

# Package ‘sqlhelper’

March 26, 2026

**Title** Easier 'SQL' Integration

**Version** 0.2.2

**Description**

Execute files of 'SQL' and manage database connections. 'SQL' statements and queries may be interpolated with string literals. Execution of individual statements and queries may be controlled with keywords. Multiple connections may be defined with 'YAML' and accessed by name.

**Depends** R (>= 4.1.0)

**Imports** DBI, yaml, rappdirs, stringr, glue, pool, methods, tibble, tidy, purrr (>= 1.0.0), sf, rlang

**License** GPL (>= 3)

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Suggests** dplyr, rmarkdown, knitr, testthat (>= 3.0.0), odbc, RSQLite, RPostgres, RMariaDB, bigrquery, spData

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** <https://majerr.github.io/sqlhelper/dev/>,  
<https://github.com/majerr/sqlhelper/>

**BugReports** <https://github.com/majerr/sqlhelper/issues>

**NeedsCompilation** no

**Author** Matthew Roberts [aut, cre, cph]

**Maintainer** Matthew Roberts <matthew@zsmr.uk>

**Repository** CRAN

**Date/Publication** 2026-03-26 14:50:02 UTC

## Contents

config_examples . . . . .	2
connect . . . . .	3

connection_info . . . . .	4
default_conn . . . . .	5
disconnect . . . . .	6
is_connected . . . . .	6
live_connection . . . . .	7
prepare_sql . . . . .	8
read_sql . . . . .	9
run_files . . . . .	11
run_queries . . . . .	13
set_default_conn_name . . . . .	15

<b>Index</b>	<b>17</b>
--------------	-----------

---

config_examples	<i>Examples of yaml configurations for database connections</i>
-----------------	---

---

## Description

Provides example configurations for several databases and a range of options

## Usage

```
config_examples(filename = NA)
```

## Arguments

filename      A string. If supplied, examples are written to a file with this name.

## Details

Irrespective of whether a filename is supplied, yaml configuration examples will be returned as a single string.

## Value

A yaml string of database configuration examples.

## Examples

```
config_examples()

# write the examples to a temporary file called 'examples.yaml'
config_examples(file.path(tempdir(), "examples.yaml"))
```

---

connect	<i>(Re-)Establish connections to databases</i>
---------	--

---

### Description

Closes any open connections, reads config files as directed by `config_filename` and `exclusive`, and creates new connections from the descriptions in those files.

### Usage

```
connect(config_filename = NA, exclusive = FALSE)
```

### Arguments

<code>config_filename</code>	String. The full name and path of a configuration file, or "site", or "user", or "example", or NA (the default). Cannot be NA if <code>exclusive = TRUE</code> .
<code>exclusive</code>	Logical. If TRUE, the file named by <code>config_filename</code> is treated as the only config file. Site and user level files are not read. This parameter is ignored if <code>config_filename</code> is missing.

### Details

If `exclusive=FALSE` (the default), configuration files will be sought in the directory returned by `rappdirs::site_config_dir()`, the directory returned by `rappdirs::user_config_dir()`, and finally a file named by `config_filename` (if not NA). If elements of those files conflict, later files overwrite the elements of earlier files.

If `exclusive=TRUE`, only 1 file, indicated by the `config_filename` parameter, will be read.

- If `config_filename = "site"`, a config file called `sqlhelper_db_conf.yml` will be sought in the directory returned by `rappdirs::site_config_dir()`
- If `config_filename = "user"`, a config file called `sqlhelper_db_conf.yml` will be sought in the directory returned by `rappdirs::user_config_dir()`
- If `config_filename` is not NULL (but not "site" or "user"), it is assumed to name a file.

A warning is raised if no valid configurations are found (e.g. `connect()` is called without arguments and no site- or user-wide files are present, or the connections in those files are invalid)

`vignette("connections")` explains how to write a config file and how to access the created connections.

### Value

NULL, invisibly

**Examples**

```
library(sqlhelper)

example_filename <- system.file("examples",
                                "sqlhelper_db_conf.yml",
                                package = "sqlhelper")

# Search for config files in rappdirs::site_config_dir(),
# rappdirs::user_config_dir(), and read from example_filename
connect(example_filename)

# Read only the named example file
connect(example_filename, exclusive=TRUE)
```

---

connection\_info      *Browse available connections*

---

**Description**

Provides information about created connections.

**Usage**

```
connection_info(name_str = ".*", exact = TRUE)
```

**Arguments**

name_str	A regular expression to be used to identify connection names to include. The default ('.*') returns all of them.
exact	TRUE or FALSE. Should name_str match the name of a connection exactly? TRUE will identify only 1 connection if name_str does not contain any metacharacters

**Value**

Null, or a tibble with 1 row per identified connection and the following fields:

**name** identifier (character)  
**description** a description of the connection, if found in the conf file (character)  
**live** is this connection valid and live? (logical)  
**driver** the name of the driver function (character)  
**conn\_str** the string used to parameterize the connection (character)  
**pool** is this a pool connection? (logical)

If no connection names matched name\_str, the tibble will be empty. If no connections have been configured (e.g. connect() has not been called), NULL is returned.

**Examples**

```
library(sqlhelper)

connect(
  system.file(
    "examples/sqlhelper_db_conf.yml",
    package="sqlhelper"
  ),
  exclusive=TRUE
)

connection_info()

connection_info("pool_sqlite")
```

---

default_conn	<i>Return the default connection</i>
--------------	--------------------------------------

---

**Description**

A convenience wrapper around `live_connection()` and `get_default_conn_name()`

**Usage**

```
default_conn()
```

**Value**

A database connection returned by `DBI::dbConnect()` or `pool::dbPool()`

**Examples**

```
library(sqlhelper)

connect(
  system.file(
    "examples/sqlhelper_db_conf.yml",
    package="sqlhelper"
  ),
  exclusive=TRUE
)

default_conn()
```

---

disconnect	<i>Close all connections and remove them from the connections cache</i>
------------	---

---

**Description**

Close all connections and remove them from the connections cache

**Usage**

```
disconnect()
```

**Value**

NULL, invisibly

**Examples**

```
library(sqlhelper)
connect(
  system.file("examples",
              "sqlhelper_db_conf.yml",
              package="sqlhelper")
)
disconnect()
```

---

is_connected	<i>Test whether a database is connected</i>
--------------	---

---

**Description**

Test whether a database is connected

**Usage**

```
is_connected(conn_name)
```

```
not_connected(conn_name)
```

**Arguments**

conn\_name      Character. The name of a connection (run [connection\\_info\(\)](#) for options)

**Value**

Logical, or NULL if conn\_name does not identify exactly 1 connection

## Examples

```
library(sqlhelper)

connect(
  system.file("examples/sqlhelper_db_conf.yml",
             package="sqlhelper")
)
connection_info()

is_connected("simple_sqlite")
is_connected("foo")
DBI::dbDisconnect(live_connection("simple_sqlite"))
is_connected("simple_sqlite")
not_connected("simple_sqlite")
disconnect()
is_connected("simple_sqlite")
not_connected("simple_sqlite")
```

---

live_connection	<i>Return the named connection or NULL</i>
-----------------	--

---

## Description

Return the named connection or NULL

## Usage

```
live_connection(conn_name)
```

## Arguments

conn_name	Chr. The name of the live connection you want (use <a href="#">connection_info</a> to get names of available connections).
-----------	--

## Value

A live connection to a database, or NULL, invisibly, if conn\_name is not the name of a live connection

## Examples

```
library(sqlhelper)
connect(
  system.file("examples/sqlhelper_db_conf.yml",
             package="sqlhelper")
)
connection_info()
```

```

conn <- live_connection("simple_sqlite")
conn

DBI::dbDisconnect(conn)
is.null(live_connection("simple_sqlite"))
is.null(live_connection("foo"))

```

---

```
prepare_sql
```

---

*prepare queries and assemble meta data prior to execution*

---

### Description

Except for `sql`, parameters are default values to be used when none are supplied in `sql` (i.e. when `sql` is a tibble returned by `read_sql()`).

### Usage

```

prepare_sql(
  sql,
  quotesql = "yes",
  values = parent.frame(),
  execmethod = "get",
  geometry = NA,
  default.conn = default_conn()
)

```

### Arguments

<code>sql</code>	An optionally-named list or character vector containing <code>sql</code> commands, or a tibble returned by <code>read_sql()</code>
<code>quotesql</code>	"yes" or "no" - should interpolated characters be quoted by default? Anything that isn't "no" is treated as "yes".
<code>values</code>	An environment containing variables to interpolate into the SQL. Pass any object that is not an environment (commonly-used options include "no", NA, FALSE or NULL) if interpolation is to be skipped, or another environment containing values to interpolate to avoid using <code>.GlobalEnv</code> .
<code>execmethod</code>	One of "get", "execute", "sendq", "sends" or "spatial" - which method should be used to execute the query? "get" means <code>DBI::dbGetQuery()</code> ; "execute" means <code>DBI::dbExecute()</code> ; "sendq" means <code>DBI::dbSendQuery</code> ; "sends" means <code>DBI::dbSendStatement()</code> ; "spatial" means <code>sf::st_read()</code> .
<code>geometry</code>	If <code>execmethod</code> is "spatial", which column contains the geometry? Ignored if <code>execmethod</code> is not "spatial".
<code>default.conn</code>	Either the name of a sqlhelper connection, or a database connection returned by <code>DBI::dbConnect()</code> or <code>pool::pool()</code> , or NA. This connection is only used by <code>glue::glue_sql()</code> to quote SQL interpolations; <code>prepare_sql()</code> does not execute any SQL code.

**Details**

The default `.conn` parameter may be used to supply a connection object that is not a configured sqlhelper connection which can then be used to interpolate quoted strings.

**Value**

A tibble containing 1 row per query with the following fields:

**qname** character. A name for this query

**quotesql** "yes" or "no". Should parameterized character values be quoted for this query?

**interpolate** "yes" or "no". Should this query be parameterized with values from R?

**execmethod** The method to execute this query. One of "get" (`DBI::dbGetQuery()`), "execute" (`DBI::dbExecute()`), "sendq" (`DBI::dbSendQuery()`), "sends" (`DBI::dbSendStatement()`) or "spatial" (`sf::st_read()`)

**geometry** character. If `execmethod` is "spatial", which is the geometry column?

**conn\_name** character. The name of the database connection to use for this query. Must be the name of a configured sqlhelper connection.

**sql** The sql query as entered

**filename** The value of `file_name`

**prepared\_sql** The sql query to be executed, i.e. with interpolations and quoting in place

**Examples**

```
library(sqlhelper)
connect(
  system.file("examples/sqlhelper_db_conf.yml",
             package="sqlhelper"),
  exclusive = TRUE
)

n <- 5
foo <- 'bar'
prepped <- prepare_sql(c("select {`foo`}", "select {n}"))
prepped
prepped$prepared_sql
```

---

read\_sql

*Read a sql file and return it's contents as a tibble*


---

**Description**

Read a sql file and return it's contents as a tibble

**Usage**

```
read_sql(file_name, cascade = TRUE)
```

**Arguments**

<code>file_name</code>	Full name and path of a file to read
<code>cascade</code>	Parameters for executing each query may be specified as comments in the SQL file. If <code>cascade=TRUE</code> , execution parameters specified in the file will be cascaded to subsequent queries where that parameter is not specified. This enables you to set a parameter (e.g. the connection name) once, for the first query in a file, and use it for all the subsequent queries.

**Details**

Multiple SQL queries in files should be terminated by semi-colons (;), as usual.

The values of `qname`, `quotesql`, `interpolate`, `execmethod`, `geometry`, and `conn_name` in the output may be specified with comments immediately preceding each query (see examples).

With the exception of `qname`, the value of each interpreted comment is cascaded to subsequent queries (assuming `cascade=TRUE`). This means you may set values once for the first query in the file and they will apply to all the queries thereafter.

See `run_queries()` for the implications of setting execution parameters. See `prepare_sql()` for the treatment of missing values in the output and their defaults. The article `vignette("execution")` has further examples of using these parameters to control execution.

**Value**

A tibble containing 1 row per query with the following fields:

**qname** character. A name for this query

**quotesql** "yes" or "no". Should parameterized character values be quoted for this query?

**interpolate** "yes" or "no". Should this query be parameterized with values from R?

**execmethod** The method to execute this query. One of "get" (`DBI::dbGetQuery()`), "execute" (`DBI::dbExecute()`), "sendq" (`DBI::dbSendQuery()`), "sends" (`DBI::dbSendStatement()`) or "spatial" (`sf::st_read()`)

**geometry** character. If `execmethod` is "spatial", which is the geometry column?

**conn\_name** character. The name of the database connection to use for this query. Must be the name of a configured sqlhelper connection.

**sql** The sql query to be executed

**filename** The value of `file_name`

**Examples**

```
library(sqlhelper)

fn <- system.file( "examples/read_sql_execution_params.SQL",
                  package="sqlhelper" )
readLines( fn ) |> writelines()

sql_tibble <- read_sql(fn)
sql_tibble
```

```

sql_tibble$sql

fn <- system.file( "examples/read_sql_comments.SQL", package="sqlhelper" )
readLines( fn ) |> writelines()

sql_tibble <- read_sql(fn)
sql_tibble
sql_tibble$sql

```

---

run\_files

*Read, prepare and execute .SQL files*


---

### Description

Accepts a character vector of SQL file names and attempts to execute the queries in each one.

### Usage

```
run_files(filenamees, ..., include_params = FALSE)
```

```
runfiles(filenamees, ..., include_params = FALSE)
```

### Arguments

filenamees      name, or vector of names, of file(s) to be executed  
...              Arguments to be passed to [run\\_queries\(\)](#), [prepare\\_sql\(\)](#), or [read\\_sql\(\)](#)  
include\_params   TRUE or FALSE. Should the parameters be included in the output?

### Details

If no default connection is supplied via `default.conn` and no connections have been configured using `connect()`, an attempt will be made to configure connections via `connect()` using the configuration search path. If no database connections are available after this attempt, an error will be raised. See `vignette("connections")` for details about the configuration search path.

`run_files()` calls `read_sql()` on each file, and `prepare_sql()` on the queries read from those files. Prepared queries are executed with `run_queries()`. The behaviour of those functions can be controlled by passing the relevant parameters to `run_files()` as the `...` argument.

`run_files()` also enables control of the arguments accepted by `run_queries()` on a per-query basis, by interpreting comments in SQL files as described for `read_sql()`. Interpreted comments precede the sql query to which they refer. Each interpretable comment must be on a line by itself and take the form:

```
-- keyword = value
```

Keywords and possible values for interpretable comments are:

**qname** A name for this query

**quotesql** "yes" or "no" - should interpolated characters be quoted?

**interpolate** "yes" or "no" - should sql be interpolated?

**execmethod** One of "get", "execute", "sendq", "sends" or "spatial" - which method should be used to execute the query? "get" means `DBI::dbGetQuery()`; "execute" means `DBI::dbExecute()`; "sendq" means `DBI::dbSendQuery`; "sends" means `DBI::dbSendStatement()`; "spatial" means `sf::st_read()`.

**geometry** The name of a spatial column. Ignored if `execmethod` is not 'spatial'

**conn\_name** The name of a connection to execute this query against

All interpreted comments except `qname` are cascaded *within their file*, meaning that if you want to use the same values throughout, you need only set them for the first query. See `read_sql()` for details.

## Value

A list of results of sql queries found in files

## See Also

[read\\_sql\(\)](#), [prepare\\_sql\(\)](#)

Other SQL runners: [run\\_queries\(\)](#)

## Examples

```
library(sqlhelper)

config_filename <- system.file("examples/sqlhelper_db_conf.yml",
                               package="sqlhelper")

readLines( config_filename ) |> writeLines()

connect(
  config_filename,
  exclusive=TRUE)

DBI::dbWriteTable( default_conn(), "iris", iris)

sf::st_write(spData::congruent, default_conn(), "congruent")
sf::st_write(spData::incongruent, live_connection("pool_sqlite"), "incongruent")

run_files_ex1 <- system.file("examples/run_files_ex1.sql", package="sqlhelper")
readLines( run_files_ex1 ) |> writeLines()

run_files_ex2 <- system.file("examples/run_files_ex2.sql", package="sqlhelper")
readLines( run_files_ex2 ) |> writeLines()

n_longest_petals <- 5
results <- run_files( c( run_files_ex1, run_files_ex2 ) )
```

```

names(results)

results$how_many_irises

results$n_longest_setosa_petal_lengths

plot(results$get_congruent, border = "orange")
plot(results$get_incongruent, border = "blue", add=TRUE)

```

---

run_queries	<i>Execute a sequence of SQL queries</i>
-------------	--

---

### Description

Accepts a character vector of SQL queries and attempts to execute each

### Usage

```

run_queries(sql, ..., default.conn = default_conn(), include_params = FALSE)

runqueries(sql, ..., default.conn = default_conn(), include_params = FALSE)

```

### Arguments

sql	An optionally-named list or character vector containing sql strings, or a tibble returned by <a href="#">read_sql()</a> or <a href="#">prepare_sql()</a> .
...	Arguments to be passed to <a href="#">read_sql()</a> or <a href="#">prepare_sql()</a>
default.conn	Either the name of a sqlhelper connection, or a database connection returned by <a href="#">DBI::dbConnect()</a> or <a href="#">pool::dbPool()</a> . This connection is used as a fall-back when the sql parameter is a tibble and no per-query connection name is supplied, or the connection name is default (see <a href="#">prepare_sql()</a> ). It may be used by <a href="#">glue::glue_sql()</a> to interpolate SQL strings, and as the connection against which to execute SQL queries.
include_params	TRUE or FALSE. Should the parameters be included in the output? Mainly useful for debugging.

### Details

If no default connection is supplied via `default.conn` and no connections have been configured using `connect()`, an attempt will be made to configure connections via `connect()` using the configuration search path. If no database connections are available after this attempt, an error will be raised. See `vignette("connections")` for details about the configuration search path.

**Value**

- If `include_params` is `FALSE` and the `sql` argument is a vector, a list containing the results of each query; element names will be taken from the `sql` argument.
- If the length of the `sql` argument is 1 and is not named, the result of that query is returned as-is (e.g. a `data.frame`), not as a 1-element list.
- If `include_params` is `TRUE`, a tibble is returned containing 1 row per query with the following fields:

**qname** character. A name for this query

**quotesql** "yes" or "no". Should parameterized character values be quoted for this query?

**interpolate** "yes" or "no". Should this query be parameterized with values from R?

**execmethod** The method to execute this query. One of "get" (`DBI::dbGetQuery()`), "execute" (`DBI::dbExecute()`), "sendq" (`DBI::dbSendQuery()`), "sends" (`DBI::dbSendStatement()`) or "spatial" (`sf::st_read()`)

**geometry** character. If `execmethod` is "spatial", this should be the name of the geometry column.

**conn\_name** character. The name of the database connection against which to execute this query. Must be the name of a configured `sqlhelper` connection.

**sql** The sql query to be executed

**filename** The value of `file_name`

**prepared\_sql** The sql query to be executed, i.e. with interpolations and quoting in place

**result** The result of the query

**See Also**

[read\\_sql\(\)](#), [prepare\\_sql\(\)](#)

Other SQL runners: [run\\_files\(\)](#)

**Examples**

```
library(sqlhelper)

readLines(
  system.file("examples/sqlhelper_db_conf.yml",
             package="sqlhelper")
) |>
writeLines()

connect(
  system.file("examples/sqlhelper_db_conf.yml", package="sqlhelper"),
  exclusive=TRUE)

DBI::dbWriteTable( default_conn(),
                  "iris",
                  iris)

n <- 5
```

```

run_queries(
  c(top_n = "select * from iris limit {n}",
    uniqs = "select distinct species as species from iris")
)

## use include_params to review the execution context
run_queries(
  c(top_n = "select * from iris limit {n}",
    uniqs = "select distinct species as species from iris"),
  include_params = TRUE
)

## pass an env of interpolation values to the 'values' parameter
## result of a single, unnamed query is returned as an object, not a
## 1-element list
e <- new.env()
e$n <- 2
run_queries(
  "select * from iris limit {n}",
  values = e
)

## Use the execmethod parameter for statements
run_queries("create table iris_setosa as select * from iris where species = 'setosa'",
  execmethod = 'execute')

run_queries("select distinct species as species from iris_setosa")

```

---

set\_default\_conn\_name *Set/get the name of the default connection to use*

---

### Description

Set/get the name of the default connection to use

### Usage

```
set_default_conn_name(conn_name)
```

```
get_default_conn_name()
```

### Arguments

conn\_name      Character string. The name a connection

### Value

get returns the name of the default connection; set returns NULL, invisibly.

**Examples**

```
library(sqlhelper)
connect(
  system.file("examples/sqlhelper_db_conf.yml",
             package="sqlhelper"),
  exclusive = TRUE
)

connection_info()

get_default_conn_name()

set_default_conn_name("pool_sqlite")

connection_info()

get_default_conn_name()
```

# Index

## \* SQL runners

- run\_files, 11
- run\_queries, 13
  
- config\_examples, 2
- connect, 3
- connection\_info, 4, 7
- connection\_info(), 6
  
- DBI::dbConnect(), 8, 13
- DBI::dbExecute(), 8–10, 12, 14
- DBI::dbGetQuery(), 8–10, 12, 14
- DBI::dbSendQuery, 8
- DBI::dbSendQuery(), 9, 10, 14
- DBI::dbSendStatement(), 8–10, 12, 14
- default\_conn, 5
- disconnect, 6
  
- get\_default\_conn\_name  
    (set\_default\_conn\_name), 15
- glue::glue\_sql(), 8, 13
  
- is\_connected, 6
  
- live\_connection, 7
  
- not\_connected (is\_connected), 6
  
- pool::dbPool(), 13
- pool::pool(), 8
- prepare\_sql, 8
- prepare\_sql(), 8, 10–14
  
- rappdirs::site\_config\_dir(), 3
- rappdirs::user\_config\_dir(), 3
- read\_sql, 9
- read\_sql(), 8, 11–14
- run\_files, 11, 14
- run\_files(), 11
- run\_queries, 12, 13
- run\_queries(), 10, 11
  
- runfiles (run\_files), 11
- runqueries (run\_queries), 13
  
- set\_default\_conn\_name, 15
- sf::st\_read(), 8–10, 12, 14