator

copy                    logical if TRUE copy support to data's source  
tempNameGenerator                    temp name generator produced by `cdata::makeTempNameGenerator`, used to record `dplyr::compute()` effects.

**Value**

augmented data

**Examples**

```
# single column key example
data <- data.frame(year = c(2005,2007,2010),
                   count = c(6,1,NA),
                   name = c('a', 'b', 'c'),
                   stringsAsFactors = FALSE)
support <- data.frame(year=2005:2010)
filled <- replayr_coalesce(data, support,
                          fills=list(count=0))
filled <- filled[order(filled$year), ]
filled

# complex key example
data <- data.frame(year = c(2005,2007,2010),
                   count = c(6,1,NA),
                   name = c('a', 'b', 'c'),
                   stringsAsFactors = FALSE)
support <- expand.grid(year=2005:2010,
                      name= c('a', 'b', 'c', 'd'),
                      stringsAsFactors = FALSE)
filled <- replayr_coalesce(data, support,
                          fills=list(count=0))
filled <- filled[order(filled$year, filled$name), ]
filled
```

---

replayr\_colClasses                    *Get column classes.*

---

**Description**

Get column classes.

**Usage**

```
replayr_colClasses(x)
```

**Arguments**

x                      tbl or item that can be coerced into such.

**Value**

list of column classes.

**Examples**

```
d <- data.frame(x=c(1,2))
replyr_colClasses(d)
```

---

replyr\_copy\_from            *Bring remote data back as a local data frame tbl.*

---

**Description**

Bring remote data back as a local data frame tbl.

**Usage**

```
replyr_copy_from(d, maxrow = 1e+06)
```

**Arguments**

d                      remote dplyr data item  
maxrow                max rows to allow (stop otherwise, set to NULL to allow any size).

**Value**

local tbl.

**Examples**

```
if (requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- dplyr::src_sqlite(":memory:", create = TRUE)
  d <- replyr_copy_to(my_db, data.frame(x=c(1,2)), 'd')
  d2 <- replyr_copy_from(d)
  print(d2)
}
```

---

replyr_copy_to	<i>Copy data to remote service.</i>
----------------	-------------------------------------

---

**Description**

Copy data to remote service.

**Usage**

```
replyr_copy_to(dest, df, name = paste(deparse(substitute(df)), collapse =
  " "), ..., rowNumberColumn = NULL, temporary = FALSE, overwrite = TRUE,
  maxrow = 1e+06, forceDelete = FALSE)
```

**Arguments**

dest	remote data source
df	local data frame
name	name for new remote table
...	force later values to be bound by name
rowNumberColumn	if not null name to add row numbers to
temporary	logical, if TRUE try to create a temporary table
overwrite	logical, if TRUE try to overwrite
maxrow	max rows to allow in a remote to remote copy.
forceDelete	logical, if TRUE try to delete table.

**Value**

remote handle

**Examples**

```
if (requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- dplyr::src_sqlite(":memory:", create = TRUE)
  d <- replyr_copy_to(my_db, data.frame(x=c(1,2)), 'd')
  print(d)
}
```

---

replyr_dim	<i>Compute dimensions of a data.frame (work around <a href="https://github.com/rstudio/sparklyr/issues/976">https://github.com/rstudio/sparklyr/issues/976</a> ).</i>
------------	---

---

**Description**

Compute dimensions of a data.frame (work around <https://github.com/rstudio/sparklyr/issues/976> ).

**Usage**

```
replyr_dim(x)
```

**Arguments**

x                   tbl or item that can be coerced into such.

**Value**

dimensions (including rows)

**Examples**

```
d <- data.frame(x=c(1,2))
replyr_dim(d)
```

---

replyr_drop_table_name	<i>Drop a table from a source</i>
------------------------	-----------------------------------

---

**Description**

Drop a table from a source

**Usage**

```
replyr_drop_table_name(dest, name)
```

**Arguments**

dest               remote data source  
name               name of table to drop



**Value**

logical TRUE if table was present

**Examples**

```
if (requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- dplyr::src_sqlite(":memory:", create = TRUE)
  d <- replyr_copy_to(my_db, data.frame(x=c(1,2)), 'd')
  print(d)
  replyr_list_tables(my_db)
  replyr_drop_table_name(my_db, 'd')
  replyr_list_tables(my_db)
}
```

---

replayr_filter	<i>Filter a tbl on a column having values in a given set.</i>
----------------	---

---

**Description**

Filter a tbl on a column having values in a given set.

**Usage**

```
replayr_filter(x, cname, values, ..., verbose = TRUE,
  tempNameGenerator = makeTempNameGenerator("replayr_filter"))
```

**Arguments**

x	tbl or item that can be coerced into such.
cname	name of the column to test values of.
values	set of values to check set membership of.
...	force later arguments to bind by name.
verbose	logical if TRUE echo warnings
tempNameGenerator	temp name generator produced by cdata::makeTempNameGenerator, used to record dplyr::compute() effects.

**Value**

new tbl with only rows where cname value is in values set.

**Examples**

```
values <- c('a', 'c')
d <- data.frame(x=c('a', 'a', 'b', 'b', 'c', 'c'), y=1:6,
               stringsAsFactors=FALSE)
replyr_filter(d, 'x', values)
```

---

replyr_get_src	<i>Get the "remote data source" where a data.frame like object lives.</i>
----------------	---

---

**Description**

Get the "remote data source" where a data.frame like object lives.

**Usage**

```
replyr_get_src(df)
```

**Arguments**

df                    data.frame style object

**Value**

source (string if data.frame, tlb, or data.table, NULL if unknown, remote source otherwise)

**Examples**

```
replyr_get_src(data.frame(x=1:2))
```

---

replyr_group_by	<i>group_by columns</i>
-----------------	-------------------------

---

**Description**

See also: <https://gist.github.com/skranz/9681509>

**Usage**

```
replyr_group_by(.data, colnames)
```

**Arguments**

.data            data object to work on  
colnames        character column name (can be a vector)

**Examples**

```
d <- data.frame(Sepal_Length= c(5.8,5.7),  
               Sepal_Width= c(4.0,4.4),  
               Species= 'setosa')  
replyr_group_by(d, 'Species')
```

---

replyr\_hasrows            *Check if a table has rows.*

---

**Description**

Check if a table has rows.

**Usage**

```
replyr_hasrows(d)
```

**Arguments**

d                tbl or item that can be coerced into such.

**Value**

number of rows

**Examples**

```
d <- data.frame(x=c(1,2))  
replyr_hasrows(d)
```

---

replyr_has_table	<i>check for table</i>
------------------	------------------------

---

### Description

Work around connection v.s. handle issues <https://github.com/tidyverse/dplyr/issues/2849>

### Usage

```
replyr_has_table(con, name)
```

### Arguments

con	connection
name	character name to check for

### Value

TRUE if table present

### Examples

```
if (requireNamespace("RSQLite", quietly = TRUE)) {
  my_db <- dplyr::src_sqlite(":memory:", create = TRUE)
  d <- replyr_copy_to(my_db, data.frame(x=c(1,2)), 'd')
  print(d)
  replyr_has_table(my_db, 'd')
}
```

---

replyr_inTest	<i>Product a column noting if another columns values are in a given set.</i>
---------------	--

---

### Description

Product a column noting if another columns values are in a given set.

### Usage

```
replyr_inTest(x, cname, values, nname, ...,
  tempNameGenerator = makeTempNameGenerator("replyr_inTest"),
  verbose = TRUE)
```

**Arguments**

x	tbl or item that can be coerced into such.
cname	name of the column to test values of.
values	set of values to check set membership of.
nname	name for new column
...	force later parameters to bind by name
tempNameGenerator	temp name generator produced by cdata::makeTempNameGenerator, used to record dplyr::compute() effects.
verbose	logical if TRUE echo warnings

**Value**

table with membership indications.

**Examples**

```
values <- c('a', 'c')
d <- data.frame(x=c('a', 'a', 'b', NA, 'c', 'c'), y=1:6,
               stringsAsFactors=FALSE)
replyr_inTest(d, 'x', values, 'match')
```

---

replyr\_is\_local\_data *Test if data is local.*

---

**Description**

Test if data is local.

**Usage**

```
replyr_is_local_data(d)
```

**Arguments**

d data frame

**Value**

TRUE if local data (data.frame, tbl/tibble)

**Examples**

```
replyr_is_local_data(data.frame(x=1:3))
```

---

replyr\_is\_MySQL\_data *Test if data is MySQL.*

---

**Description**

Test if data is MySQL.

**Usage**

```
replyr_is_MySQL_data(d)
```

**Arguments**

d                    data frame

**Value**

TRUE if Spark data

**Examples**

```
replyr_is_MySQL_data(data.frame(x=1:3))
```

---

replyr\_is\_Spark\_data *Test if data is Spark.*

---

**Description**

Test if data is Spark.

**Usage**

```
replyr_is_Spark_data(d)
```

**Arguments**

d                    data frame

**Value**

TRUE if Spark data

**Examples**

```
replyr_is_Spark_data(data.frame(x=1:3))
```

---

replyr\_list\_tables     *list tables*

---

**Description**

Work around connection v.s. handle issues <https://github.com/tidyverse/dplyr/issues/2849>

**Usage**

```
replyr_list_tables(con)
```

**Arguments**

con                    connection

**Value**

list of tables names

**Examples**

```
my_db <- dplyr::src_sqlite(":memory:", create = TRUE)
d <- replyr_copy_to(my_db, data.frame(x=c(1,2)), 'd',
  overwrite=TRUE, temporary=TRUE)
print(d)
replyr_list_tables(my_db)
```

---

replyr\_mapRestrictCols

*Map names of columns to known values and drop other columns.*

---

**Description**

Restrict a data item's column names and re-name them in bulk.

**Usage**

```
replyr_mapRestrictCols(x, nmap, ..., restrict = FALSE, reverse = FALSE)
```

**Arguments**

x	data item to work on
nmap	named list mapping with keys specifying new column names, and values as original column names.
...	force later arguments to bind by name
restrict	logical if TRUE restrict to columns mentioned in nmap.
reverse	logical if TRUE apply the inverse of nmap instead of nmap.

**Details**

Something like `replay::replay_mapRestrictCols` is only useful to get control of a function that is not parameterized (in the sense it has hard-coded column names inside its implementation that don't match column names in our data).

**Value**

data item with columns renamed (and possibly restricted).

**See Also**

[let](#), [replay\\_reverseMap](#), [replay\\_apply\\_f\\_mapped](#)

**Examples**

```
# an external function with hard-coded column names
DecreaseRankColumnByOne <- function(d) {
  d$RankColumn <- d$RankColumn - 1
  d
}

# our example data, with different column names
d <- data.frame(Sepal_Length=c(5.8,5.7),
               Sepal_Width=c(4.0,4.4),
               Species='setosa',rank=c(1,2))
print(d)

# map our data to expected column names so we can use function
nmap <- c(GroupColumn='Species',
          ValueColumn='Sepal_Length',
          RankColumn='rank')
print(nmap)
dm <- replay_mapRestrictCols(d,nmap)
print(dm)

# can now apply code that expects hard-coded names.
dm <- DecreaseRankColumnByOne(dm)

# map back to our original column names (for the columns we retained)
```



```
# Note: can only map back columns that were retained in first mapping.
replayr_mapRestrictCols(dm, nmap, reverse=TRUE)
```

---

replayr_ncol	<i>Compute number of columns of a data.frame (work around <a href="https://github.com/rstudio/sparklyr/issues/976">https://github.com/rstudio/sparklyr/issues/976</a> ).</i>
--------------	--

---

### Description

Compute number of columns of a data.frame (work around <https://github.com/rstudio/sparklyr/issues/976> ).

### Usage

```
replayr_ncol(x)
```

### Arguments

x                      tbl or item that can be coerced into such.

### Value

number of columns

### Examples

```
d <- data.frame(x=c(1,2))
replayr_ncol(d)
```

---

replayr_nrow	<i>Compute number of rows of a tbl.</i>
--------------	---

---

### Description

Number of row in a table. This function is not "group aware" it returns the total number of rows, not rows per dplyr group. Also replayr\_nrow depends on data being returned to count, so some corner cases (such as zero columns) will count as zero rows. In particular work around dplyr issue 2871 <https://github.com/tidyverse/dplyr/issues/2871>.

### Usage

```
replayr_nrow(x)
```

**Arguments**

x                   tbl or item that can be coerced into such.

**Value**

number of rows

**Examples**

```
d <- data.frame(x=c(1,2))
replyr_nrow(d)
```

---

replyr_quantile	<i>Compute quantiles on remote column (NA's filtered out) using binary search.</i>
-----------------	--

---

**Description**

NA's filtered out and does not break ties the same as stats::quantile.

**Usage**

```
replyr_quantile(x, cname, probs = seq(0, 1, 0.25), ...,
  tempNameGenerator = makeTempNameGenerator("replyr_quantile"))
```

**Arguments**

x                   tbl or item that can be coerced into such.

cname               column name to compute over

probs               numeric vector of probabilities with values in [0,1].

...                 force later arguments to be bound by name.

tempNameGenerator  
temp name generator produced by cdata::makeTempNameGenerator, used to record dplyr::compute() effects.

**Examples**

```
d <- data.frame(xvals=rev(1:1000))
replyr_quantile(d, 'xvals')
```

---

replyr_quantilec	<i>Compute quantiles on remote column (NA's filtered out) using cum-sum.</i>
------------------	--

---

**Description**

NA's filtered out and does not break ties the same as stats::quantile.

**Usage**

```
replyr_quantilec(x, cname, probs = seq(0, 1, 0.25), ...,
  tempNameGenerator = makeTempNameGenerator("replyr_quantilec"))
```

**Arguments**

x	tbl or item that can be coerced into such.
cname	column name to compute over (not 'n' or 'csum')
probs	numeric vector of probabilities with values in [0,1].
...	force later arguments to bind by name.
tempNameGenerator	temp name generator produced by cdata::makeTempNameGenerator, used to record dplyr::compute() effects.

**Examples**

```
d <- data.frame(xvals=rev(1:1000))
replyr_quantilec(d, 'xvals')
```

---

replyr_rename	<i>Rename a column</i>
---------------	------------------------

---

**Description**

Rename a column

**Usage**

```
replyr_rename(.data, ..., newName, oldName)
```

**Arguments**

.data	data object to work on
...	force later arguments to bind by name
newName	character new column name
oldName	character old column name

**Examples**

```
d <- data.frame(Sepal_Length= c(5.8,5.7),
                Sepal_Width= c(4.0,4.4),
                Species= 'setosa', rank=c(1,2))
replyr_rename(d, newName = 'family', oldName = 'Species')
```

---

replyr_reverseMap	<i>Reverse a name assignment map (which are written NEW-NAME=OLDNAME).</i>
-------------------	--

---

**Description**

Reverse a name assignment map (which are written NEWNAME=OLDNAME).

**Usage**

```
replyr_reverseMap(nmap)
```

**Arguments**

nmap	named list mapping with keys specifying new column names, and values as original column names.
------	--

**Value**

inverse map

**See Also**

[let](#), [replyr\\_apply\\_f\\_mapped](#), [replyr\\_mapRestrictCols](#)

**Examples**

```
mp <- c(A='x', B='y')
print(mp)
replyr_reverseMap(mp)
```

---

replyr_select	<i>select columns</i>
---------------	-----------------------

---

**Description**

select columns

**Usage**

```
replyr_select(.data, colnames)
```

**Arguments**

.data	data object to work on
colnames	character column names

**Examples**

```
d <- data.frame(Sepal_Length= c(5.8,5.7),
                Sepal_Width= c(4.0,4.4),
                Species= 'setosa', rank=c(1,2))
replyr_select(d, c('Sepal_Length', 'Species'))
```

---

replyr_split	<i>split a data item by values in a column.</i>
--------------	---

---

**Description**

Partitions from by values in grouping column, and returns list. Only advised for a moderate number of groups and better if grouping column is an index. This plus lapply and replyr::bind\_rows is powerful enough to implement "The Split-Apply-Combine Strategy for Data Analysis" <https://www.jstatsoft.org/article/view/v040>

**Usage**

```
replyr_split(df, gcolumn, ..., ocolumn = NULL, decreasing = FALSE,
             partitionMethod = "extract", maxgroups = 100, eagerCompute = FALSE)
```

**Arguments**

df	remote dplyr data item
gcolumn	grouping column
...	force later values to be bound by name
ocolumn	ordering column (optional)
decreasing	if TRUE sort in decreasing order by ocolumn
partitionMethod	method to partition the data, one of 'split' (only works over local data frames), or 'extract'
maxgroups	maximum number of groups to work over
eagerCompute	if TRUE call compute on split results

**Value**

list of data items

**Examples**

```
d <- data.frame(group=c(1,1,2,2,2),
                order=c(.1,.2,.3,.4,.5),
                values=c(10,20,2,4,8))
dSplit <- replyr_split(d, 'group', partitionMethod='extract')
dApp <- lapply(dSplit, function(di) data.frame(as.list(colMeans(di))))
replyr_bind_rows(dApp)
```

---

replyr_summary	<i>Compute usable summary of columns of tbl.</i>
----------------	--

---

**Description**

Compute per-column summaries and return as a data.frame. Warning: can be an expensive operation.

**Usage**

```
replyr_summary(x, ..., countUniqueNum = FALSE, countUniqueNonNum = FALSE,
              cols = NULL, compute = TRUE)
```

**Arguments**

**x** tbl or item that can be coerced into such.  
**...** force additional arguments to be bound by name.  
**countUniqueNum** logical, if true include unique non-NA counts for numeric cols.  
**countUniqueNonNum** logical, if true include unique non-NA counts for non-numeric cols.  
**cols** if not NULL set of columns to restrict to.  
**compute** logical if TRUE call compute before working

**Details**

Can be slow compared to `dplyr::summarize_all()` (but serves a different purpose). Also, for numeric columns includes NaN in nna count (as is typical for R, e.g., `is.na(NaN)`). And note: `replyr_summary()` currently skips "raw" columns.

**Value**

summary of columns.

**Examples**

```

d <- data.frame(p= c(TRUE, FALSE, NA),
               r= I(list(1,2,3)),
               s= NA,
               t= as.raw(3:5),
               w= 1:3,
               x= c(NA,2,3),
               y= factor(c(3,5,NA)),
               z= c('a',NA,'z'),
               stringsAsFactors=FALSE)
# sc <- sparklyr::spark_connect(version='2.2.0',
#                               master = "local")
# dS <- replyr_copy_to(sc, dplyr::select(d, -r, -t), 'dS',
#                     temporary=TRUE, overwrite=TRUE)
# replyr_summary(dS)
# my_db <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
# RSQLite::initExtension(my_db)
# dM <- replyr_copy_to(my_db, dplyr::select(d, -r, -t), 'dM',
#                     temporary=TRUE, overwrite=TRUE)
# replyr_summary(dM)
d$q <- list(1,2,3)
replyr_summary(d)

```

---

replayr\_testCols      *Run test on columns.*

---

### Description

Applies user function to head of each column. Good for determining things such as column class.

### Usage

```
replayr_testCols(x, f, n = 6L)
```

### Arguments

x                    tbl or item that can be coerced into such.  
f                    test function (returning logical, not depending on data length).  
n                    number of rows to use in calculation.

### Value

logical vector of results.

### Examples

```
d <- data.frame(x=c(1,2),y=c('a','b'))  
replayr_testCols(d,is.numeric)
```

---

replayr\_union\_all      *Union two tables.*

---

### Description

Spark 2\* union\_all has issues (<https://github.com/WinVector/replyr/blob/master/issues/UnionIssue.md>), and explosed union\_all semantics differ from data-source backend to backend. This is an attempt to provide a join-based replacement.

### Usage

```
replayr_union_all(tabA, tabB, ..., useDplyrLocal = TRUE,  
  useSparkRbind = TRUE,  
  tempNameGenerator = makeTempNameGenerator("replayr_union_all"))
```



**Arguments**

tabA	not-NULL table with at least 1 row.
tabB	not-NULL table with at least 1 row on same data source as tabA and common columns.
...	force later arguments to be bound by name.
useDplyrLocal	logical if TRUE use dplyr::bind_rows for local data.
useSparkRbind	logical if TRUE try to use rbind on Sparklyr data
tempNameGenerator	temp name generator produced by cdata::makeTempNameGenerator, used to record dplyr::compute() effects.

**Value**

table with all rows of tabA and tabB (union\_all).

**Examples**

```
d1 <- data.frame(x = c('a','b'), y = 1, stringsAsFactors= FALSE)
d2 <- data.frame(x = 'c', z = 1, stringsAsFactors= FALSE)
replayr_union_all(d1, d2, useDplyrLocal= FALSE)
```

---

replayr\_uniqueValues     *Compute number of unique values for each level in a column.*

---

**Description**

Compute number of unique values for each level in a column.

**Usage**

```
replayr_uniqueValues(x, cname)
```

**Arguments**

x	tbl or item that can be coerced into such.
cname	name of columns to examine, must not be equal to 'replayr_private_value_n'.

**Value**

unique values for the column.

**Examples**

```
d <- data.frame(x=c(1,2,3,3))
replayr_uniqueValues(d, 'x')
```

---

tableDescription	<i>Build a nice description of a table.</i>
------------------	---

---

### Description

Please see <http://www.win-vector.com/blog/2017/05/managing-spark-data-handles-in-r/> for details. Note: one usually needs to alter the keys column which is just populated with all columns.

### Usage

```
tableDescription(tableName, handle, ...,  
  keyInspector = key_inspector_all_cols)
```

### Arguments

tableName	name of table to add to join plan.
handle	table or table handle to add to join plan (can already be in the plan).
...	force later arguments to bind by name.
keyInspector	function that determines preferred primary key set for table.

### Details

Please see vignette('DependencySorting', package = 'replayr') and vignette('joinController', package = 'replayr') for more details.

### Value

table describing the data.

### See Also

[buildJoinPlan](#), [keysAreUnique](#), [makeJoinDiagramSpec](#), [executeLeftJoinPlan](#)

### Examples

```
d <- data.frame(x=1:3, y=NA)  
tableDescription('d', d)
```

---

topoSortTables	<i>Topologically sort join plan so values are available before uses.</i>
----------------	--

---

**Description**

Depends on igraph package. Please see vignette('DependencySorting', package = 'replyr') and vignette('joinController', package= 'replyr') for more details.

**Usage**

```
topoSortTables(columnJoinPlan, leftTableName, ...)
```

**Arguments**

```
columnJoinPlan  join plan
leftTableName   which table is left
...             force later arguments to bind by name
```

**Value**

list with dependencyGraph and sorted columnJoinPlan

**Examples**

```
# note: employeeanddate is likely built as a cross-product
#       join of an employee table and set of dates of interest
#       before getting to the join controller step. We call
#       such a table "row control" or "experimental design."

my_db <- dplyr::src_sqlite("memory:",
                          create = TRUE)
tDesc <- replyr::example_employeeAndDate(my_db)
columnJoinPlan <- buildJoinPlan(tDesc, check= FALSE)
# unify keys
columnJoinPlan$resultColumn[columnJoinPlan$resultColumn=='id'] <- 'eid'
# look at plan defects
print(paste('problems:',
            inspectDescrAndJoinPlan(tDesc, columnJoinPlan)))
# fix plan
if(requireNamespace('igraph', quietly = TRUE)) {
  sorted <- topoSortTables(columnJoinPlan, 'employeeanddate')
  print(paste('problems:',
              inspectDescrAndJoinPlan(tDesc, sorted$columnJoinPlan)))
  # plot(sorted$dependencyGraph)
}
DBI::dbDisconnect(my_db$con)
```

```
my_db <- NULL
```

---

%land%

*Land a value to variable from a pipeline.*

---

## Description

%land% and %->% ("writearrow") copy a pipeline value to a variable on the right hand side. %land\_% and %->\_% copy a pipeline value to a variable named by the value referenced by its right hand side argument.

## Usage

```
value %land% name
```

```
value %->% name
```

```
value %->_% name
```

```
value %land_% name
```

## Arguments

value	value to write
name	variable to write to

## Details

Technically these operators are not "-> assignment", so they might not be specifically prohibited in an oppugnant reading of some style guides.

## Value

value

## Examples

```
sin(7) %->% z1
sin(7) %->_% 'z2'
varname <- 'z3'
sin(7) %->_% varname
```

# Index

`%->% (%land%)`, 44  
`%->_% (%land%)`, 44  
`%land_% (%land%)`, 44  
`%land%`, 44

`addConstantColumn`, 3

`buildJoinPlan`, 4, 6, 10, 14, 15, 42

`dplyr_src_to_db_handle`, 5

`executeLeftJoinPlan`, 4, 5, 10, 14, 15, 42  
`expandColumn`, 7

`gapply`, 8, 16

`inspectDescrAndJoinPlan`, 4, 6, 10

`key_inspector_all_cols`, 12  
`key_inspector_postgresql`, 13  
`key_inspector_sqlite`, 13  
`keysAreUnique`, 11, 42

`let`, 17, 32, 36

`makeJoinDiagramSpec`, 4, 6, 10, 14, 15, 42

`renderJoinDiagram`, 14, 15  
`replyr`, 16  
`replyr-package (replyr)`, 16  
`replyr_add_ids`, 16  
`replyr_apply_f_mapped`, 17, 32, 36  
`replyr_arrange`, 18  
`replyr_bind_rows`, 18  
`replyr_check_ranks`, 19  
`replyr_coalesce`, 20  
`replyr_colClasses`, 21  
`replyr_copy_from`, 22  
`replyr_copy_to`, 23  
`replyr_dim`, 24  
`replyr_drop_table_name`, 24  
`replyr_filter`, 25  
`replyr_get_src`, 26  
`replyr_group_by`, 26  
`replyr_has_table`, 28  
`replyr_hasrows`, 27  
`replyr_inTest`, 28  
`replyr_is_local_data`, 29  
`replyr_is_MySQL_data`, 30  
`replyr_is_Spark_data`, 30  
`replyr_list_tables`, 31  
`replyr_mapRestrictCols`, 17, 31, 36  
`replyr_ncol`, 33  
`replyr_nrow`, 33  
`replyr_quantile`, 34  
`replyr_quantilec`, 35  
`replyr_rename`, 35  
`replyr_reverseMap`, 17, 32, 36  
`replyr_select`, 37  
`replyr_split`, 37  
`replyr_summary`, 38  
`replyr_testCols`, 40  
`replyr_union_all`, 40  
`replyr_uniqueValues`, 41

`tableDescription`, 4, 6, 10, 11, 14, 15, 42  
`topoSortTables`, 43