

# Package ‘qryflow’

June 8, 2026

**Title** Execute Multi-Step 'SQL' Workflows

**Version** 0.4.0

**Description** Execute multi-step 'SQL' workflows by leveraging specially formatted comments to define and control execution. This enables users to mix queries, commands, and metadata within a single script. Results are returned as named objects for use in downstream workflows.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Imports** DBI

**Suggests** knitr, rmarkdown, RSQLite, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**VignetteBuilder** knitr

**URL** <https://christian-million.github.io/qryflow/>,  
<https://github.com/christian-million/qryflow/>

**BugReports** <https://github.com/christian-million/qryflow/issues>

**NeedsCompilation** no

**Author** Christian Million [aut, cre, cph]

**Maintainer** Christian Million <christianmillion93@gmail.com>

**Depends** R (>= 4.1.0)

**Repository** CRAN

**Date/Publication** 2026-06-07 23:50:02 UTC

## Contents

collapse_sql_lines . . . . .	2
example_db_connect . . . . .	3
example_sql_path . . . . .	3

extract_all_tags . . . . .	4
is_tag_line . . . . .	5
ls_qryflow_types . . . . .	6
new_qryflow_chunk . . . . .	6
qryflow . . . . .	7
qryflow_execute . . . . .	8
qryflow_handler_exists . . . . .	10
qryflow_meta . . . . .	10
qryflow_parse . . . . .	11
qryflow_results . . . . .	12
qryflow_run . . . . .	13
read_sql_lines . . . . .	14
register_qryflow_type . . . . .	15
validate_qryflow_handler . . . . .	16

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

---

collapse_sql_lines	<i>Collapse SQL lines into single character</i>
--------------------	---

---

## Description

A thin wrapper around `paste0(x, collapse = '\\n')` to standardize the way qryflow collapses SQL lines.

## Usage

```
collapse_sql_lines(x)
```

## Arguments

`x` character vector of SQL lines

## Value

a character vector of length 1

## Examples

```
path <- example_sql_path()
lines <- read_sql_lines(path)
sql <- collapse_sql_lines(lines)
```

---

example\_db\_connect      *Create an example in-memory database*

---

### Description

This function creates a connection to an in-memory SQLite database, with the option to add a table to the database. This function is intended to facilitate examples, vignettes, and package tests.

### Usage

```
example_db_connect(df = NULL)
```

### Arguments

df                      Optional data.frame to add to the database.

### Value

connection from [DBI::dbConnect\(\)](#)

### Examples

```
con <- example_db_connect(mtcars)

x <- DBI::dbGetQuery(con, "SELECT * FROM mtcars;")

head(x)

DBI::dbDisconnect(con)
```

---

example\_sql\_path      *Get path to qryflow example SQL scripts*

---

### Description

qryflow provides example SQL scripts in its `inst/sql` directory. Use this function to retrieve the path to an example script. This function is intended to facilitate examples, vignettes, and package tests.

### Usage

```
example_sql_path(path = "mtcars.sql")
```

### Arguments

path                    filename of the example script.

**Value**

path to example SQL script

**Examples**

```
path <- example_sql_path("mtcars.sql")
file.exists(path)
```

---

extract_all_tags	<i>Extract tagged metadata from a SQL chunk</i>
------------------	---

---

**Description**

extract\_all\_tags() scans SQL for specially formatted comment tags (e.g., -- @tag: value) and returns them as a named list. This is exported with the intent to be useful for users extending qryflow. It's typically used against a single SQL chunk, such as one parsed from a .sql file.

**Usage**

```
extract_all_tags(text)

subset_tags(tags, keep, negate = FALSE)
```

**Arguments**

text	A character vector of SQL lines or a file path to a SQL script.
tags	A named list of tags, typically from extract_all_tags(). Used in subset_tags().
keep	A character vector of tag names to keep or exclude in subset_tags().
negate	Logical; if TRUE, subset_tags() returns all tags except those listed in keep.

**Value**

- extract\_all\_tags(): A named list of all tags found in the SQL chunk.
- subset\_tags(): A filtered named list of tags or NULL if none remain.

**See Also**

[qryflow\\_parse\(\)](#), [ls\\_qryflow\\_types\(\)](#)

**Examples**

```
filepath <- example_sql_path('mtcars.sql')
parsed <- qryflow_parse(filepath)

chunk <- parsed[[1]]
tags <- extract_all_tags(chunk$sql)
subset_tags(tags, keep = c("query"))
```

---

is_tag_line	<i>Detect the presence of a properly structured tagline</i>
-------------	---

---

## Description

Checks whether a specially structured comment line is formatted in the way that qryflow expects.

## Usage

```
is_tag_line(line)
```

## Arguments

line            A character vector to check. It is a vectorized function.

## Details

Tag lines should look like this: -- @key: value

- Begins with an inline comment (--)
- An @ precedes a tag type (e.g., type, name, query, exec) and is followed by a colon (:)
- A value is provided

## Value

Logical. Indicating whether each line matches tag specification.

## Examples

```
a <- "-- @query: df_mtcars"
b <- "-- @exec: prep_tbl"
c <- "-- @type: query"

lines <- c(a, b, c)

is_tag_line(lines)
```

---

ls_qryflow_types	<i>List currently registered chunk types</i>
------------------	--

---

**Description**

Helper function to access the names of the currently registered chunk types.

**Usage**

```
ls_qryflow_types()
```

**Value**

Character vector of registered chunk types

**Examples**

```
ls_qryflow_types()
```

---

new_qryflow_chunk	<i>Create an instance of the qryflow_chunk class</i>
-------------------	--

---

**Description**

Create an instance of the qryflow\_chunk class

**Usage**

```
new_qryflow_chunk(
  type = character(),
  name = character(),
  sql = character(),
  tags = NULL,
  results = NULL,
  meta = init_meta()
)
```

**Arguments**

type	Character indicating the type of chunk (e.g., "query", "exec")
name	Name of the chunk
sql	SQL statement associated with chunk
tags	Optional, additional tags included in chunk
results	Optional, filled in after chunk execution
meta	Optional, stores meta data on the object

**Details**

Exported for users intending to extend qryflow. Subsequent processes rely on the structure of a qryflow\_chunk.

**Value**

An list-like object of class qryflow\_chunk

**Examples**

```
chunk <- new_qryflow_chunk("query", "df_name", "SELECT * FROM mtcars;")
```

---

 qryflow

---

*Run a multi-step SQL workflow and return query results*


---

**Description**

qryflow() is high level convenience function. It executes a SQL workflow defined in a tagged .sql script or character string and returns query results as R objects.

The SQL script can contain multiple-steps (chunks), each tagged with @query or @exec. Query results are captured and returned as a named list, where names correspond to the @query tags.

**Usage**

```
qryflow(
  con,
  sql,
  ...,
  on_error = c("stop", "warn", "collect"),
  verbose = getOption("qryflow.verbose", FALSE),
  simplify = TRUE,
  default_type = getOption("qryflow.default_type", "query")
)
```

**Arguments**

con	A database connection from <code>DBI::dbConnect()</code>
sql	A file path to a .sql workflow or a character string containing SQL code.
...	Additional arguments passed to <code>qryflow_run()</code> or <code>qryflow_results()</code> .
on_error	Controls behaviour when a chunk fails during execution. One of "stop" (default), "warn", or "collect". "stop" halts execution immediately and returns the partially executed workflow. "warn" records the error in the chunk's meta, signaling immediately. "collect" gathers all errors from across all chunks and reports them at the end.

verbose	Logical. If TRUE, emits a message before each chunk identifying its name and type, and prints a summary on completion reporting total chunks run, successes, errors, and elapsed time. Defaults to FALSE. The global default can be set with <code>options(qryflow.verbose = TRUE)</code> .
simplify	Logical; if TRUE (default), a list of length 1 is simplified to the single result object.
default_type	The default chunk type (defaults to "query"). The global default can be set with <code>options(qryflow.default_type = 'query')</code> .

### Details

This is a wrapper around the combination of `qryflow_run()`, which always provides a list of results and metadata, and `qryflow_results()`, which filters the output of `qryflow_run()` to only include the results of the SQL.

### Value

A named list of query results, or a single result if `simplify = TRUE` and only one chunk exists.

### See Also

`qryflow_run()`, `qryflow_results()`

### Examples

```
con <- example_db_connect(mtcars)
filepath <- example_sql_path("mtcars.sql")

results <- qryflow(con, filepath)

head(results$df_mtcars)

DBI::dbDisconnect(con)
```

---

qryflow_execute	<i>Execute a parsed qryflow SQL workflow</i>
-----------------	--

---

### Description

`qryflow_execute()` takes a `qryflow` object (as returned by `qryflow_parse()`), executes each chunk (e.g., `@query`, `@exec`), and collects the results and timing metadata.

This function is used internally by `qryflow_run()`, but can be called directly in concert with `qryflow_parse()` if you want to manually control parsing and execution.

**Usage**

```

qryflow_execute(
  con,
  x,
  ...,
  on_error = c("stop", "warn", "collect"),
  verbose = getOption("qryflow.verbose", FALSE)
)

```

**Arguments**

con	A database connection from <a href="#">DBI::dbConnect()</a>
x	A qryflow object, typically created by <a href="#">qryflow_parse()</a>
...	Reserved for future use
on_error	Controls behaviour when a chunk fails during execution. One of "stop" (default), "warn", or "collect". "stop" halts execution immediately and returns the partially executed workflow. "warn" records the error in the chunk's meta, signaling immediately. "collect" gathers all errors from across all chunks and reports them at the end.
verbose	Logical. If TRUE, emits a message before each chunk identifying its name and type, and prints a summary on completion reporting total chunks run, successes, errors, and elapsed time. Defaults to FALSE. The global default can be set with <code>options(qryflow.verbose = TRUE)</code> .

**Value**

An object of class `qryflow`, containing executed chunks with results and a meta attribute that includes timing and source information.

**See Also**

[qryflow\\_run\(\)](#), [qryflow\\_parse\(\)](#)

**Examples**

```

con <- example_db_connect(mtcars)

filepath <- example_sql_path("mtcars.sql")

parsed <- qryflow_parse(filepath)

executed <- qryflow_execute(con, parsed)

DBI::dbDisconnect(con)

```

qryflow\_handler\_exists

*Check existence of a given handler in the registry*

---

### **Description**

Checks whether the specified handler exists in the handler registry environment.

### **Usage**

```
qryflow_handler_exists(type)
```

### **Arguments**

type                    chunk type to check (e.g., "query", "exec")

### **Value**

Logical. Does type exist in the handler registry?

### **Examples**

```
qryflow_handler_exists("query")
```

---

qryflow\_meta

*Extract metadata from qryflow objects*

---

### **Description**

Extract metadata from qryflow objects

### **Usage**

```
qryflow_meta(x)
```

### **Arguments**

x                        qryflow or qryflow\_chunk object

**Examples**

```

con <- example_db_connect(mtcars)

filepath <- example_sql_path("mtcars.sql")

parsed <- qryflow_parse(filepath)
qryflow_meta(parsed)
qryflow_meta(parsed[[1]])

results <- qryflow_execute(con, parsed)
qryflow_meta(results)
qryflow_meta(results[[1]])

DBI::dbDisconnect(con)

```

---

qryflow\_parse

*Parse a SQL workflow into tagged chunks*


---

**Description**

qryflow\_parse() reads a SQL file or character vector and parses it into discrete chunks based on @query, @exec, and other custom markers.

**Usage**

```

qryflow_parse(
  sql,
  ...,
  default_type = getOption("qryflow.default_type", "query")
)

```

**Arguments**

sql	A file path to a SQL workflow file, or a character vector containing SQL lines.
...	Reserved for future use.
default_type	The default chunk type (defaults to "query"). The global default can be set with options(qryflow.default_type = "query").

**Details**

This function is used internally by [qryflow\\_run\(\)](#), but can also be used directly to preprocess or inspect the structure of a SQL workflow.

**Value**

An object of class qryflow, which is a structured list of SQL chunks and metadata.

**See Also**

[qryflow\(\)](#), [qryflow\\_run\(\)](#), [qryflow\\_execute\(\)](#)

**Examples**

```
filepath <- example_sql_path("mtcars.sql")
parsed <- qryflow_parse(filepath)
```

---

qryflow_results	<i>Extract results from a qryflow_workflow object</i>
-----------------	---

---

**Description**

`qryflow_results()` retrieves the results from a list returned by [qryflow\\_run\(\)](#), typically one that includes parsed and executed SQL chunks.

**Usage**

```
qryflow_results(x, ..., simplify = FALSE)
```

**Arguments**

<code>x</code>	Results from <a href="#">qryflow_run()</a> , usually containing a mixture of <code>qryflow_chunk</code> objects.
<code>...</code>	Reserved for future use.
<code>simplify</code>	Logical; if TRUE, simplifies the result to a single object if only one query chunk is present. Defaults to FALSE.

**Value**

A named list of query results, or a single result object if `simplify = TRUE` and only one result is present.

**See Also**

[qryflow\(\)](#), [qryflow\\_run\(\)](#)

**Examples**

```
con <- example_db_connect(mtcars)
filepath <- example_sql_path("mtcars.sql")
obj <- qryflow_run(con, filepath)
results <- qryflow_results(obj)
DBI::dbDisconnect(con)
```

---

qryflow\_run

*Parse and execute a tagged SQL workflow*


---

### Description

qryflow\_run() reads a SQL workflow from a file path or character string, parses it into tagged statements, and executes those statements against a database connection.

This function might be preferable for those who want a qryflow execution to consistently return a qryflow object. Whereas the qryflow() function may return a list or other objects, depending on the arguments, qryflow\_run() always returns a qryflow object. Results can be extracted using qryflow\_results().

### Usage

```
qryflow_run(
  con,
  sql,
  ...,
  on_error = c("stop", "warn", "collect"),
  verbose = getOption("qryflow.verbose", FALSE),
  default_type = getOption("qryflow.default_type", "query")
)
```

### Arguments

con	A database connection from <code>DBI::dbConnect()</code>
sql	A character string representing either the path to a .sql file or raw SQL content.
...	Additional arguments passed to <code>qryflow_execute()</code> .
on_error	Controls behaviour when a chunk fails during execution. One of "stop" (default), "warn", or "collect". "stop" halts execution immediately and returns the partially executed workflow. "warn" records the error in the chunk's meta, signaling immediately. "collect" gathers all errors from across all chunks and reports them at the end.
verbose	Logical. If TRUE, emits a message before each chunk identifying its name and type, and prints a summary on completion reporting total chunks run, successes, errors, and elapsed time. Defaults to FALSE. The global default can be set with <code>options(qryflow.verbose = TRUE)</code> .
default_type	The default chunk type (defaults to "query"). The global default can be set with <code>options(qryflow.default_type = 'query')</code> .

### Value

A qryflow object representing the evaluated workflow, containing query results, execution meta-data, or both, depending on the contents of the SQL script.

**See Also**

[qryflow\(\)](#), [qryflow\\_results\(\)](#), [qryflow\\_execute\(\)](#), [qryflow\\_parse\(\)](#)

**Examples**

```
con <- example_db_connect(mtcars)

filepath <- example_sql_path("mtcars.sql")

obj <- qryflow_run(con, filepath)

obj$df_mtcars$sql
obj$df_mtcars$results

results <- qryflow_results(obj)

head(results$df_mtcars$results)

DBI::dbDisconnect(con)
```

---

read\_sql\_lines

*Standardizes lines read from string, character vector, or file*

---

**Description**

This is a generic function to ensure lines read from a file, a single character vector, or already parsed lines return the same format. This helps avoid re-reading entire texts by enabling already read lines to pass easily.

This is useful for folks who may want to extend qryflow.

**Usage**

```
read_sql_lines(x)
```

**Arguments**

x                    a filepath or character vector containing SQL

**Value**

A `qryflow_sql` object (inherits from `character`) with a length equal to the number of lines read

## Examples

```
# From a file #####
path <- example_sql_path()
read_sql_lines(path)

# From a single string #####
sql <- "SELECT *
FROM mtcars;"
read_sql_lines(sql)

# From a character #####
lines <- c("SELECT *", "FROM mtcars;")
read_sql_lines(lines)
```

---

register\_qryflow\_type *Register custom chunk types*

---

## Description

Use this function to register a custom chunk type with qryflow

## Usage

```
register_qryflow_type(type, handler, overwrite = FALSE)
```

## Arguments

type	Character indicating the chunk type (e.g., "exec", "query")
handler	A function to execute the SQL associated with the type. Must accept arguments "chunk", "con", and "...".
overwrite	Logical. Overwrite existing handler, if exists?

## Details

To avoid manually registering your custom type each session, consider adding the registration code to your .Rprofile or creating a package that leverages [.onLoad\(\)](#)

## Value

Logical. Indicating whether types were successfully registered.

**Examples**

```
# Create custom handler #####
custom_handler <- function(con, chunk, ...){
  # Custom execution code will go here...
  # return(result)
}

register_qryflow_type("query-send", custom_handler, overwrite = TRUE)
```

---

validate\_qryflow\_handler

*Ensure correct handler structure*

---

**Description**

This function checks that the passed object is a function and contains the arguments "con", "chunk", and "..." - in that order. This is to help ensure users only register valid handlers.

**Usage**

```
validate_qryflow_handler(handler)
```

**Arguments**

handler            object to check

**Value**

Logical. Generates an error if the object does not pass all the criteria.

**Examples**

```
custom_func <- function(con, chunk, ...){

  # Parsing Code Goes Here

}

validate_qryflow_handler(custom_func)
```

# Index

`.onLoad()`, 15

`collapse_sql_lines`, 2

`DBI::dbConnect()`, 3, 7, 9, 13

`example_db_connect`, 3

`example_sql_path`, 3

`extract_all_tags`, 4

`is_tag_line`, 5

`ls_qryflow_types`, 6

`ls_qryflow_types()`, 4

`new_qryflow_chunk`, 6

`qryflow`, 7

`qryflow()`, 12, 14

`qryflow_execute`, 8

`qryflow_execute()`, 12–14

`qryflow_handler_exists`, 10

`qryflow_meta`, 10

`qryflow_parse`, 11

`qryflow_parse()`, 4, 8, 9, 14

`qryflow_results`, 12

`qryflow_results()`, 7, 8, 14

`qryflow_run`, 13

`qryflow_run()`, 7–9, 11, 12

`read_sql_lines`, 14

`register_qryflow_type`, 15

`subset_tags (extract_all_tags)`, 4

`validate_qryflow_handler`, 16