Package 'pgTools'

January 25, 2021

Type Package

Title Functions for Generating PostgreSQL Statements/Scripts

Version 0.1.0

Author Timothy Conwell

Maintainer Timothy Conwell <timconwell@gmail.com>

Description Create PostgreSQL statements/scripts from R, optionally executing the SQL statements. Common SQL operations are included, although not every configurable option is available at this time.
SQL output is intended to be compliant with PostgreSQL syntax specifications. PostgreSQL documentation is available here
<https://www.postgresql.org/docs/current/index.html>.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends DBI, data.table

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-01-25 08:20:05 UTC

R topics documented:

`alterDATABASE` *Generate a postgreSQL ALTER DATABASE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL ALTER DATABASE statement, optionally execute the statement if con is not NULL.

Usage

```
alterDATABASE(
  name,
  allow_connections = NULL,
  connection_limit = NULL,
  is_template = NULL,
  rename_to = NULL,
  owner_to = NULL,
  set_tablespace = NULL,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL ALTER DATABASE.
allow_connections	A string, the "allowconn" parameter for postgreSQL ALTER DATABASE.
connection_limit	A string, the "connlimit" parameter for postgreSQL ALTER DATABASE.
is_template	A string, the "istemplate" parameter for postgreSQL ALTER DATABASE.
rename_to	A string, the "new_name" parameter for postgreSQL ALTER DATABASE.
owner_to	A string, the "new_owner" parameter for postgreSQL ALTER DATABASE.
set_tablespace	A string, the "new_tablespace" parameter for postgreSQL ALTER DATABASE.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL ALTER DATABASE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
alterDATABASE("dbTest01", rename_to = "dbProd01")
```

---

alterSCHEMA *Generate a postgreSQL ALTER SCHEMA statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL ALTER SCHEMA statement, optionally execute the statement if con is not NULL.

Usage

```
alterSCHEMA(name, rename_to = NULL, owner_to = NULL, con = NULL)
```

Arguments

name	A string, the "name" parameter for postgreSQL ALTER SCHEMA.
rename_to	A string, the "new_name" parameter for postgreSQL ALTER SCHEMA.
owner_to	A string, the "new_owner" parameter for postgreSQL ALTER SCHEMA.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE SCHEMA statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
alterSCHEMA("dev", rename_to = "prod")
```

`alterTABLE` *Generate a postgreSQL ALTER TABLE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL ALTER TABLE statement, optionally execute the statement if con is not NULL.

Usage

```
alterTABLE(
  name,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  action,
  con = NULL
)
```

Arguments

`name`	A string, the "name" parameter for postgreSQL ALTER TABLE statement.
`if_exists`	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL ALTER TABLE statement.
`cascade`	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL ALTER TABLE statement.
`restrict`	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL ALTER TABLE statement.
`action`	A string, the "action" parameter for postgreSQL ALTER TABLE statement.
`con`	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL ALTER TABLE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
alterTABLE("sample", action = "ADD COLUMN IF NOT EXISTS col4 BOOLEAN")
```

callFUNCTION *Generate a postgreSQL statement to execute a function, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL statement to execute a function, optionally execute the statement if con is not NULL.

Usage

```
callFUNCTION(
  x = list(),
  schema = NULL,
  func,
  quote_text = TRUE,
  cast = TRUE,
  types,
  con = NULL
)
```

Arguments

x	A named list, names must match the parameter names of the SQL function, values are the values to set the parameters to when executing the SQL function.
schema	A string, the schema name of the SQL function.
func	A string, the name of the SQL function.
quote_text	TRUE/FALSE, if TRUE, calls quoteText() to add single quotes around character strings.
cast	TRUE/FALSE, if TRUE, will add SQL to cast the parameters to the specified type.
types	A vector of character strings specifying the SQL data types of the function parameters, the position of the type should match the position of the parameter for that type in x.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL statement to execute a function; or the results retrieved by DBI::dbGetQuery after executing the statement.

Examples

```
callFUNCTION(
x = list(a = 1, b = 2),
schema = NULL,
func = "sample_add",
quote_text = TRUE,
cast = FALSE,
types = c("INT", "INT")
)
```

`callPROCEDURE` *Generate a postgreSQL statement to execute a procedure, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL statement to execute a procedure, optionally execute the statement if con is not NULL.

Usage

```
callPROCEDURE(
  x = list(),
  schema = NULL,
  proc,
  quote_text = TRUE,
  cast = TRUE,
  types,
  con = NULL
)
```

Arguments

`x`	A named list, names must match the parameter names of the SQL procedure, values are the values to set the parameters to when executing the SQL procedure.
`schema`	A string, the schema name of the SQL procedure.
`proc`	A string, the name of the SQL procedure.
`quote_text`	TRUE/FALSE, if TRUE, calls quoteText() to add single quotes around character strings.
`cast`	TRUE/FALSE, if TRUE, will add SQL to cast the parameters to the specified type.
`types`	A vector of character strings specifying the SQL data types of the procedure parameters, the position of the type should match the position of the parameter for that type in x.
`con`	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL statement to execute a procedure; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
callPROCEDURE(
x = list(a = 1, b = 2),
schema = NULL,
proc = "sample_add",
quote_text = TRUE,
cast = FALSE,
types = c("INT", "INT")
)
```

`createDATABASE` *Generate a postgreSQL CREATE DATABASE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE DATABASE statement, optionally execute the statement if con is not NULL.

Usage

```
createDATABASE(
  name,
  owner = NULL,
  template = NULL,
  encoding = NULL,
  locale = NULL,
  lc_collate = NULL,
  lc_ctype = NULL,
  tablespace = NULL,
  allow_connections = NULL,
  connection_limit = NULL,
  is_template = NULL,
  con = NULL
)
```

Arguments

`name`	A string, the "name" parameter for postgreSQL CREATE DATABASE.
`owner`	A string, the "user_name" parameter for postgreSQL CREATE DATABASE.
`template`	A string, the "template" parameter for postgreSQL CREATE DATABASE.
`encoding`	A string, the "encoding" parameter for postgreSQL CREATE DATABASE.

locale	A string, the "locale" parameter for postgreSQL CREATE DATABASE
lc_collate	A string, the "lc_collate" parameter for postgreSQL CREATE DATABASE.
lc_ctype	A string, the "lc_ctype" parameter for postgreSQL CREATE DATABASE.
tablespace	A string, the "tablespace_name" parameter for postgreSQL CREATE DATABASE.
allow_connections	A string, the "allowconn" parameter for postgreSQL CREATE DATABASE.
connection_limit	A string, the "connlimit" parameter for postgreSQL CREATE DATABASE.
is_template	A string, the "istemplate" parameter for postgreSQL CREATE DATABASE.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE DATABASE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createDATABASE("dbTest01")
```

---

createEXTENSION *Generate a postgreSQL CREATE EXTENSION statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE EXTENSION statement, optionally execute the statement if con is not NULL.

Usage

```
createEXTENSION(
  name,
  if_not_exists = FALSE,
  schema = NULL,
  version = NULL,
  cascade = FALSE,
  con = NULL
)
```

Arguments

name	A string, the "extension_name" parameter for postgreSQL CREATE EXTENSION.
if_not_exists	TRUE/FALSE, if TRUE, adds "IF NOT EXISTS" to postgreSQL CREATE EXTENSION statement.
schema	A string, the "schema_name" parameter for postgreSQL CREATE EXTENSION.
version	A string, the "version" parameter for postgreSQL CREATE EXTENSION.
cascade	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL CREATE EXTENSION statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE EXTENSION statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createEXTENSION("pgcrypto")
```

---

createFUNCTION *Generate a postgreSQL CREATE FUNCTION statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE FUNCTION statement, optionally execute the statement if con is not NULL.

Usage

```
createFUNCTION(
  name,
  args = NULL,
  or_replace = FALSE,
  returns = NULL,
  returns_table = NULL,
  language = "SQL",
  definition,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL CREATE FUNCTION.
args	A named list, names are the argument names, values are strings with the argument data types.
or_replace	TRUE/FALSE, if TRUE, adds "OR REPLACE" to postgreSQL CREATE FUNCTION statement.
returns	A string, the "returns" parameter for postgreSQL CREATE FUNCTION.
returns_table	A named list, names are the return table column names, values are strings with the return table data types.
language	A string, the "language" parameter for postgreSQL CREATE FUNCTION.
definition	A string, the "definition" parameter for postgreSQL CREATE FUNCTION.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE PROCEDURE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createFUNCTION(
name = "sample_add",
args = list(a = "INTEGER", b = "INTEGER"),
returns = "INT",
language = "plpgsql",
definition = "BEGIN RETURN sample_add.a + sample_add.b; END;"
)
```

---

createPROCEDURE *Generate a postgreSQL CREATE PROCEDURE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE PROCEDURE statement, optionally execute the statement if con is not NULL.

Usage

```
createPROCEDURE(
  name,
  args = NULL,
  or_replace = FALSE,
  language = "SQL",
  definition,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL CREATE PROCEDURE.
args	A named list, names are the argument names, values are strings with the argument data types.
or_replace	TRUE/FALSE, if TRUE, adds "OR REPLACE" to postgreSQL CREATE PROCEDURE statement.
language	A string, the "language" parameter for postgreSQL CREATE PROCEDURE.
definition	A string, the "definition" parameter for postgreSQL CREATE PROCEDURE.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE PROCEDURE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createPROCEDURE(
name = "sample",
args = list(a = "INTEGER", b = "TEXT"),
definition = "INSERT INTO tbl(col1, col2) VALUES (a, b);"
)
```

---

createSCHEMA *Generate a postgreSQL CREATE SCHEMA statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE SCHEMA statement, optionally execute the statement if con is not NULL.

Usage

```
createSCHEMA(name, authorization = NULL, if_not_exists = FALSE, con = NULL)
```

Arguments

name	A string, the "schema_name" parameter for postgreSQL CREATE SCHEMA.
authorization	A string, the "role_specification" parameter for postgreSQL CREATE SCHEMA.
if_not_exists	TRUE/FALSE, if TRUE, adds "IF NOT EXISTS" to postgreSQL CREATE SCHEMA statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE SCHEMA statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createSCHEMA("dev")
```

`createTABLE` *Generate a postgreSQL CREATE TABLE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE TABLE statement, optionally execute the statement if con is not NULL.

Usage

```
createTABLE(
  name,
  columns,
  constraints = NULL,
  temporary = FALSE,
  if_not_exists = FALSE,
  con = NULL
)
```

Arguments

`name`	A string, the "table_name" parameter for postgreSQL CREATE TABLE.
`columns`	A named list, names are the SQL column names, values are strings with the SQL column data types, constraints, etc.
`constraints`	A named list, names are the SQL constraint names, values are strings with the SQL constraint.
`temporary`	TRUE/FALSE, if TRUE, adds "TEMPORARY" to postgreSQL CREATE TABLE statement.
`if_not_exists`	TRUE/FALSE, if TRUE, adds "IF NOT EXISTS" to postgreSQL CREATE TABLE statement.
`con`	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE TABLE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createTABLE(
name = "sample",
columns = list(col1 = "SERIAL NOT NULL", col2 = "INTEGER", col3 = "TEXT"),
constraints = list(sample_constraint = "UNIQUE(col3)")
)
```

`createTRIGGER` *Generate a postgreSQL CREATE TRIGGER statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL CREATE TRIGGER statement, optionally execute the statement if con is not NULL.

Usage

```
createTRIGGER(name, when, event, on, for_each_row = FALSE, func, con = NULL)
```

Arguments

`name`	A string, the "name" parameter for postgreSQL CREATE TRIGGER.
`when`	A string, the "when" parameter (BEFORE, AFTER, INSTEAD OF) for postgreSQL CREATE TRIGGER.
`event`	A string, the "event" parameter (INSERT/UPDATE/DELETE/TRUNCATE) for postgreSQL CREATE TRIGGER.
`on`	A string, the "table_name" parameter for postgreSQL CREATE TRIGGER.
`for_each_row`	TRUE/FALSE, if TRUE, adds "FOR EACH ROW" to postgreSQL CREATE TRIGGER statement.
`func`	A string, the function call to be executed by the postgreSQL CREATE TRIGGER.
`con`	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL CREATE TRIGGER statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
createTRIGGER(
name = "sample_trigger",
when = "AFTER",
event = "INSERT",
on = "sample_table",
for_each_row = TRUE,
func = "function_sample()"
)
```

DELETE *Generate a postgreSQL DELETE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DELETE statement, optionally execute the statement if con is not NULL.

Usage

```
DELETE(schema = NULL, table, where = NULL, con = NULL)
```

Arguments

schema	A string, the schema name of the SQL table to DELETE from.
table	A string, the table name of the SQL table to DELETE from.
where	A named list, names are the columns for comparison, values are lists with a comparison operator and a value the comparison operator will check against. ex: list(col1 = list(comparison = "=", value = quoteText("b")))
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DELETE statement; or the results retrieved by DBI::dbGetQuery after executing the statement.

Examples

```
DELETE(
schema = "test",
table = "table1",
where = list(
  col1 = list(comparison = "=", value = quoteText("b")),
  col2 = list(comparison = "IS", value = "NULL")
)
)
```

doubleQuoteText *Add double quotes to strings.*

Description

Add double quotes to strings.

Usage

```
doubleQuoteText(x, char_only = TRUE)
```

Arguments

x	A string.
char_only	TRUE/FALSE, if TRUE, adds quotes only if is.character(x) is TRUE.

Value

A string, with double quotes added.

Examples

```
doubleQuoteText("Sample quotes.")
```

dropDATABASE *Generate a postgreSQL DROP DATABASE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP DATABASE statement, optionally execute the statement if con is not NULL.

Usage

```
dropDATABASE(name, if_exists = FALSE, force = FALSE, con = NULL)
```

Arguments

name	A string, the "name" parameter for postgreSQL DROP DATABASE.
if_exists	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP DATABASE statement.
force	TRUE/FALSE, if TRUE, adds "FORCE" to postgreSQL DROP DATABASE statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP DATABASE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropDATABASE("dbTest01")
```

`dropEXTENSION` *Generate a postgreSQL DROP EXTENSION statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP EXTENSION statement, optionally execute the statement if con is not NULL.

Usage

```
dropEXTENSION(
  name,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

`name`	A string, the "name" parameter for postgreSQL DROP EXTENSION.
`if_exists`	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP EXTENSION statement.
`cascade`	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL DROP EXTENSION statement.
`restrict`	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL DROP EXTENSION statement.
`con`	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP EXTENSION statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropEXTENSION("pgcrypto")
```

dropFUNCTION *Generate a postgreSQL DROP FUNCTION statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP FUNCTION statement, optionally execute the statement if con is not NULL.

Usage

```
dropFUNCTION(
  name,
  args = NULL,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL DROP FUNCTION.
args	A named list, names are the argument names, values are strings with the argument data types.
if_exists	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP FUNCTION statement.
cascade	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL DROP FUNCTION statement.
restrict	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL DROP FUNCTION statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP FUNCTION statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropFUNCTION(
name = "sample",
args = list(a = "INTEGER", b = "TEXT")
)
```

dropPROCEDURE *Generate a postgreSQL DROP PROCEDURE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP PROCEDURE statement, optionally execute the statement if con is not NULL.

Usage

```
dropPROCEDURE(
  name,
  args = NULL,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL DROP PROCEDURE.
args	A named list, names are the argument names, values are strings with the argument data types.
if_exists	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP PROCEDURE statement.
cascade	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL DROP PROCEDURE statement.
restrict	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL DROP PROCEDURE statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP PROCEDURE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropPROCEDURE(
name = "sample",
args = list(a = "INTEGER", b = "TEXT")
)
```

dropSCHEMA *Generate a postgreSQL DROP SCHEMA statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP SCHEMA statement, optionally execute the statement if con is not NULL.

Usage

```
dropSCHEMA(
  name,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL DROP SCHEMA.
if_exists	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP SCHEMA statement.
cascade	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL DROP SCHEMA statement.
restrict	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL DROP SCHEMA statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP SCHEMA statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropSCHEMA("dev")
```

dropTABLE *Generate a postgreSQL DROP TABLE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP TABLE statement, optionally execute the statement if con is not NULL.

Usage

```
dropTABLE(
  name,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL DROP TABLE.
if_exists	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP TABLE statement.
cascade	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL DROP TABLE statement.
restrict	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL DROP TABLE statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP TABLE statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropTABLE("sample")
```

dropTRIGGER *Generate a postgreSQL DROP TRIGGER statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL DROP TRIGGER statement, optionally execute the statement if con is not NULL.

Usage

```
dropTRIGGER(
  name,
  on,
  if_exists = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

name	A string, the "name" parameter for postgreSQL DROP TRIGGER.
on	A string, the "table_name" parameter for postgreSQL DROP TRIGGER.
if_exists	TRUE/FALSE, if TRUE, adds "IF EXISTS" to postgreSQL DROP TRIGGER statement.
cascade	TRUE/FALSE, if TRUE, adds "CASCADE" to postgreSQL DROP TRIGGER statement.
restrict	TRUE/FALSE, if TRUE, adds "RESTRICT" to postgreSQL DROP TRIGGER statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DROP TRIGGER statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
dropTRIGGER(
name = "sample_trigger",
on = "sample_table"
)
```

INSERT *Generate a postgreSQL INSERT statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL INSERT statement, optionally execute the statement if con is not NULL.

Usage

```
INSERT(
  x,
  schema = NULL,
  table,
  prepare = TRUE,
  types,
  returning = NULL,
  quote_text = TRUE,
  cast = TRUE,
  con = NULL
)
```

Arguments

x	A data.table, column names must match the column names of the destination SQL table.
schema	A string, the schema name of the destination SQL table.
table	A string, the table name of the destination SQL table.
prepare	TRUE/FALSE, if TRUE, creates a postgreSQL prepared statement for inserting the data.
types	A vector of character strings specifying the SQL data types of the destination columns, the position of the type should match the position of the column for that type in x. Required if prepare or cast is TRUE.
returning	A vector of character strings specifying the SQL column names to be returned by the INSERT statement.
quote_text	TRUE/FALSE, if TRUE, calls quoteText() to add single quotes around character strings.
cast	TRUE/FALSE, if TRUE, will add SQL to cast the data to be inserted to the specified type.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL INSERT statement; or a string, postgreSQL prepared statement; or the results retrieved by DBI::dbGetQuery after executing the statement.

Examples

```
INSERT(
x = as.data.table(list(col1 = c("a", "b", "c"), col2 = c(1, 2, 3))),
schema = "test",
table = "table1",
prepare = TRUE,
types = c("TEXT", "INTEGER"),
returning = NULL,
quote_text = TRUE,
cast = TRUE
)
```

---

quoteText *Add single quotes to strings, useful for converting R strings into SQL formatted strings.*

Description

Add single quotes to strings, useful for converting R strings into SQL formatted strings.

Usage

```
quoteText(x, char_only = TRUE)
```

Arguments

x	A string.
char_only	TRUE/FALSE, if TRUE, adds quotes only if is.character(x) is TRUE.

Value

A string, with single quotes added to match postgreSQL string formatting.

Examples

```
quoteText("Sample quotes.")
```

`sampleStr` *Generates (pseudo)random strings of the specified char length. Used in INSERT and UPDATE to avoid overwriting existing columns of data.tables. Not as fast as stringi:stri_rand_strings.*

Description

Generates (pseudo)random strings of the specified char length. Used in INSERT and UPDATE to avoid overwriting existing columns of data.tables. Not as fast as stringi:stri_rand_strings.

Usage

```
sampleStr(char)
```

Arguments

`char`	A integer, the number of chars to include in the output string.

Value

A string.

Examples

```
sampleStr(10)
```

`sql_80_char_comment` *Add a 80 char SQL comment, intended to be used for visual breaks in documents.*

Description

Add a 80 char SQL comment, intended to be used for visual breaks in documents.

Usage

```
sql_80_char_comment()
```

Value

A string, 80 chars of "-".

Examples

```
sql_80_char_comment()
```

sql_comment *Add a single line SQL comment.*

Description

Add a single line SQL comment.

Usage

```
sql_comment(x)
```

Arguments

x	A string.

Value

A string prefixed with "–".

Examples

```
sql_comment("Sample single line comment.")
```

sql_script *Create a SQL script, optionally execute the statement if con is not NULL.*

Description

Create a SQL script, optionally execute the statement if con is not NULL.

Usage

```
sql_script(..., path = NULL, con = NULL)
```

Arguments

...	A string, SQL command to be combined into one document or statement.
path	A string, the file path (inlcude the file name) to save the script
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, SQL commands combined into one document or statement; or the results retrieved by DBI::dbSendQuery after executing the statement.

Examples

```
sql_script(
createSCHEMA("dev"),
sql_80_char_comment(),
createTABLE(name = "sample",
columns = list(col1 = "SERIAL NOT NULL", col2 = "INTEGER", col3 = "TEXT"),
constraints = list(sample_constraint = "UNIQUE(col3)")
))
```

TRUNCATE *Generate a postgreSQL TRUNCATE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL TRUNCATE statement, optionally execute the statement if con is not NULL.

Usage

```
TRUNCATE(
  schema = NULL,
  table,
  restart_identity = FALSE,
  continue_identity = FALSE,
  cascade = FALSE,
  restrict = FALSE,
  con = NULL
)
```

Arguments

schema	A string, the schema name of the SQL table to TRUNCATE.
table	A string, the table name of the SQL table to TRUNCATE.
restart_identity	TRUE/FALSE, if TRUE, will add RESTART IDENTITY to the statement.
continue_identity	TRUE/FALSE, if TRUE, will add CONTINUE IDENTITY to the statement.
cascade	TRUE/FALSE, if TRUE, will add CASCADE to the statement.
restrict	TRUE/FALSE, if TRUE, will add RESTRICT to the statement.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL DELETE statement; or the results retrieved by DBI::dbGetQuery after executing the statement.

Examples

```
TRUNCATE(
schema = "test",
table = "table1"
)
```

UPDATE *Generate a postgreSQL UPDATE statement, optionally execute the statement if con is not NULL.*

Description

Generate a postgreSQL UPDATE statement, optionally execute the statement if con is not NULL.

Usage

```
UPDATE(
  x,
  schema = NULL,
  table,
  where,
  prepare = TRUE,
  types,
  returning = NULL,
  quote_text = TRUE,
  cast = TRUE,
  con = NULL
)
```

Arguments

x	A named list, names must match the column names of the destination SQL table, values are the values to set the SQL records to.
schema	A string, the schema name of the destination SQL table.
table	A string, the table name of the destination SQL table.
where	A named list, names are the columns for comparison, values are lists with a comparison operator and a value the comparison operator will check against. ex: list(col1 = list(comparison = "=", value = quoteText("b")))
prepare	TRUE/FALSE, if TRUE, creates a postgreSQL prepared statement for inserting the data.
types	A vector of character strings specifying the SQL data types of the destination columns, the position of the type should match the position of the column for that type in x. Required if prepare or cast is TRUE.
returning	A vector of character strings specifying the SQL column names to be returned by the INSERT statement.

quote_text	TRUE/FALSE, if TRUE, calls quoteText() to add single quotes around character strings.
cast	TRUE/FALSE, if TRUE, will add SQL to cast the data to be inserted to the specified type.
con	A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

Value

A string, postgreSQL UPDATE statement; or a string, postgreSQL prepared statement; or the results retrieved by DBI::dbGetQuery after executing the statement.

Examples

```
UPDATE(
x = list(col1 = "a", col2 = 1),
schema = "test",
table = "table1",
where = list(
  col1 = list(comparison = "=", value = quoteText("b")),
  col2 = list(comparison = "IS", value = "NULL")
),
prepare = FALSE,
types = c("TEXT", "INTEGER"),
returning = c("col3"),
quote_text = TRUE,
cast = TRUE
)
```

Index