

# Package ‘rqdatatable’

July 29, 2018

**Type** Package

**Title** 'rquery' for 'data.table'

**Version** 0.1.3

**Date** 2018-07-28

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

**Description** Implements the 'rquery' piped Codd-style query algebra using 'data.table'. This allows for a high-speed in memory implementation of Codd-style data manipulation tools.

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

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

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**ByteCompile** true

**VignetteBuilder** knitr

**Depends** R (>= 3.4.0), rquery (>= 0.5.0)

**Imports** wrapr (>= 1.5.0), data.table (>= 1.11.4)

**RoxygenNote** 6.0.1

**Suggests** knitr, rmarkdown, DBI, RSQLite, parallel, testthat

**NeedsCompilation** no

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

**Repository** CRAN

**Date/Publication** 2018-07-29 16:50:02 UTC

## R topics documented:

ex_data_table . . . . .	2
ex_data_table.default . . . . .	4
ex_data_table.relop_drop_columns . . . . .	4
ex_data_table.relop_extend . . . . .	5
ex_data_table.relop_natural_join . . . . .	6
ex_data_table.relop_non_sql . . . . .	7
ex_data_table.relop_null_replace . . . . .	8
ex_data_table.relop_orderby . . . . .	9
ex_data_table.relop_project . . . . .	10
ex_data_table.relop_rename_columns . . . . .	11
ex_data_table.relop_select_columns . . . . .	11
ex_data_table.relop_select_rows . . . . .	12
ex_data_table.relop_set_indicator . . . . .	13
ex_data_table.relop_sql . . . . .	14
ex_data_table.relop_table_source . . . . .	15
ex_data_table.relop_theta_join . . . . .	16
ex_data_table.relop_unionall . . . . .	17
ex_data_table_parallel . . . . .	18
rqdatatable . . . . .	19
rq_df_funciton_node . . . . .	19
rq_df_grouped_funciton_node . . . . .	20
<b>Index</b>	<b>23</b>

---

ex_data_table	<i>Execute an rquery pipeline with data.table sources.</i>
---------------	--

---

### Description

data.tables are looked for by name in the tables argument and in the execution environment.

### Usage

```
ex_data_table(optree, ..., tables = list(), source_usage = NULL,
              source_limit = NULL, env = parent.frame())
```

### Arguments

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

## Details

- `ex_data_table.relop_drop_columns`: implement drop columns
- `ex_data_table.relop_extend`: implement extend/assign operator
- `ex_data_table.relop_natural_join`: implement natural join
- `ex_data_table.relop_non_sql`: direct function (non-sql) operator (not implemented for `data.table`)
- `ex_data_table.relop_null_replace`: implement NA/NULL replacement
- `ex_data_table.relop_orderby`: implement row ordering
- `ex_data_table.relop_project`: implement row ordering
- `ex_data_table.relop_rename_columns`: implement column renaming
- `ex_data_table.relop_select_columns`: implement select columns
- `ex_data_table.relop_select_rows`: implement select rows
- `ex_data_table.relop_sql`: direct sql operator (not implemented for `data.table`)
- `ex_data_table.relop_table_source`: implement data source
- `ex_data_table.relop_theta_join`: implement theta join (not implemented for `data.table`)
- `ex_data_table.relop_unionall`: implement row binding

## Value

resulting `data.table` (intermediate tables can sometimes be mutated as is practice with `data.table`).

## Examples

```
a <- data.table::data.table(x = c(1, 2) , y = c(20, 30), z = c(300, 400))
optree <- local_td(a) %.>%
  select_columns(., c("x", "y")) %.>%
  select_rows_nse(., x<2 & y<30)
cat(format(optree))
print(ex_data_table(optree)[])

# other ways to execute the pipeline include
as.data.frame(optree)
as.data.table(optree)
data.frame(x = 0, y = 4, z = 400) %.>% optree
```

---

ex\_data\_table.default *default non-impementation.*

---

### Description

Throw on error if this method is called, signalling that a specific data.table implementation is needed for this method.

### Usage

```
## Default S3 method:
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

### Arguments

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

---

ex\_data\_table.relop\_drop\_columns  
*Implement drop columns.*

---

### Description

data.table based implementation.

### Usage

```
## S3 method for class 'relop_drop_columns'
ex_data_table(optree, ..., tables = list(),
  source_limit = NULL, source_usage = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
source_usage	list mapping source table names to vectors of columns used.
env	environment to work in.

**Examples**

```
dL <- data.frame(x = 1, y = 2, z = 3)
rquery_pipeline <- local_td(dL) %>%
  drop_columns(., "y")
ex_data_table(rquery_pipeline)[[]
```

---

```
ex_data_table.relop_extend
```

*Implement extend/assign operator.*

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_extend'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Details**

Will re-order columns if there are ordering terms.

**Examples**

```
dL <- build_frame(
  "subjectID", "surveyCategory", "assessmentTotal", "one" |
  1, "withdrawal behavior", 5, 1 |
  1, "positive re-framing", 2, 1 |
  2, "withdrawal behavior", 3, 1 |
  2, "positive re-framing", 4, 1 )
rquery_pipeline <- local_td(dL) %>%
  extend_nse(.,
    probability :=%
      exp(assessmentTotal * 0.237)/
      sum(exp(assessmentTotal * 0.237)),
    count :=% sum(one),
    rank :=% rank(),
    orderby = c("assessmentTotal", "surveyCategory"),
    reverse = c("assessmentTotal"),
    partitionby = 'subjectID') %>%
  orderby(., c("subjectID", "probability"))
ex_data_table(rquery_pipeline)[[]
```

---

```
ex_data_table.relop_natural_join
  Natural join.
```

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_natural_join'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```

d1 <- build_frame(
  "key", "val", "val1" |
  "a"   , 1  , 10   |
  "b"   , 2  , 11   |
  "c"   , 3  , 12   )
d2 <- build_frame(
  "key", "val", "val2" |
  "a"   , 5  , 13   |
  "b"   , 6  , 14   |
  "d"   , 7  , 15   )

# key matching join
optree <- natural_join(local_td(d1), local_td(d2),
  jointype = "FULL", by = 'key')
ex_data_table(optree)[]

# full cross-product join
# (usually with jointype = "FULL", but "LEFT" is more
# compatible with rquery field merg semantics).
optree2 <- natural_join(local_td(d1), local_td(d2),
  jointype = "LEFT", by = NULL)
ex_data_table(optree2)[]
# notice ALL non-"by" fields take coalesce to left table.

```

---

```
ex_data_table.relop_non_sql
```

*Direct non-sql (function) node, not implemented for data.table case.*

---

**Description**

Passes a single table to a function that takes a single data.frame as its argument, and returns a single data.frame.

**Usage**

```

## S3 method for class 'relop_non_sql'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())

```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.

source\_limit if not null limit all table sources to no more than this many rows (used for debugging).  
 env environment to work in.

**See Also**

[rq\\_df\\_funciton\\_node](#), [rq\\_df\\_grouped\\_funciton\\_node](#)

**Examples**

```
set.seed(3252)
d <- data.frame(a = rnorm(1000), b = rnorm(1000))

optree <- local_td(d) %.>%
  quantile_node(.)
ex_data_table(optree)

p2 <- local_td(d) %.>%
  rsummary_node(.)
ex_data_table(p2)[[]

summary(d)
```

---

ex\_data\_table.relop\_null\_replace  
*Replace NAs.*

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_null_replace'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree relop operations tree.  
 ... not used, force later arguments to bind by name.  
 tables named list map from table names used in nodes to data.tables and data.frames.  
 source\_usage list mapping source table names to vectors of columns used.  
 source\_limit if not null limit all table sources to no more than this many rows (used for debugging).  
 env environment to work in.



**Examples**

```
dL <- build_frame(
  "x", "y" |
  2L , 5 |
  NA , 7 |
  NA , NA )
rquery_pipeline <- local_td(dL) %.>%
  null_replace(., c("x", "y"), 0, note_col = "nna")
ex_data_table(rquery_pipeline)[[]
```

---

```
ex_data_table.relop_orderby
```

*Reorder rows.*

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_orderby'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```
dL <- build_frame(
  "x", "y" |
  2L , "b" |
  1L , "a" |
  3L , "c" )
rquery_pipeline <- local_td(dL) %.>%
  orderby(., "y")
ex_data_table(rquery_pipeline)[[]
```

---

```
ex_data_table.relop_project
```

*Implement projection operator.*

---

## Description

data.table based implementation.

## Usage

```
## S3 method for class 'relop_project'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

## Arguments

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

## Examples

```
dL <- build_frame(
  "subjectID", "surveyCategory", "assessmentTotal" |
  1, "withdrawal behavior", 5 |
  1, "positive re-framing", 2 |
  2, "withdrawal behavior", 3 |
  2, "positive re-framing", 4 )
test_p <- local_td(dL) %.>%
  extend_nse(.,
    one :=% 1) %.>%
  project_nse(.,
    maxscore = max(assessmentTotal),
    groupby = 'subjectID')
cat(format(test_p))
ex_data_table(test_p)[[]]
```

---

```
ex_data_table.relop_rename_columns
  Rename columns.
```

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_rename_columns'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```
dL <- build_frame(
  "x", "y" |
  2L , "b" |
  1L , "a" |
  3L , "c" )
rquery_pipeline <- local_td(dL) %>%
  rename_columns(., c("x" = "y", "y" = "x"))
ex_data_table(rquery_pipeline)[]
```

---

```
ex_data_table.relop_select_columns
  Implement drop columns.
```

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_select_columns'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```
dL <- data.frame(x = 1, y = 2, z = 3)
rquery_pipeline <- local_td(dL) %.>%
  select_columns(., "y")
ex_data_table(rquery_pipeline)[]
```

---

```
ex_data_table.relop_select_rows
  Select rows by condition.
```

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_select_rows'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```
dL <- build_frame(
  "x", "y" |
  2L , "b" |
  1L , "a" |
  3L , "c" )
rquery_pipeline <- local_td(dL) %>%
  select_rows_nse(., x <= 2)
ex_data_table(rquery_pipeline)[ ]
```

---

```
ex_data_table.relop_set_indicator
```

*Implement set\_indicatoroperator.*

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_set_indicator'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```
d <- data.frame(a = c("1", "2", "1", "3"),
  b = c("1", "1", "3", "2"),
  q = 1,
  stringsAsFactors = FALSE)
set <- c("1", "2")
op_tree <- local_td(d) %>%
  set_indicator(., "one_two", "a", set) %>%
  set_indicator(., "z", "a", c())
```

```
ex_data_table(op_tree[])
```

---

```
ex_data_table.relop_sql
```

*Direct sql node.*

---

### Description

Execute one step using the rquery.rquery\_db\_executor SQL supplier. Note: it is not a good practice to use SQL nodes in data.table intended pipelines (loss of class information and cost of data transfer). This implementation is only here for completeness.

### Usage

```
## S3 method for class 'relop_sql'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

### Arguments

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

### Examples

```
# WARNING: example tries to change rquery.rquery_db_executor option to RSQLite and back.
if (requireNamespace("DBI", quietly = TRUE) &&
  requireNamespace("RSQLite", quietly = TRUE)) {
  # example database connection
  my_db <- DBI::dbConnect(RSQLite::SQLite(),
    ":memory:")
  old_o <- options(list("rquery.rquery_db_executor" = list(db = my_db)))

  # example data
  d <- data.frame(v1 = c(1, 2, NA, 3),
    v2 = c(NA, "b", NA, "c"),
    v3 = c(NA, NA, 7, 8),
    stringsAsFactors = FALSE)
```

```

# example xform
vars <- column_names(d)
# build a NA/NULLs per-row counting expression.
# names are "quoted" by wrapping them with as.name().
# constants can be quoted by an additional list wrapping.
expr <- lapply(vars,
  function(vi) {
    list("+ (CASE WHEN (",
      as.name(vi),
      "IS NULL ) THEN 1.0 ELSE 0.0 END)")
  })
expr <- unlist(expr, recursive = FALSE)
expr <- c(list(0.0), expr)

# instantiate the operator node
op_tree <- local_td(d) %.>%
  sql_node(., "num_missing" %:=% list(expr))
cat(format(op_tree))

ex_data_table(op_tree, tables = list(d = d)) %.>%
  print(.)
# d %.>% op_tree

options(old_o)
DBI::dbDisconnect(my_db)
}

```

---

```
ex_data_table.relop_table_source
```

*Build a data source description.*

---

## Description

data.table based implementation. Looks for tables first in tables and then in env. Will accept any data.frame that can be converted to data.table.

## Usage

```

## S3 method for class 'relop_table_source'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())

```

## Arguments

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.

source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

### Examples

```
dL <- build_frame(
  "x", "y" |
  2L , "b" |
  1L , "a" |
  3L , "c" )
rquery_pipeline <- local_td(dL)
ex_data_table(rquery_pipeline)[[]
```

---

```
ex_data_table.relop_theta_join
  Theta join (database implementation).
```

---

### Description

Limited implementation. All terms must be of the form: "(table1.col CMP table2.col) (, (table1.col CMP table2.col))".

### Usage

```
## S3 method for class 'relop_theta_join'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

### Arguments

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.



**Examples**

```
d1 <- data.frame(AUC = 0.6, R2 = 0.2)
d2 <- data.frame(AUC2 = 0.4, R2 = 0.3)

optree <- theta_join_se(local_td(d1), local_td(d2), "AUC >= AUC2")

ex_data_table(optree, tables = list(d1 = d1, d2 = d2)) %>%
  print(.)
```

---

```
ex_data_table.relop_unionall
```

*Bind tables together by rows.*

---

**Description**

data.table based implementation.

**Usage**

```
## S3 method for class 'relop_unionall'
ex_data_table(optree, ..., tables = list(),
  source_usage = NULL, source_limit = NULL, env = parent.frame())
```

**Arguments**

optree	relop operations tree.
...	not used, force later arguments to bind by name.
tables	named list map from table names used in nodes to data.tables and data.frames.
source_usage	list mapping source table names to vectors of columns used.
source_limit	if not null limit all table sources to no more than this many rows (used for debugging).
env	environment to work in.

**Examples**

```
dL <- build_frame(
  "x", "y" |
  2L , "b" |
  1L , "a" |
  3L , "c" )
rquery_pipeline <- unionall(list(local_td(dL), local_td(dL)))
ex_data_table(rquery_pipeline)[,]
```

---

 ex\_data\_table\_parallel

*Execute an rquery pipeline with data.table in parallel.*


---

### Description

Execute an rquery pipeline with `data.table` in parallel, partitioned by a given column. Note: usually the overhead of partitioning and distributing the work will by far overwhelm any parallel speedup. Also `data.table` itself already seems to exploit some thread-level parallelism (one often sees user time > elapsed time). Requires the `parallel` package. For a worked example with significant speedup please see [https://github.com/WinVector/rqdatatable/blob/master/extras/Parallel\\_rqdatatable.md](https://github.com/WinVector/rqdatatable/blob/master/extras/Parallel_rqdatatable.md).

### Usage

```
ex_data_table_parallel(optree, partition_column, cl = NULL, ...,
  tables = list(), source_limit = NULL, debug = FALSE,
  env = parent.frame())
```

### Arguments

<code>optree</code>	relop operations tree.
<code>partition_column</code>	character name of column to partition work by.
<code>cl</code>	a cluster object, created by package <code>parallel</code> or by package <code>snow</code> . If <code>NULL</code> , use the registered default cluster.
<code>...</code>	not used, force later arguments to bind by name.
<code>tables</code>	named list map from table names used in nodes to <code>data.tables</code> and <code>data.frames</code> .
<code>source_limit</code>	if not null limit all table sources to no more than this many rows (used for debugging).
<code>debug</code>	logical if <code>TRUE</code> use <code>lapply</code> instead of <code>parallel::clusterApplyLB</code> .
<code>env</code>	environment to look for values in.

### Details

Care must be taken that the calculation partitioning is coarse enough to ensure a correct calculation. For example: anything one is joining on, aggregating over, or ranking over must be grouped so that all elements affecting a given result row are in the same level of the partition.

### Value

resulting `data.table` (intermediate tables can sometimes be mutated as is practice with `data.table`).

---

rqdatatable	<i>rqdatatable: Relational Query Generator for Data Manipulation Implemented by data.table</i>
-------------	--

---

**Description**

Implements the rquery piped query algebra using `data.table`. This allows for a high-speed in memory implementation of Codd-style data manipulation tools.

---

rq_df_funciton_node	<i>Helper to build data.table capable non-sql nodes.</i>
---------------------	--

---

**Description**

Helper to build `data.table` capable non-sql nodes.

**Usage**

```
rq_df_funciton_node(., f, ..., f_db = NULL, columns_produced, display_form,
  orig_columns = FALSE)
```

**Arguments**

.	or <code>data.frame</code> input.
f	function that takes a <code>data.table</code> to a <code>data.frame</code> (or <code>data.table</code> ).
...	force later arguments to bind by name.
f_db	implementation signature: <code>f_db(db, incoming_table_name, outgoing_table_name)</code> (db being a database handle). NULL defaults to using f.
columns_produced	character columns produces by f.
display_form	display form for node.
orig_columns	<code>orig_columns</code> , if TRUE assume all input columns are present in derived table.

**Value**

relop non-sql node implementation.

**See Also**

[ex\\_data\\_table.relop\\_non\\_sql](#), [rq\\_df\\_grouped\\_funciton\\_node](#)

**Examples**

```

# a node generator is something an expert can
# write and part-time R users can use.
grouped_regression_node <- function(., group_col = "group", xvar = "x", yvar = "y") {
  force(group_col)
  formula_str <- paste(yvar, "~", xvar)
  f <- function(df) {
    dlist <- split(df, df[[group_col]])
    clist <- lapply(dlist,
      function(di) {
        mi <- lm(as.formula(formula_str), data = di)
        ci <- as.data.frame(summary(mi)$coefficients)
        ci$Variable <- rownames(ci)
        rownames(ci) <- NULL
        ci[[group_col]] <- di[[group_col]][[1]]
        ci
      })
    data.table::rbindlist(clist)
  }
  columns_produced =
    c("Variable", "Estimate", "Std. Error", "t value", "Pr(>|t|)", group_col)
  rq_df_funciton_node(
    ., f,
    columns_produced = columns_produced,
    display_form = paste0(yvar, "~", xvar, " grouped by ", group_col))
}

# work an example
set.seed(3265)
d <- data.frame(x = rnorm(1000),
  y = rnorm(1000),
  group = sample(letters[1:5], 1000, replace = TRUE),
  stringsAsFactors = FALSE)

rquery_pipeline <- local_td(d) %>%
  grouped_regression_node(.)

cat(format(rquery_pipeline))

ex_data_table(rquery_pipeline)[]

```

---

```
rq_df_grouped_funciton_node
```

*Helper to build data.table capable non-sql nodes.*

---

**Description**

Helper to build data.table capable non-sql nodes.

**Usage**

```
rq_df_grouped_funciton_node(., f, ..., f_db = NULL, columns_produced,
  group_col, display_form)
```

**Arguments**

. or data.frame input.

f function that takes a data.table to a data.frame (or data.table).

... force later arguments to bind by name.

f\_db implementation signature: f\_db(db, incoming\_table\_name, outgoing\_table\_name) (db being a database handle). NULL defaults to using f.

columns\_produced character columns produces by f.

group\_col character, column to split by.

display\_form display form for node.

**Value**

relop non-sql node implementation.

**See Also**

[ex\\_data\\_table.relop\\_non\\_sql](#), [rq\\_df\\_funciton\\_node](#)

**Examples**

```
# a node generator is something an expert can
# write and part-time R users can use.
grouped_regression_node <- function(., group_col = "group", xvar = "x", yvar = "y") {
  force(group_col)
  formula_str <- paste(yvar, "~", xvar)
  f <- function(di) {
    mi <- lm(as.formula(formula_str), data = di)
    ci <- as.data.frame(summary(mi)$coefficients)
    ci$Variable <- rownames(ci)
    rownames(ci) <- NULL
    colnames(ci) <- c("Estimate", "Std_Error", "t_value", "p_value", "Variable")
    ci
  }
  columns_produced =
    c("Estimate", "Std_Error", "t_value", "p_value", "Variable", group_col)
  rq_df_grouped_funciton_node(
    ., f,
    columns_produced = columns_produced,
    group_col = group_col,
    display_form = paste0(yvar, "~", xvar, " grouped by ", group_col))
}
```

```
# work an example
set.seed(3265)
d <- data.frame(x = rnorm(1000),
               y = rnorm(1000),
               group = sample(letters[1:5], 1000, replace = TRUE),
               stringsAsFactors = FALSE)

rquery_pipeline <- local_td(d) %>%
  grouped_regression_node(.)

cat(format(rquery_pipeline))

ex_data_table(rquery_pipeline[])

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

  rquery::to_sql(rquery_pipeline, my_db) %>%
    print(.)

  dR <- rquery::rq_copy_to(my_db,
                          d = d,
                          table_name = "d",
                          overwrite = TRUE,
                          temporary = TRUE)
  tbl <- rquery::materialize(my_db, rquery_pipeline,
                           overwrite = FALSE,
                           temporary = TRUE)
  DBI::dbReadTable(my_db, tbl$table_name) %>%
    print(.)

  DBI::dbDisconnect(my_db)
}
```

# Index

`ex_data_table`, [2](#)  
`ex_data_table.default`, [4](#)  
`ex_data_table.relop_drop_columns`, [3, 4](#)  
`ex_data_table.relop_extend`, [3, 5](#)  
`ex_data_table.relop_natural_join`, [3, 6](#)  
`ex_data_table.relop_non_sql`, [3, 7, 19, 21](#)  
`ex_data_table.relop_null_replace`, [3, 8](#)  
`ex_data_table.relop_orderby`, [3, 9](#)  
`ex_data_table.relop_project`, [3, 10](#)  
`ex_data_table.relop_rename_columns`, [3, 11](#)  
`ex_data_table.relop_select_columns`, [3, 11](#)  
`ex_data_table.relop_select_rows`, [3, 12](#)  
`ex_data_table.relop_set_indicator`, [13](#)  
`ex_data_table.relop_sql`, [3, 14](#)  
`ex_data_table.relop_table_source`, [3, 15](#)  
`ex_data_table.relop_theta_join`, [3, 16](#)  
`ex_data_table.relop_unionall`, [3, 17](#)  
`ex_data_table_parallel`, [18](#)  
  
`rq_df_funciton_node`, [8, 19, 21](#)  
`rq_df_grouped_funciton_node`, [8, 19, 20](#)  
`rqdatatable`, [19](#)  
`rqdatatable-package (rqdatatable)`, [19](#)