Package 'commons'

August 7, 2026

Title AI Agents for Data Analysis

Version 0.0.1

Description Implements large language model agents for answering questions about database tables. Agents combine 'DBI' database connections with table and column metadata from a data dictionary (<https://data-dict.tidyverse.org/>), using registered calculations when applicable and generating database queries otherwise. They can also be embedded in a 'Shiny' application.

License MIT + file LICENSE

URL https://github.com/posit-dev/commons, https://posit-dev.github.io/commons/

BugReports https://github.com/posit-dev/commons/issues

Imports cli, DBI, ellmer (>= 0.4.1), jsonlite, knitr, rlang (>= 1.1.0), roxygen2, S7, utils

Suggests bsicons, bslib, dbplyr, dplyr, duckdb, rmarkdown, shiny, shinychat, testthat (>= 3.0.0), withr, yaml

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Config/roxygen2/version 8.0.0

Config/Needs/website tidyverse/tidytemplate

NeedsCompilation no

Author Simon Couch [aut, cre] (ORCID: <https://orcid.org/0000-0001-5676-5107>), Sara Altman [aut], Garrick Aden-Buie [ctb] (ORCID: <https://orcid.org/0000-0002-7111-0077>), Carson Sievert [ctb] (ORCID: <https://orcid.org/0000-0002-4958-2844>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Simon Couch <simon.couch@posit.co>

Repository CRAN

Date/Publication 2026-08-07 15:50:02 UTC

Contents

`commons` *Create a commons agent*

Description

`commons()` builds an ellmer::Chat with a system prompt that describes the available data and tools for searching measures and context, inspecting tables, and running SQL queries.

Usage

```
commons(
  client,
  data_sources,
  ...,
  semantic_layer = NULL,
 system_prompt = ellmer::interpolate_file(system.file("prompts/system-prompt.md",
    package = "commons"), date = Sys.Date())
)
```

Arguments

`client`	An ellmer::Chat giving the provider and model to use, e.g. `ellmer::chat_anthropic()`. Any system prompt or tools already set on the client are ignored, with a warning; pass a system prompt to `system_prompt` instead.
`data_sources`	A `data_source()`, or a named list of them. Measures can take a source's connection as an argument named after the source; see `semantic_layer()`. When there are several sources, the `run_sql` and `describe_table` tools take a source's name as a `source` argument.
`...`	These dots are for future extensions and must be empty.
`semantic_layer`	An optional `semantic_layer()`.
`system_prompt`	The agent's system prompt, as a single string. The default loads the markdown prompt shipped with commons and interpolates its `{{date}}` keyword. To customize it, copy the file into your project and interpolate the edited version:

```
file.copy(
  system.file("prompts/system-prompt.md", package = "commons"),
  "system-prompt.md"
)
commons(
  # ...
  system_prompt = ellmer::interpolate_file(
    "system-prompt.md",
    date = Sys.Date()
  )
)
```

Pass values for any `{{keyword}}` tokens you add as arguments to `ellmer::interpolate_file()`. commons appends the table and data dictionary documentation, so omit it from the file.

Details

`client` supplies the provider and model. commons replaces the client's system prompt and tools. Use `agent$chat()` to ask questions or `shinychat::chat_mod_ui()` and `shinychat::chat_mod_server()` to embed the agent in a Shiny app.

The agent first searches its `semantic_layer()` for a matching measure. If one matches, the agent runs a calculation defined by your data team. If no measure matches, it reads the data documentation, inspects the relevant tables, and writes a SQL query.

Value

An ellmer::Chat carrying commons' system prompt and tools.

See Also

`data_source()` for database connections and table documentation, and `semantic_layer()` for governed calculations.

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  con <- DBI::dbConnect(duckdb::duckdb())
  DBI::dbWriteTable(
    con,
    "sales",
    data.frame(
      region = c("EMEA", "EMEA", "APAC"),
      revenue = c(100, 250, 90)
    )
  )

  measure_file <- tempfile(fileext = ".R")
  writeLines(
    c(
      "#' Revenue by region",
```

```
    "#'",
    "#' @param region `string` Sales region.",
    "#' @measure",
    "revenue_by_region <- function(region, warehouse) {",
    "  DBI::dbGetQuery(",
    "    warehouse,",
    "    'SELECT sum(revenue) AS revenue FROM sales WHERE region = ?',",
    "    params = list(region)",
    "  )",
    "}"
  ),
  measure_file
)
layer <- semantic_layer(measure_file)
unlink(measure_file)

agent <- commons(
  ellmer::chat_anthropic(),
  data_sources = list(warehouse = data_source(con)),
  semantic_layer = layer
)

## Not run:
# Users will need an Anthropic API key to run this code.
agent$chat("How much revenue came from EMEA?")

## End(Not run)

DBI::dbDisconnect(con, shutdown = TRUE)
}
```

`data_source` *Create a data source*

Description

A data source combines a **DBI** connection with a table registry for a `commons()` agent. The agent queries the connection directly; no data is copied.

Usage

```
data_source(..., tables = NULL, dictionary = NULL)
```

Arguments

`...`	A single **DBI** connection.
`tables`	Tables to describe in the agent's system prompt. Supply a character vector of table names, schema-qualified strings such as `"schema.table"`, or DBI::Id objects. The registry does not restrict which tables the connection can query. The

default is every table returned by DBI::dbListTables(). Strings containing dots are treated as schema-qualified names; use DBI::Id(table = "a.b") for a literal table name containing dots.

dictionary — An optional path to a data dictionary describing the source's tables and columns, in the data-dict.yaml format. See the Data dictionaries section.

Value

A commons_data_source object.

Data dictionaries

A data dictionary records what each table's rows represent, what its columns mean, allowed values and units, table relationships, and domain terms. commons uses it in three places:

- The dataset-level description and details, along with the glossary, appear in the system prompt. Use these fields for rules that span tables and guidance about which tables answer a question.
- The first time a conversation touches a table—via the describe_table tool or a SQL query—the tool result includes that table's prose, documented columns, relationships, and relevant glossary definitions. describe_table merges documented columns with the live schema.
- The search_context tool searches the dictionary's prose, with one passage for each table and glossary term.

Trust

Before passing a query to the database, the run_sql tool checks its leading statement keyword against a denylist of common data- and schema-modifying operations. The check is a keyword filter; database permissions remain the access-control boundary. Open the connection in read-only mode where the backend supports it, and grant it access only to the tables the agent needs.

See Also

commons() to build an agent over one or more data sources.

Examples

```
if (requireNamespace("duckdb", quietly = TRUE)) {
  con <- DBI::dbConnect(duckdb::duckdb())
  DBI::dbWriteTable(
    con,
    "sales",
    data.frame(region = c("EMEA", "APAC"), revenue = c(100, 200))
  )

  src <- data_source(con)
  src$tables

  DBI::dbDisconnect(con, shutdown = TRUE)
}
```

`semantic_layer` *Create a semantic layer*

Description

A semantic layer collects governed measures for a `commons()` agent. The agent searches these measures before writing SQL. When a measure matches, the agent runs the calculation defined by your data team.

Usage

```
semantic_layer(...)
```

Arguments

`...`	Paths to R scripts or directories containing functions marked with `@measure`.

Details

A measure is an ordinary R function documented with roxygen comments and marked with `@measure`. It can take model-supplied and injected arguments:

- Arguments documented with `@param` are supplied by the model.
- Undocumented arguments are hidden from the model. `commons()` supplies a matching data source's connection or keeps the argument's default. It errors if neither is available.

Because the measure receives its connection as an argument, the semantic layer can be defined before the connection is opened.

For another dependency, such as an API client, use a default expression that constructs it. Referring to a variable instead would make the measure depend on the environment where the semantic layer was created.

Value

A `commons_semantic_layer` object.

See Also

`commons()` to give a semantic layer to an agent.

Examples

```
measure_file <- tempfile(fileext = ".R")
writeLines(
  c(
    "#' Revenue by region",
    "#'",
    "#' @param region `string` Sales region.",
```

```
    "#' @measure",
    "revenue_by_region <- function(region, warehouse) {",
    "  DBI::dbGetQuery(",
    "    warehouse,",
    "    'SELECT sum(revenue) AS revenue FROM sales WHERE region = ?',",
    "    params = list(region)",
    "  )",
    "}"
  ),
  measure_file
)

layer <- semantic_layer(measure_file)
unlink(measure_file)
```

Index