

# Package ‘sqlHelpers’

May 9, 2026

**Type** Package

**Title** Collection of 'SQL' Utilities for 'T-SQL' and 'Postgresql'

**Version** 0.1.2

**Description** Includes functions for interacting with common meta data fields,  
writing insert statements, calling functions, and more for 'T-SQL' and 'Postgresql'.

**License** GPL (>= 3)

**Encoding** UTF-8

**Depends** data.table, toolbox

**Imports** DBI, odbc, parallel, stringi

**RoxygenNote** 7.2.0

**NeedsCompilation** no

**Author** Timothy Conwell [aut, cre]

**Maintainer** Timothy Conwell <timconwell@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-10-14 19:50:02 UTC

## Contents

|  |    |
|--|----|
| call_function . . . . .                    | 2  |
| connect . . . . .                          | 3  |
| create_table_from_data_frame . . . . .     | 3  |
| drop_table . . . . .                       | 4  |
| fetch_columns . . . . .                    | 5  |
| fetch_function_definition . . . . .        | 5  |
| fetch_function_output_parameters . . . . . | 6  |
| fetch_function_parameters . . . . .        | 7  |
| fetch_tables . . . . .                     | 7  |
| insert_batch_chunker . . . . .             | 8  |
| insert_values . . . . .                    | 8  |
| quoteText2 . . . . .                       | 10 |
| sqlizeNames . . . . .                      | 11 |

|  |    |
|--|----|
| sqlizeTypes . . . . .                  | 11 |
| truncate_table . . . . .               | 12 |
| t_sql_bulk_insert . . . . .            | 12 |
| t_sql_script_create_table . . . . .    | 13 |
| t_sql_script_proc_definition . . . . . | 14 |
| update_values . . . . .                | 14 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>16</b> |
|--------------|-----------|

---

|               |                                       |
|---------------|---------------------------------------|
| call_function | <i>Call a SQL function/procedure.</i> |
|---------------|---------------------------------------|

---

### Description

Call a SQL function/procedure.

### Usage

```
call_function(con, schema, function_name, args, dialect = "T-SQL", cast = TRUE)
```

### Arguments

|               |  |
|---------------|--|
| con           | A database connection.   |
| schema        | A string, the schema to query.   |
| function_name | A string, the function/procedure to query.   |
| args          | A named list or vector, names are the parameter names and values are the parameter values. |
| dialect       | A string, "T-SQL" or "Postgresql".   |
| cast          | TRUE/FALSE, if TRUE, will add SQL to cast the parameters to the specified type.            |

### Value

A data.table.

### Examples

```
call_function(con = NULL)
```

---

|         |  |
|---------|--|
| connect | <i>Connect to a database using a connection string via DBI/odbc.</i> |
|---------|--|

---

**Description**

Connect to a database using a connection string via DBI/odbc.

**Usage**

```
connect(
  con_str = "Driver={PostgreSQL ANSI};Host=localhost;Port=5432;Database=postgres;"
)
```

**Arguments**

con\_str            A database connection string.

**Value**

A database connection.

**Examples**

```
connect(NULL)
```

---

|                              |  |
|------------------------------|--|
| create_table_from_data_frame |  |
|------------------------------|--|

*Generate a CREATE TABLE statement based on a data.frame, optionally execute the statement if con is not NULL.*

---

**Description**

Generate a CREATE TABLE statement based on a data.frame, optionally execute the statement if con is not NULL.

**Usage**

```
create_table_from_data_frame(x, table_name, con = NULL)
```

**Arguments**

x                    A data.frame.  
table\_name           A string, the name of the SQL table to create.  
con                   A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.

**Value**

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

**Examples**

```
create_table_from_data_frame(x = iris, table_name = "test")
```

---

|            |  |
|------------|--|
| drop_table | <i>Generate a DROP TABLE statement, optionally execute the statement if con is not NULL.</i> |
|------------|--|

---

**Description**

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

**Usage**

```
drop_table(args, con = NULL)
```

**Arguments**

|      |   |
|------|---|
| args | A string, the arguments to add to the DROP TABLE statement.                   |
| con  | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery. |

**Value**

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

**Examples**

```
drop_table("sample")
```

---

|               |   |
|---------------|---|
| fetch_columns | <i>Retrieve the columns/types in a table.</i> |
|---------------|---|

---

**Description**

Retrieve the columns/types in a table.

**Usage**

```
fetch_columns(con, schema, table)
```

**Arguments**

|        |                                |
|--------|--------------------------------|
| con    | A database connection.         |
| schema | A string, the schema to query. |
| table  | A string, the table to query.  |

**Value**

A data.table.

**Examples**

```
fetch_columns(con = NULL)
```

---

|                           |   |
|---------------------------|---|
| fetch_function_definition | <i>Retrieve the definition of a function/procedure.</i> |
|---------------------------|---|

---

**Description**

Retrieve the definition of a function/procedure.

**Usage**

```
fetch_function_definition(con, schema, function_name, type = "FUNCTION")
```

**Arguments**

|               |  |
|---------------|--|
| con           | A database connection.                     |
| schema        | A string, the schema to query.             |
| function_name | A string, the function/procedure to query. |
| type          | A string, "FUNCTION" or "PROCEDURE".       |

**Value**

A data.table.

**Examples**

```
fetch_function_definition(con = NULL)
```

---

```
fetch_function_output_parameters
```

*Retrieve the output parameters of a function/procedure.*

---

**Description**

Retrieve the output parameters of a function/procedure.

**Usage**

```
fetch_function_output_parameters(con, schema, function_name, type = "FUNCTION")
```

**Arguments**

|               |  |
|---------------|--|
| con           | A database connection.                     |
| schema        | A string, the schema to query.             |
| function_name | A string, the function/procedure to query. |
| type          | A string, "FUNCTION" or "PROCEDURE".       |

**Value**

A data.table.

**Examples**

```
fetch_function_output_parameters(con = NULL)
```

---

`fetch_function_parameters`*Retrieve the input parameters of a function/procedure.*

---

**Description**

Retrieve the input parameters of a function/procedure.

**Usage**

```
fetch_function_parameters(con, schema, function_name, type = "FUNCTION")
```

**Arguments**

|                            |  |
|----------------------------|--|
| <code>con</code>           | A database connection.                     |
| <code>schema</code>        | A string, the schema to query.             |
| <code>function_name</code> | A string, the function/procedure to query. |
| <code>type</code>          | A string, "FUNCTION" or "PROCEDURE".       |

**Value**

A data.table.

**Examples**

```
fetch_function_parameters(con = NULL)
```

---

`fetch_tables`*Retrieve the tables in a schema*

---

**Description**

Retrieve the tables in a schema

**Usage**

```
fetch_tables(con, schema)
```

**Arguments**

|                     |                                |
|---------------------|--------------------------------|
| <code>con</code>    | A database connection.         |
| <code>schema</code> | A string, the schema to query. |

**Value**

A data.table.

**Examples**

```
fetch_tables(con = NULL)
```

---

```
insert_batch_chunker
```

*Helper function for INSERT*

---

**Description**

Helper function for INSERT

**Usage**

```
insert_batch_chunker(x, n_batches, batch_size)
```

**Arguments**

|            |   |
|------------|---|
| x          | A vector of data to insert.                               |
| n_batches  | Integer, the number of batches needed to insert the data. |
| batch_size | Integer, the size of each batch.                          |

**Value**

A list.

**Examples**

```
insert_batch_chunker(c(1, 2, 3), 1, 100)
```

---

```
insert_values
```

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

---

**Description**

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

**Usage**

```
insert_values(
  x = NULL,
  schema = NULL,
  table,
  returning = NULL,
  quote_text = TRUE,
  cast = TRUE,
  types = NULL,
  batch_size = 1000,
  con = NULL,
  table_is_temporary = FALSE,
  retain_insert_order = FALSE,
  n_cores = 1,
  connect_db_name = NULL,
  dialect = "T-SQL"
)
```

**Arguments**

|                     |   |
|---------------------|---|
| x                   | A list, data.frame or data.table, 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.  |
| 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.                    |
| types               | A vector of types to use for casting columns. If blank, will look at meta data about table to decide types. |
| batch_size          | Integer, the maximum number of records to submit in one statement.  |
| con                 | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.                               |
| table_is_temporary  | TRUE/FALSE, if TRUE, prevents parallel processing.  |
| retain_insert_order | TRUE/FALSE, if TRUE, prevents parallel processing.  |
| n_cores             | A integer, the number of cores to use for parallel forking (passed to parallel::mclapply as mc.cores).      |
| connect_db_name     | The name of the database to pass to connect() when inserting in parallel.                                   |
| dialect             | A string, "T-SQL" or "Postgresql".  |

**Value**

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

**Examples**

```
insert_values(  
  x = list(col1 = c("a", "b", "c"), col2 = c(1, 2, 3)),  
  schema = "test",  
  table = "table1",  
  types = c("VARCHAR(12)", "INT")  
)
```

---

`quoteText2`

*Add single quotes to strings using `stringi::stri_join`, useful for converting R strings into SQL formatted strings.*

---

**Description**

Add single quotes to strings using `stringi::stri_join`, useful for converting R strings into SQL formatted strings.

**Usage**

```
quoteText2(x, char_only = TRUE, excluded_chars = c("NULL"))
```

**Arguments**

`x` A string.  
`char_only` TRUE/FALSE, if TRUE, adds quotes only if `is.character(x)` is TRUE.  
`excluded_chars` A character vector, will not add quotes if a value is in `excluded_chars`.

**Value**

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

**Examples**

```
quoteText2("Sample quotes.")
```

---

|             |  |
|-------------|--|
| sqlizeNames | <i>Convert a column name into a SQL compatible name.</i> |
|-------------|--|

---

**Description**

Convert a column name into a SQL compatible name.

**Usage**

```
sqlizeNames(x, dialect = "T-SQL")
```

**Arguments**

|         |                                    |
|---------|------------------------------------|
| x       | A string, a column name.           |
| dialect | A string, "T-SQL" or "Postgresql". |

**Value**

A string, a SQL compatible column name.

**Examples**

```
sqlizeNames("column 100 - sample b")
```

---

|             |   |
|-------------|---|
| sqlizeTypes | <i>Get the equivalent SQL data type for a given R object.</i> |
|-------------|---|

---

**Description**

Get the equivalent SQL data type for a given R object.

**Usage**

```
sqlizeTypes(x, dialect = "T-SQL")
```

**Arguments**

|         |                                    |
|---------|------------------------------------|
| x       | A R object.                        |
| dialect | A string, "T-SQL" or "Postgresql". |

**Value**

A string, the equivalent SQL data type for x.

**Examples**

```
sqlizeTypes(100.1209)
```

---

|                |  |
|----------------|--|
| truncate_table | <i>Generate a TRUNCATE TABLE statement, optionally execute the statement if con is not NULL.</i> |
|----------------|--|

---

**Description**

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

**Usage**

```
truncate_table(args, con = NULL)
```

**Arguments**

|      |   |
|------|---|
| args | A string, the arguments to add to the TRUNCATE TABLE statement.               |
| con  | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery. |

**Value**

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

**Examples**

```
truncate_table(args = "table1")
```

---

|                   |   |
|-------------------|---|
| t_sql_bulk_insert | <i>Generate a BULK INSERT statement, optionally execute the statement if con is not NULL.</i> |
|-------------------|---|

---

**Description**

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

**Usage**

```
t_sql_bulk_insert(file, schema = NULL, table, con = NULL, ...)
```

**Arguments**

|        |   |
|--------|---|
| file   | A string, the file path to the file with data to insert.                      |
| schema | A string, the schema name of the destination SQL table.                       |
| table  | A string, the table name of the destination SQL table.                        |
| con    | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery. |
| ...    | named arguments are passed to the WITH statement.                             |

**Value**

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

**Examples**

```
t_sql_bulk_insert(  
  file = tempfile(),  
  schema = "test",  
  table = "table1",  
  format = 'CSV',  
  first_row = 2,  
)
```

---

t\_sql\_script\_create\_table

*Generate a CREATE TABLE statement for an existing table in Microsoft SQL Server.*

---

**Description**

Generate a CREATE TABLE statement for an existing table in Microsoft SQL Server.

**Usage**

```
t_sql_script_create_table(con, table)
```

**Arguments**

|       |   |
|-------|---|
| con   | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery. |
| table | A string, the schema qualified table name of an existing SQL table.           |

**Value**

A data table, contains the DDL scripts for creating a table.

**Examples**

```
t_sql_script_create_table(con = NULL)
```

---

t\_sql\_script\_proc\_definition

*Fetch the object definition of a proc in Microsoft SQL Server.*

---

### Description

Fetch the object definition of a proc in Microsoft SQL Server.

### Usage

```
t_sql_script_proc_definition(con, proc)
```

### Arguments

|      |   |
|------|---|
| con  | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.               |
| proc | A string, the database and schema qualified table name of an existing SQL stored procedure. |

### Value

A string, contains the script for defining a stored procedure.

### Examples

```
t_sql_script_proc_definition(con = NULL)
```

---

update\_values

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

---

### Description

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

### Usage

```
update_values(
  x,
  schema = NULL,
  table,
  where = NULL,
  returning = NULL,
  quote_text = TRUE,
  cast = TRUE,
  types = NULL,
  con = NULL,
  dialect = "T-SQL"
)
```

**Arguments**

|            |   |
|------------|---|
| x          | A list, data.frame or data.table, 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.  |
| where      | A string, conditions to add to a WHERE statement.   |
| returning  | A vector of character strings specifying the SQL column names to be returned by the UPDATE 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.                    |
| types      | A vector of types to use for casting columns. If blank, will look at meta data about table to decide types. |
| con        | A database connection that can be passed to DBI::dbSendQuery/DBI::dbGetQuery.                               |
| dialect    | A string, "T-SQL" or "Postgresql".  |

**Value**

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

**Examples**

```
update_values(  
  x = list(col1 = c("a"), col2 = c(1)),  
  schema = "test",  
  table = "table1",  
  where = "1=1",  
  types = c("VARCHAR(12)", "INT")  
)
```

# Index

`call_function`, [2](#)  
`connect`, [3](#)  
`create_table_from_data_frame`, [3](#)  
  
`drop_table`, [4](#)  
  
`fetch_columns`, [5](#)  
`fetch_function_definition`, [5](#)  
`fetch_function_output_parameters`, [6](#)  
`fetch_function_parameters`, [7](#)  
`fetch_tables`, [7](#)  
  
`insert_batch_chunker`, [8](#)  
`insert_values`, [8](#)  
  
`quoteText2`, [10](#)  
  
`sqlizeNames`, [11](#)  
`sqlizeTypes`, [11](#)  
  
`t_sql_bulk_insert`, [12](#)  
`t_sql_script_create_table`, [13](#)  
`t_sql_script_proc_definition`, [14](#)  
`truncate_table`, [12](#)  
  
`update_values`, [14](#)