Package 'RSQL'

February 5, 2020

Type Package

Title Database Agnostic Package to Generate and Process 'SQL' Queries in R

Version 0.1.1

Language en-US

Maintainer Alejandro Baranek <abaranek@dc.uba.ar>

Description
Allows the user to generate and execute select, insert, update and delete 'SQL' queries the underlying database without having to explicitly write 'SQL' code.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

Collate 'sql-lib.R' 'zzz.R'

Imports lgr, R6, DBI, RSQLite

Suggests knitr, testthat, rmarkdown, dplyr, covr

VignetteBuilder knitr

BugReports https://github.com/rOpenStats/rsql/issues

URL https://github.com/rOpenStats/rsql

NeedsCompilation no

Author Alejandro Baranek [cre, aut],
Leonardo Belen [aut]

Repository CRAN

Date/Publication 2020-02-05 16:10:11 UTC

R topics documented:

`add_grep_exact_match` *add_grep_exact_match*

Description

add_grep_exact_match

Usage

```
add_grep_exact_match(text)
```

Arguments

text	TEST

---

add_quotes *Adds quotes to a string*

Description

Adds quotes to a string

Usage

```
add_quotes(text)
```

Arguments

text	The string to quote

---

cbind_coerced *TODO: WHAT DOES THIS DO AGAIN?*

Description

TODO: WHAT DOES THIS DO AGAIN?

Usage

```
cbind_coerced(...)
```

Arguments

...	The parameters

createRSQL *Produces a RSQL object*

Description

Produces a RSQL object

Usage

```
createRSQL(drv, dbname, user = NULL, password = NULL, host = NULL, port = NULL)
```

Arguments

drv	Driver name
dbname	Database name
user	Database user name
password	Database password
host	Database host
port	Database port

dequote *Removes the quotes from the string*

Description

Removes the quotes from the string

Usage

```
dequote(text)
```

Arguments

text	The string to remove the quotes from.

df_verify *Checks that the columns are in the data.frame*

Description

Checks that the columns are in the data.frame

Usage

```
df_verify(dataframe, columns)
```

Arguments

dataframe	The data.frame
columns	The columns to check

execute_get_insert *Executes the insert statement*

Description

Executes the insert statement

Usage

```
execute_get_insert(dbconn, sql_select, sql_insert, ...)
```

Arguments

dbconn	The db connection
sql_select	The SQL select query
sql_insert	The SQL insert query
...	other variables to considered.

getMtcarsdbPath *getCarsdbPath*

Description

getCarsdbPath

Usage

```
getMtcarsdbPath(copy = TRUE)
```

Arguments

copy	a boolean that states whether it should be copied to the home directory or not.

getPackageDir *Get package directory*

Description

Gets the path of package data.

Usage

```
getPackageDir()
```

is_quoted *Determines if the string is quoted or not*

Description

Determines if the string is quoted or not

Usage

```
is_quoted(text, quotes_symbols = c("'", "'"))
```

Arguments

text	The text to test
quotes_symbols	The quotation characters

parse_where_clause *Parses a where clause.*

Description

Parses a where clause.

Usage

```
parse_where_clause(where_clause_list = c())
```

Arguments

where_clause_list	The list of params

remove_quotes *Removes quotes from data.frame columns*

Description

Removes quotes from data.frame columns

Usage

```
remove_quotes(text)
```

Arguments

text	The text column to remove quotes from.

rename_col *renames a column on a data.frame*

Description

renames a column on a data.frame

Usage

```
rename_col(df, name, replace_name)
```

Arguments

df	The date.frame
name	The name of the column
replace_name	The new name of the column

re_quote *TODO: WHAT IS THIS FUNCTION DOING AGAIN?*

Description

TODO: WHAT IS THIS FUNCTION DOING AGAIN?

Usage

```
re_quote(text, quotes = "'")
```

Arguments

text	The string
quotes	The quotes

rm_quotes *Removes quotes from the String*

Description

Removes quotes from the String

Usage

```
rm_quotes(text, quotes = "'")
```

Arguments

text	The string to remove quotes from
quotes	Quote characters

RSQL *rsql*

Description

A package to work with SQL datasources in a simple manner

Usage

```
.onLoad(libname, pkgname)
```

Arguments

libname	Library name
pkgname	Package name

Author(s)

Alejandro Baranek <abaranek@dc.uba.ar>, Leonardo Javier Belen <leobelen@gmail.com> Executes code while loading the package.

RSQL.class *The class that provides the SQL functionality.*

Description

This class is intended to simplify SQL commands.

Public fields

`driver` driver name

`db.name` database name

`available.functions` for generating select expressions

`entity.field.regexp` for scrape a field or table expression

`entity.select.regexp` for scrape a select expressions expression

`conn` The connection handler

`last.query` The last query

`last.rs` The last resultset

`select.counter` An instance select counter

`insert.counter` An instance insert counter

`update.counter` An instance update counter

`delete.counter` An instance delete counter

`command.counter` An instance command counter

Methods

Public methods:

- RSQL.class$new()
- RSQL.class$setupRegexp()
- RSQL.class$finalize()
- RSQL.class$checkEntitiesNames()
- RSQL.class$gen_select()
- RSQL.class$gen_insert()
- RSQL.class$gen_update()
- RSQL.class$gen_delete()
- RSQL.class$execute_select()
- RSQL.class$execute_update()
- RSQL.class$execute_insert()
- RSQL.class$execute_command()
- RSQL.class$execute_delete()
- RSQL.class$retrieve()
- RSQL.class$retrieve_insert()
- RSQL.class$disconnect()
- RSQL.class$clone()

Method `new()`**:** Initializes a connection

Usage:

```
RSQL.class$new(
  drv,
  dbname,
  user = NULL,
  password = NULL,
  host = NULL,
  port = NULL
)
```

Arguments:

`drv` driver name

`dbname` database name

`user` user name

`password` password

`host` host name

`port` port number

Method `setupRegexp()`**:** initialize regexp for scraping entities

Usage:

```
RSQL.class$setupRegexp(force = FALSE)
```

Arguments:

`force` force setup?

Returns: regexp for scraping select expressions

Method `finalize()`**:** Class destructor

Usage:

```
RSQL.class$finalize()
```

Method `checkEntitiesNames()`**:** Checks if an entity exists

Usage:

```
RSQL.class$checkEntitiesNames(entities, entity.type)
```

Arguments:

`entities` entities to check

`entity.type` entity type to check against

Method `gen_select()`**:** Generates a select

Usage:

```
RSQL.class$gen_select(
  select_fields,
  table,
  where_fields = names(where_values),
  where_values = NULL,
  group_by = c(),
  order_by = c(),
  top = 0,
  distinct = FALSE
)
```

Arguments:

`select_fields` fields to be selected

`table` table to select from

`where_fields` fields in the where clause

`where_values` values to the fields on the where clause

`group_by` fields to group by

`order_by` fields to order by

`top` where does the resultset starts?

`distinct` provides a way to select distinct rows

Method `gen_insert()`**:** Generate insert statement

Usage:

```
RSQL.class$gen_insert(table, values_df, insert_fields = names(values_df))
```

Arguments:

`table` The table to insert into

`values_df` The values to insert. Must be defined as data.frame of values

`insert_fields` the fields to insert into

Method `gen_update()`**:** Generate insert statement

Usage:

```
RSQL.class$gen_update(
  table,
  update_fields = names(values),
  values,
  where_fields = names(where_values),
  where_values = NULL
)
```

Arguments:

`table` the table to insert into

`update_fields` the fields to update

`values` the values to update

`where_fields` a where clause to the insert

`where_values` the values to add to the where clause

Method `gen_delete()`**:** Generate a delete statement

Usage:

```
RSQL.class$gen_delete(
  table,
  where_fields = names(where_values),
  where_values = NULL
)
```

Arguments:

`table` the table to insert into

`where_fields` a where clause to the insert

`where_values` the fields to add to the where clause

Method `execute_select()`**:** Performs an execution on the database

Usage:

```
RSQL.class$execute_select(sql_select)
```

Arguments:

`sql_select` the sql select statement to perform

Method `execute_update()`**:** Performs an update on the database

Usage:

```
RSQL.class$execute_update(sql_update)
```

Arguments:

`sql_update` the sql update statement to perform

Method `execute_insert()`**:** Performs an insert on the database

Usage:

```
RSQL.class$execute_insert(sql_insert)
```

Arguments:

`sql_insert` the sql insert statement to perform

Method `execute_command()`**:** Performs a command on the database

Usage:

```
RSQL.class$execute_command(sql_command)
```

Arguments:

`sql_command` the sql statement to perform

Method `execute_delete()`**:** Performs an deletion on the database

Usage:

```
RSQL.class$execute_delete(sql_delete)
```

Arguments:

`sql_delete` the sql delete statement to perform

Method `retrieve()`**:** Performs an insert on the database. This is a composite function

Usage:

```
RSQL.class$retrieve(
  table,
  fields_uk = names(values_uk),
  values_uk,
  fields = names(values),
  values,
  field_id = "id"
)
```

Arguments:

`table` The table

`fields_uk` The fields unique key

`values_uk` The values unique key

`fields` The fields (Not used. Included for compatibility)

`values` The values (Not used. Included for compatibility)

`field_id` The field of the serial id

Method `retrieve_insert()`**:** Obtain id if object exists on the database. Insert object if not.

Usage:

```
RSQL.class$retrieve_insert(
  table,
  fields_uk = names(values_uk),
  values_uk,
  fields = names(values),
  values,
  field_id = "id"
)
```

Arguments:

`table` The table

`fields_uk` The fields unique key

`values_uk` The values unique key

`fields` The fields

`values` The values

`field_id` The field of the serial id

Method `disconnect()`**:** Disconnects the instance from the database

Usage:

```
RSQL.class$disconnect()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RSQL.class$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(RSQL)
db.name <- getMtcarsdbPath(copy = TRUE)
rsql <- createRSQL(drv = RSQLite::SQLite(), dbname = db.name)
where_values_df <- data.frame(carb = 8, stringsAsFactors = FALSE)
select_sql <- rsql$gen_select(
 select_fields = "*", #c("wt", "qsec"),
 table = "mtcars",
 where_values = where_values_df)
mtcars.observed <- rsql$execute_select(select_sql)
 mtcars.observed

 mtcars.new <- mtcars.observed
mtcars.new$carb <- 9
insert_sql <- rsql$gen_insert(table = "mtcars", values_df = mtcars.new)
rsql$execute_insert(sql_insert = insert_sql)

where_values_df <- data.frame(carb = 9, stringsAsFactors = FALSE)
select_sql <- rsql$gen_select(
  select_fields = "*", #c("wt", "qsec"),
 table = "mtcars",
 where_values = where_values_df)
mtcars.observed <- rsql$execute_select(select_sql)
mtcars.observed
```

sql_execute_delete *sql_execute_delete*

Description

Executes a delete on the Database

Usage

```
sql_execute_delete(sql_delete, dbconn = NULL)
```

Arguments

sql_delete	The delete SQL
dbconn	The Database Connection to run the query against

sql_execute_insert *Executes a statement on the database.*

Description

Executes a statement on the database.

Usage

```
sql_execute_insert(sql_insert, dbconn = NULL)
```

Arguments

sql_insert	The SQL String
dbconn	The Database Connection to run the query against

sql_execute_select *Executes a select on the database*

Description

Executes a select on the database

Usage

```
sql_execute_select(sql_select, dbconn = NULL)
```

Arguments

sql_select	The delete SQL
dbconn	The Database Connection to run the query against

sql_execute_update *Executes an update on the database*

Description

Executes an update on the database

Usage

```
sql_execute_update(sql_update, dbconn = NULL)
```

Arguments

sql_update	The update SQL
dbconn	The Database Connection to run the query against

sql_gen_delete *Generates a Delete Statement*

Description

Generates a Delete Statement

Usage

```
sql_gen_delete(table, where_fields = names(where_values), where_values = NULL)
```

Arguments

table	The table from which the delete statement will be generated
where_fields	The fields used in the where section
where_values	The values used in the where section

sql_gen_insert *Generates an insert statement.*

Description

Generates an insert statement.

Usage

```
sql_gen_insert(table, values_df, insert_fields = names(values_df))
```

Arguments

table	The table to be affected
values_df	The values to insert. Must be defined as data.frame of values
insert_fields	The fields to insert

sql_gen_select *Generates a Select Statement*

Description

Generates a Select Statement

Usage

```
sql_gen_select(
  select_fields,
  table,
  where_fields = names(where_values),
  where_values = NULL,
  group_by = c(),
  order_by = c(),
  top = 0,
  distinct = FALSE
)
```

Arguments

select_fields	The fields to be selected
table	The table to be used in the select
where_fields	The fields used in the where section
where_values	The values used in the where section
group_by	Group by fields

order_by	Order by fields
top	Retrieve top records
distinct	it adds a distinct clause to the query.

---

sql_gen_update *Generates an update statement*

Description

Generates an update statement

Usage

```
sql_gen_update(
  table,
  update_fields = names(values),
  values,
  where_fields = names(where_values),
  where_values
)
```

Arguments

table	The table to update
update_fields	The fields to update
values	The values to update
where_fields	The fields for where statement
where_values	The values for where statement

---

sql_gen_where *Generates a where statement to be used on a SQL statement.*

Description

Generates a where statement to be used on a SQL statement.

Usage

```
sql_gen_where(where_fields = names(where_values), where_values)
```

Arguments

where_fields	The fields used in the where section
where_values	The values used in the where section

sql_gen_where_list *Generates a where list statement to be used on a SQL statement.*

Description

Generates a where list statement to be used on a SQL statement.

Usage

```
sql_gen_where_list(where_fields, where_values)
```

Arguments

where_fields	The fields used in the where section
where_values	The values used in the where section

sql_gen_where_or *Generates a where (or) statement to be used on a SQL statement.*

Description

Generates a where (or) statement to be used on a SQL statement.

Usage

```
sql_gen_where_or(where_fields = names(where_values), where_values)
```

Arguments

where_fields	The fields used in the where section
where_values	The values used in the where section

sql_retrieve *Retrieves Statement*

Description

Retrieves Statement

Usage

```
sql_retrieve(
  table,
  fields_uk = names(values_uk),
  values_uk,
  fields = names(values),
  values = NULL,
  field_id = "id",
  dbconn = NULL
)
```

Arguments

table	The table
fields_uk	The fields unique key
values_uk	The values unique key
fields	The fields (Not used. Included for compatibility)
values	The values (Not used. Included for compatibility)
field_id	The field of the serial id
dbconn	The database connection

sql_retrieve_insert *Retrieves or insert Statement*

Description

Retrieves or insert Statement

Usage

```
sql_retrieve_insert(
  table,
  fields_uk = names(values_uk),
  values_uk,
  fields = names(values),
  values = NULL,
```

```
  field_id = "id",
  dbconn = NULL
)
```

Arguments

table	The table
fields_uk	The fields unique key
values_uk	The values unique key
fields	The fields
values	The values
field_id	The field of the serial id
dbconn	The database connection

---

trim *Returns string w/o leading or trailing whitespace*

Description

Returns string w/o leading or trailing whitespace

Usage

```
trim(x)
```

Arguments

x	The string

---

trim_leading *Returns string w/o leading whitespace*

Description

Returns string w/o leading whitespace

Usage

```
trim_leading(x)
```

Arguments

x	The string

trim_trailing *Returns string w/o trailing whitespace*

Description

Returns string w/o trailing whitespace

Usage

```
trim_trailing(x)
```

Arguments

x	The string

%IN% *Operator IN for multiple columns*

Description

Operator IN for multiple columns

Usage

```
x %IN% y
```

Arguments

x	TEST
y	TEST

Index