Package 'sqltargets'

June 20, 2024

Type Package

Title 'Targets' Extension for 'SQL' Queries

Version 0.1.0

Maintainer David Ranzolin <daranzolin@gmail.com>

Description Provides an extension for 'SQL' queries as separate file within 'targets' pipelines. The shorthand creates two targets, the query file and the query result.

License MIT + file LICENSE

Encoding UTF-8

Imports cli, DBI, fs, glue, purrr, readr, rlang, stringr, tarchetypes, targets, withr

URL https://github.com/daranzolin/sqltargets

BugReports https://github.com/daranzolin/sqltargets/issues

RoxygenNote 7.2.1

Suggests testthat (>= 3.0.0), RSQLite (>= 2.2.0)

Config/testthat/edition 3

NeedsCompilation no

Author David Ranzolin [aut, cre, cph]

Repository CRAN

Date/Publication 2024-06-20 19:40:02 UTC

Contents

`sqltargets` sqltargets *package*

Description

targets extension for SQL files

Details

See the README on GitHub

`sqltargets_option_get` *Get or Set sqltargets Options*

Description

Get or Set sqltargets Options

Usage

```
sqltargets_option_get(option_name)

sqltargets_option_set(option_name, option_value)
```

Arguments

`option_name`	Character. Option name. See Details.
`option_value`	Value to assign to option ‘x‘.

Details

Available Options

- ‘"sqltargets.glue_sql_opening_delimiter"‘ - character. Length 1. Two characters. The opening delimiter passed to ‘glue::glue_sql()‘. - ‘"sqltargets.glue_sql_closing_delimiter"‘ - character. Length 1. Two characters. The closing delimiter passed to ‘glue::glue_sql()‘.

Value

No return value, called for side effects

tar_sql *Target with a SQL query.*

Description

Shorthand to include a SQL query in a ‘targets‘ pipeline.

Usage

```
tar_sql(
  name,
  path,
  query_params = list(),
  format = targets::tar_option_get("format"),
  tidy_eval = targets::tar_option_get("tidy_eval"),
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue")
)
```

Arguments

`name`	Symbol, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `set.seed()` on the result to locally recreate the target's initial RNG state.
`path`	Character of length 1 to the single ‘*.sql‘ source file to be executed. Defaults to the working directory of the ‘targets‘ pipeline.
`query_params`	Code, can be ‘NULL‘. ‘query_params‘ evaluates to a named list of parameters that are passed via ‘glue::glue_sql()‘. The list is quoted (not evaluated until the target runs) so that upstream targets can serve as parameter values.

`format` Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.

`tidy_eval` Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.

`repository` Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

`iteration` Character of length 1, name of the iteration mode of the target. Choices:

- `"vector"`: branching happens with `vctrs::vec_slice()` and aggregation happens with `vctrs::vec_c()`.
- `"list"`, branching happens with `[[]]` and aggregation happens with `list()`.
- `"group"`: `dplyr::group_by()`-like functionality to branch over subsets of a data frame. The target's return value must be a data frame with a special `tar_group` column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the `tar_group()` function to see how you can create the special `tar_group` column with `dplyr::group_by()`.

`error` Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.)
- `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.

`memory` Character of length 1, memory strategy. If `"persistent"`, the target stays in memory until the end of the pipeline (unless `storage` is `"worker"`, in which case `targets` unloads the value from memory right after storing it in order to avoid sending copious data over a network). If `"transient"`, the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. `format = "file"` with `repository = "aws"`), this memory strategy applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

`garbage_collection`
: Logical, whether to run `base::gc()` just before the target runs.

`deployment`
: Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If `"worker"`, the target builds on a parallel worker. If `"main"`, the target builds on the host machine / process managing the pipeline.

`priority`
: Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get built earlier (and polled earlier in `tar_make_future()`).

`resources`
: Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.

`storage`
: Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

  - `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
  - `"worker"`: the worker saves/uploads the value.
  - `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).
    If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.
    The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

`retrieval`
: Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

  - `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target builds.
  - `"worker"`: the worker loads the targets dependencies.
  - `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

`cue`
: An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

Details

‘tar_sql()‘ is an alternative to ‘tar_target()‘ for SQL queries that depend on upstream targets. The SQL source files (‘*.sql‘ files) should mention dependency targets with ‘tar_load()‘ within SQL comments (’–’). (Do not use ‘tar_load_raw()‘ or ‘tar_read_raw()‘ for this.) Then, ‘tar_sql()‘ defines a special kind of target. It 1. Finds all the ‘tar_load()‘/‘tar_read()‘ dependencies in the query and inserts them into the target’s command. This enforces the proper dependency relationships. (Do not use ‘tar_load_raw()‘ or ‘tar_read_raw()‘ for this.) 2. Sets ‘format = "file"‘ (see ‘tar_target()‘) so ‘targets‘ watches the files at the returned paths and reruns the query if those files change. 3. Creates another upstream target to watch the query file for changes ’<target name> ‘sqltargets_option_get("sqltargets.target_file_suffix")‘’.

Value

A data frame

Examples

```
targets::tar_dir({  # tar_dir() runs code from a temporary directory.
  # Unparameterized SQL query:
  lines <- c(
    "-- !preview conn=DBI::dbConnect(RSQLite::SQLite())",
    "-- targets::tar_load(data1)",
    "-- targets::tar_load(data2)",
    "select 1 AS my_col",
    ""
  )
  # In tar_dir(), not part of the user's file space:
  writeLines(lines, "query.sql")
  # Include the query in a pipeline as follows.
  targets::tar_script({
    library(tarchetypes)
    library(sqltargets)
    list(
      tar_sql(query, path = "query.sql")
    )
  }, ask = FALSE)
})
```

`tar_sql_deps` *List SQL query dependencies.*

Description

List the target dependencies of one or more SQL queries.

Usage

```
tar_sql_deps(path)
```

Arguments

path	Character vector, path to one or more SQL queries.

Value

Character vector of the names of targets that are dependencies of the SQL query.

Examples

```
lines <- c(
  "-- !preview conn=DBI::dbConnect(RSQLite::SQLite())",
  "-- targets::tar_load(data1)",
  "-- targets::tar_read(data2)",
  "select 1 as my_col",
  ""
)
query <- tempfile()
writeLines(lines, query)
tar_sql_deps(query)
```

---

tar_sql_raw *Target with a SQL query.*

Description

Shorthand to include a SQL query in a ‘targets‘ pipeline.

Usage

```
tar_sql_raw(
  name,
  path = ".",
  query_params = query_params,
  format = format,
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = targets::tar_option_get("garbage_collection"),
  deployment = "main",
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  deps = deps
)
```

Arguments

`name`	Character of length 1, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`. In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `set.seed()` on the result to locally recreate the target's initial RNG state.
`path`	Character of length 1 to the single '*.sql' source file to be executed. Defaults to the working directory of the 'targets' pipeline.
`query_params`	A named list of parameters for parameterized queries.
`format`	Optional storage format for the target's return value. With the exception of `format = "file"`, each target gets a file in `_targets/objects`, and each format is a different way to save and load this file. See the "Storage formats" section for a detailed list of possible data storage formats.
`error`	Character of length 1, what to do if the target stops and throws an error. Options: • `"stop"`: the whole pipeline stops and throws an error. • `"continue"`: the whole pipeline keeps going. • `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.) • `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline.
`memory`	Character of length 1, memory strategy. If `"persistent"`, the target stays in memory until the end of the pipeline (unless `storage` is `"worker"`, in which case `targets` unloads the value from memory right after storing it in order to avoid sending copious data over a network). If `"transient"`, the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value. For cloud-based dynamic files (e.g. `format = "file"` with `repository = "aws"`), this memory strategy applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.
`garbage_collection`	Logical, whether to run `base::gc()` just before the target runs.
`deployment`	Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. If `"worker"`, the target builds on a parallel worker. If `"main"`, the target builds on the host machine / process managing the pipeline.

`priority`	Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get built earlier (and polled earlier in `tar_make_future()`).
`resources`	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
`retrieval`	Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values: • `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target builds. • `"worker"`: the worker loads the targets dependencies. • `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.
`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
`deps`	Optional character vector of the adjacent upstream dependencies of the target, including targets and global objects. If `NULL`, dependencies are resolved automatically as usual.

Details

‘tar_sql()‘ is an alternative to ‘tar_target()‘ for SQL queries that depend on upstream targets. The SQL source files (‘*.sql‘ files) should mention dependency targets with ‘tar_load()‘ within SQL comments (’–’). (Do not use ‘tar_load_raw()‘ or ‘tar_read_raw()‘ for this.) Then, ‘tar_sql()‘ defines a special kind of target. It 1. Finds all the ‘tar_load()‘/‘tar_read()‘ dependencies in the query and inserts them into the target's command. This enforces the proper dependency relationships. (Do not use ‘tar_load_raw()‘ or ‘tar_read_raw()‘ for this.) 2. Sets ‘format = "file"‘ (see ‘tar_target()‘) so ‘targets‘ watches the files at the returned paths and reruns the query if those files change. 3. Creates another upstream target to watch the query file for changes ’<target name> ‘sqltargets_option_get("sqltargets.target_file_suffix")‘’.

Value

A data frame

Examples

```
targets::tar_dir({  # tar_dir() runs code from a temporary directory.
  # Unparameterized SQL query:
  lines <- c(
    "-- !preview conn=DBI::dbConnect(RSQLite::SQLite())",
    "-- targets::tar_load(data1)",
    "-- targets::tar_load(data2)",
    "select 1 AS my_col",
    ""
  )
  # In tar_dir(), not part of the user's file space:
```

```
  writeLines(lines, "query.sql")
  # Include the query in a pipeline as follows.
  targets::tar_script({
    library(tarchetypes)
    library(sqltargets)
    list(
      tar_sql(query, path = "query.sql")
    )
  }, ask = FALSE)
})
```

Index