Package ‘validatedb’

March 18, 2021

Title Validate Data in a Database using ‘validate’
Version 0.1.3
Description Check whether records in a database table are valid using
validation rules in R syntax specified with R package ‘validate’.
R validation checks are automatically translated in SQL using ‘dbplyr’.
License GPL-3
Encoding UTF-8
LazyData true
RoxygenNote 7.1.1
Depends validate
Imports dplyr, dbplyr, methods
Suggests testthat, RSQLite, covr
URL https://github.com/data-cleaning/validatedb
BugReports https://github.com/data-cleaning/validatedb/issues
Collate 'aggregate.R' 'tbl_validation.R' 'as-data-frame.R' 'compute.R'
   'confront.R' 'confront_tbl.R' 'confront_tbl_sparse.R'
   'is_record_based.R' 'rule_works_on_tbl.R' 'show_query.R'
   'summary.R' 'tbl.R' 'validatedb-package.R' 'values.R'
NeedsCompilation no
Author Edwin de Jonge [aut, cre] (<https://orcid.org/0000-0002-6580-4718>),
   Mark van der Loo [aut]
Maintainer Edwin de Jonge <edwindjonge@gmail.com>
Repository CRAN
Date/Publication 2021-03-18 09:10:03 UTC

R topics documented:

aggregate.tbl_validation .................................................. 2
as.data.frame.tbl_validation-method ................................ 3
Count the number of invalid rules or records.

Description
See the number of valid and invalid checks either by rule or by record.

Usage

```r
## S3 method for class 'tbl_validation'
aggregate(x, by = c("rule", "record", "id"), ...)
```

Arguments

- `x`: `tbl_validation()` object
- `by`: either by "rule" or by "record"
- `...`: not used

Details
The result of a `confront()` on a db tbl results in a lazy sqquery. That is it builds a query without executing it. To store the result in the database use `compute()` or `values()`.

Value

A `dbplyr::tbl_dbi()` object that represents the aggregation query (to be executed) on the database.

Examples

```r
income <- data.frame(id = 1:2, age=c(12,35), salary = c(1000,NA))
con <- dbplyr::src_memdb()
tbl_income <- dplyr::copy_to(con, income, overwrite=TRUE)
print(tbl_income)

# Let's define a rule set and confront the table with it:
rules <- validator( is_adult = age >= 18
                     , has_income = salary > 0
```
# and confront!
# in general with a db table it is handy to use a key
cf <- confront(tbl_income, rules, key="id")
aggregate(cf, by = "rule")
aggregate(cf, by = "record")

# to tweak performance of the db query the following options are available
# 1) store validation result in db
cf <- confront(tbl_income, rules, key="id", compute = TRUE)
# or identical
cf <- confront(tbl_income, rules, key="id")
cf <- compute(cf)

# 2) Store the validation sparsely
cf_sparse <- confront(tbl_income, rules, key="id", sparse=TRUE )

show_query(cf_sparse)
values(cf_sparse, type="tbl")

---

as.data.frame.tbl_validation-method

Retrieve validation results as a data.frame

### Description

Retrieve validation results as a data.frame

### Usage

```r
## S4 method for signature 'tbl_validation'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

### Arguments

- `x`  
  `tbl_validation()`, result of a confront() of tbl with a rule set.
- `row.names`  
  ignored
- `optional`  
  ignored
- `...`  
  ignored

### Value

data.frame, result of the query on the database.
Examples

```r
# create a table in a database
income <- data.frame(id = letters[1:2], age=c(12,35), salary = c(1000,NA))
con <- dbplyr::src_memdb()
tbl_income <- dplyr::copy_to(con, income, overwrite=TRUE)

# Let's define a rule set and confront the table with it:
rules <- validator( is_adult = age >= 18
                   , has_income = salary > 0
                   , mean_age = mean(age,na.rm=TRUE) > 20
                   )

# and confront!
 cf <- confront(tbl_income, rules, key = "id")
 as.data.frame(cf)

# and now with a sparse result:
 cf <- confront(tbl_income, rules, key = "id", sparse=TRUE)
 as.data.frame(cf)
```

compute.tbl_validation

---

**Description**

Stores the validation result in the db using the `dplyr::compute()` of the db back-end. This method changes the `tbl_validation` object! Note that for most back-ends the default setting is a temporary table with a random name.

**Usage**

```r
## S3 method for class 'tbl_validation'
compute(x, name, ...)
```

**Arguments**

- `x` : `tbl_validation()`, result of a `confront()` of tbl with a rule set.
- `name` : optional, when omitted, a random name is used.
- `...` : passed through to compute on `x$query`

**Value**

A `dbplyr::tbl_dbi()` object that refers to the computed (temporary) table in the database. See `dplyr::compute()`.

**See Also**

Other `tbl_validation`: `tbl_validation-class`
confront.tbl_sql  

Validate data in database tbl with validator rules.

Description

Confront dbplyr::tbl_dbi() objects with validate::validator() rules, making it possible to execute validator() rules on database tables. Validation results can be stored in the db or retrieved into R.

Usage

confront.tbl_sql(tbl, x, ref, key = NULL, sparse = FALSE, compute = FALSE, ...)

## S4 method for signature 'ANY,validator,ANY'
confront(dat, x, ref, key = NULL, sparse = FALSE, ...)

Arguments

- tbl dbplyr::tbl_dbi() table in a database, retrieved with tbl()
- x validate::validator() object with validation rules.
- ref reference object (not working)
- key character with key column name.
- sparse logical should only fails be stored in the db?
- compute logical if TRUE the check stores a temporary table in the database.
- ... passed through to compute(), if compute is TRUE
- dat an object of class ‘tbl_sql‘.

Details

validatedb builds upon dplyr and dbplyr, so it works on all databases that have a dbplyr compatible database driver (DBI / odbc). validatedb translates validator rules into dplyr commands resulting in a lazy query object. The result of a validation can be stored in the database using compute or retrieved into R with values.

Value

a tbl_validation() object, containing the confrontation query and processing information.

See Also

Other validation: tbl_validation-class, values, tbl_validation-method
Examples

# create a table in a database
income <- data.frame(id = letters[1:2], age=c(12,35), salary = c(1000,NA))
con <- dbplyr::src_memdb()
tbl_income <- dplyr::copy_to(con, income, overwrite=TRUE)
print(tbl_income)

# Let's define a rule set and confront the table with it:
rules <- validator( is_adult = age >= 18
 , has_income = salary > 0
 , mean_age = mean(age,na.rm=TRUE) > 20
 )

# and confront!
cf <- confront(tbl_income, rules)
print(cf)
summary(cf)

# Values (i.e. validations on the table) can be retrieved like in 'validate'
# with 'type="matrix"' (simplify = TRUE)
values(cf, type = "matrix")

# But often this seems more handy:
values(cf, type = "tbl")

# We can see the sql code by using 'show_query':
show_query(cf)

# identical
show_query(values(cf, type = "tbl"))

# adding a key often is handy in a database
cf <- confront(tbl_income, rules, key = "id")
print(cf)
values(cf, type="tbl")

# sparse results in db
cf_sparse <- confront(tbl_income, rules, sparse=TRUE)
values(cf_sparse, type="tbl")

confront_tbl

create a table with per record if it abides to the rule.

Description

create a table with per record if it abides to the rule.

Usage

confront_tbl(tbl, x, key = NULL)
## Arguments

- **tbl**: `dbplyr::tbl_dbi()` object that refers to the confrontation query.
- **x**: `validate::validator()` object with validation rules.
- **key**: character with key column name.

## Details

The return value of the function is a list with:

- `$query`: A `dbplyr::tbl_dbi()` object that refers to the confrontation query.
- `$errors`: The validation rules that are not working on the database.
- `$working`: A logical with which expression are working on the database.
- `$exprs`: All validation expressions.
- `$nexprs`: Number of working expression.

## Value

A list with needed information, see details.

---

### confront_tbl_sparse
Create a sparse confrontation query

---

## Description

Create a sparse confrontation query. Only errors and missing are stored. This can be useful alternative to `confront_tbl()` which stores all results of a tbl validation in a table with `length(rules)` columns and `nrow(tbl)` rows. Note that the result of this function is a (lazy) query object that still needs to be executed in the database, e.g. with `dplyr::collect()`, `dplyr::collapse()` or `dplyr::compute()`.

## Usage

```r
confront_tbl_sparse(tbl, x, key = NULL, union_all = TRUE, check_rules = TRUE)
```

## Arguments

- **tbl**: `dbplyr::tbl_dbi()` object that refers to the database, retrieved with `tbl()`.
- **x**: `validate::validator()` object with validation rules.
- **key**: character with key column name.
- **union_all**: if `FALSE` each rule is a separate query.
- **check_rules**: if `TRUE` it is checked which rules 'work' on the db.
Details

The return value of the function is a list with:

- `$query`: A `dbplyr::tbl_dbi()` object that refers to the confrontation query.
- `$errors`: The validation rules that are not working on the database.
- `$working`: A logical with which expression are working on the database.
- `$exprs`: All validation expressions.

Value

A object with the necessary information: see details

See Also

Other validation: `tbl_validation-class, values, tbl_validation-method`

Examples

```r
# create a table in a database
income <- data.frame(id = letters[1:2], age=c(12,35), salary = c(1000,NA))
con <- dbplyr::src_memdb()
tbl_income <- dplyr::copy_to(con, income, overwrite=TRUE)
print(tbl_income)

# Let's define a rule set and confront the table with it:
rules <- validator(is_adult = age >= 18,
                   has_income = salary > 0,
                   mean_age = mean(age,na.rm=TRUE) > 20)

# and confront!
cf <- confront(tbl_income, rules)
print(cf)
summary(cf)

# Values (i.e. validations on the table) can be retrieved like in `validate`
# with `type="matrix"` (simplify = TRUE)
values(cf, type = "matrix")

# But often this seems more handy:
values(cf, type = "tbl")

# We can see the sql code by using `show_query`:
show_query(cf)

# identical
show_query(values(cf, type = "tbl"))

# adding a key often is handy in a database
cf <- confront(tbl_income, rules, key = "id")
print(cf)
```
rule_works_on_tbl

values(cf, type="tbl")

# sparse results in db
cf_sparse <- confront(tbl_income, rules, sparse=TRUE)
values(cf_sparse, type="tbl")

rule_works_on_tbl tests for each rule if it can be executed on the database

Description

tests for each rule if it can be executed on the database

Usage

rule_works_on_tbl(tbl, x)

Arguments

tbl a tbl object with columns used in x
x a validate::validator() object

Value

logical encoding which validation rules "work" on the database.

show_query.tbl_validation

Show generated sql code

Description

Shows the generated sql code for the validation of the tbl.

Usage

## S3 method for class 'tbl_validation'
show_query(x, ...)

Arguments

x tbl_validation() object, result of a confront.tbl_sql().
... passed through.

Value

Same result as dplyr::show_query, i.e. the SQL text of the query.
Validation object for tbl object

Description

Validation information for a database tbl, result of a `confront.tbl_sql()`.

Details

The `tbl_validation` object contains all information needed for the confrontation of validation rules with the data in the database table. It contains:

- `$query`: a `dbplyr::tbl_dbi` object with the query to be executed on the database
- `$tbl`: the `dbplyr::tbl_dbi` pointing to the table in the database
- `$key`: Whether there is a key column, and if so, what it is.
- `$record_based`: logical with which rules are record based.
- `$exprs`: list of validation rule expressions
- `$working`: logical, which of the rules work on the database. (whether the database supports this expression)
- `$errors`: list of validation rules that did not execute on the database.
- `$sparse`: If TRUE the query is stored as a sparse validation object.

Value

tbl_validation object. See details.

See Also

Other validation: `confront.tbl_sql()`, `values.tbl_validation-method`
Other tbl_validation: `compute.tbl_validation()`

Description

Retrieve the result of a validation/confrontation.
values.tbl_validation-method

Usage

```r
## S4 method for signature 'tbl_validation'
values(
  x,
  simplify = type == "matrix",
  type = c("tbl", "matrix", "list", "data.frame"),
  ...
)
```

Arguments

- `x` - `tbl_validation()`, result of a `confront()` of tbl with a rule set.
- `simplify` - only use when `type = "list"` see validate::values
- `type` - whether to return a list/matrix or to return a query on the database.
- `...` - not used

Details

Since the validation is done on a database, there are multiple options for storing the result of the validation. The results show per record whether they are valid according to the validation rules supplied.

- Use `compute` (see `confront.tbl_sql()`) to store the result in the database
- Use `sparse` to only calculate "fails" and "missings"

Default type "tbl" is that everything is "lazy", so the query and/or storage has to be done explicitly by the user. The other types execute the query and retrieve the result into R. When this creates memory problems, the `tbl` option is to be preferred.

Results for `type`:

- `tbl`: a `dbplyr::tbl_dbi` object, pointing to the database
- `matrix`: a R matrix, similar to `validate::values()`.
- `list`: a R list, similar to `validate::values()`.
- `data.frame`: the result of `tbl` stored in a `data.frame`.

Value

depending on `type` the result is different, see details

See Also

Other validation: `confront.tbl_sql()`, `tbl_validation-class`
Examples

# create a table in a database
income <- data.frame(id = letters[1:2], age=c(12,35), salary = c(1000,NA))
con <- dbplyr::src_memdb()
tbl_income <- dplyr::copy_to(con, income, overwrite=TRUE)
print(tbl_income)

# Let's define a rule set and confront the table with it:
rules <- validator( is_adult = age >= 18
                   , has_income = salary > 0
                   , mean_age = mean(age,na.rm=TRUE) > 20
                   )

# and confront!
cf <- confront(tbl_income, rules)
print(cf)
summary(cf)

# Values (i.e. validations on the table) can be retrieved like in `validate`
# with `type="matrix"` (simplify = TRUE)
values(cf, type = "matrix")

# But often this seems more handy:
values(cf, type = "tbl")

# We can see the sql code by using `show_query`:
show_query(cf)

# identical
show_query(values(cf, type = "tbl"))

# adding a key often is handy in a database
cf <- confront(tbl_income, rules, key = "id")
print(cf)
values(cf, type="tbl")

# sparse results in db
cf_sparse <- confront(tbl_income, rules, sparse=TRUE)
values(cf_sparse, type="tbl")
Index

* confront
  confront_tbl_sparse, 7
* tbl_validation
  compute.tbl_validation, 4
  tbl_validation-class, 10
* validation
  confront.tbl_sql, 5
  tbl_validation-class, 10
  values, tbl_validation-method, 10

aggregate.tbl_validation, 2
as.data.frame, tbl_validation-method, 3

compute(), 2, 5
compute.tbl_validation, 4, 10
confront(), 2
confront, ANY, validator, ANY-method
  (confront.tbl_sql), 5
confront.tbl_sql, 5, 10, 11
confront.tbl_sql(), 9–11
confront_tbl, 6
confront_tbl(), 7
confront_tbl_sparse, 7

dplyr::tbl_dbi, 10, 11
dplyr::tbl_dbi(), 2, 4, 5, 7, 8
dplyr::collapse(), 7
dplyr::collect(), 7
dplyr::compute(), 4, 7
dplyr::show_query, 9

rule_works_on_tbl, 9

show_query.tbl_validation, 9

tbl(), 5, 7
tbl_validation (tbl_validation-class), 10
tbl_validation(), 2–5, 9, 11
tbl_validation-class, 10