Package 'ggsql'

May 27, 2026

Title Grammar of Graphics for SQL

Version 0.3.2

Description Allows you to write queries that combine SQL (Structured Query Language) data retrieval with visualization specifications in a single, composable syntax. The 'ggsql' package binds directly with the 'ggsql' 'Rust' library and allows you to set up readers and writers and execute queries against it. The package also offers 'knitr' and 'shiny' integration allowing the user to use 'ggsql' in both frameworks.

License MIT + file LICENSE

Encoding UTF-8

SystemRequirements Cargo (Rust's package manager), rustc, ODBC driver manager (unixODBC on Linux, iODBC or unixODBC on macOS; built into Windows)

RoxygenNote 7.3.3

Imports cli, htmltools, htmlwidgets, jsonlite, knitr, nanoarrow, R6, rlang (>= 1.1.0), yaml

Suggests gapminder, quarto, reticulate, rmarkdown, rsvg, shiny, testthat (>= 3.0.0), V8, withr

Config/testthat/edition 3

Config/rextendr/version 0.4.2.9000

Depends R (>= 4.2)

VignetteBuilder quarto

URL https://r.ggsql.org, https://github.com/posit-dev/ggsql-r

Config/Needs/website tidyverse/tidytemplate

Config/Needs/js node

BugReports https://github.com/posit-dev/ggsql-r/issues

NeedsCompilation yes

Author Thomas Lin Pedersen [aut, cre] (ORCID: <https://orcid.org/0000-0002-5147-4711>), George Stagg [aut] (ORCID: <https://orcid.org/0009-0006-3173-9846>),

Teun van den Brand [aut] (ORCID:
<https://orcid.org/0000-0002-9335-7468>),
Posit, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Thomas Lin Pedersen <thomas.pedersen@posit.co>

Repository CRAN

Date/Publication 2026-05-27 09:10:08 UTC

Contents

`custom_reader` *Create a reader backed by R callbacks*

Description

Construct a reader whose behavior is defined entirely by R functions you supply. This makes it possible to plug in data sources that aren't provided natively by ggsql (e.g. an in-memory store, a custom HTTP API, a DBI connection, etc.) without touching the Rust side.

Usage

```
custom_reader(execute_sql, register = NULL, unregister = NULL)
```

Arguments

`execute_sql`	A function `function(sql)` that executes `sql` and returns either a data frame or a raw vector containing Arrow IPC stream bytes (as produced by `nanoarrow::as_nanoarrow_array_stre` / arrow IPC writers).
`register`	Optional `function(name, df, replace)` that registers `df` as a table named `name`. `replace` is `TRUE` if the caller expects an existing table with the same name to be replaced.
`unregister`	Optional `function(name)` that removes a previously registered table.

Details

Only `execute_sql` is required. If `register` or `unregister` are omitted, calling `ggsql_register()` / `ggsql_unregister()` on the returned reader raises an error.

Value

A `Reader` object, usable anywhere the other `*_reader()` constructors are accepted.

See Also

Other readers: `duckdb_reader()`, `odbc_reader()`, `snowflake_reader()`

Examples

```
# A trivial reader backed by a list of data frames in an environment,
# delegating the actual SQL engine to an in-memory DuckDB.
store <- new.env(parent = emptyenv())
backend <- duckdb_reader()
reader <- custom_reader(
  execute_sql = function(sql) ggsql_execute_sql(backend, sql),
  register = function(name, df, replace) {
    store[[name]] <- df
    ggsql_register(backend, df, name, replace = replace)
  },
  unregister = function(name) {
    rm(list = name, envir = store)
    ggsql_unregister(backend, name)
  }
)
ggsql_register(reader, mtcars, "cars")
ggsql_execute_sql(reader, "SELECT mpg, disp FROM cars LIMIT 3")
```

---

duckdb_reader *Create a DuckDB reader*

Description

Creates a DuckDB database connection that can execute SQL queries and register data frames as queryable tables. The default creates an empty in-memory database but you can also pass the path to a DuckDB database to directly interact with that.

Usage

```
duckdb_reader(database = NULL)
```

Arguments

`database`	Path to a DuckDB database file, or `NULL` (the default) for an in-memory database.

Value

A Reader object.

See Also

Other readers: custom_reader(), odbc_reader(), snowflake_reader()

Examples

```
reader <- duckdb_reader()
ggsql_register(reader, mtcars, "cars")
df <- ggsql_execute_sql(reader, "SELECT mpg, disp FROM cars LIMIT 5")
```

---

ggsqlOutput *Shiny bindings for ggsql*

Description

Render ggsql visualizations in a Shiny application. renderGgsql() accepts either a ggsql query string or a Spec object (returned by ggsql_execute()). When given a string, it validates and executes the query against reader.

Usage

```
ggsqlOutput(outputId, width = "100%", height = "400px")

renderGgsql(expr, reader = NULL, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	Output variable to read from.
width, height	CSS dimensions for the output container.
expr	An expression that returns a ggsql query string or a Spec object. Strings may contain r:varname references that resolve variables from the expression's local scope (see Examples).
reader	A Reader object created by duckdb_reader(). When NULL (the default), the session reader set by ggsql_session_reader() is used.
env	The environment in which to evaluate expr.
quoted	Logical. Is expr a quoted expression?

Value

ggsqlOutput() returns a Shiny UI element. renderGgsql() returns a Shiny render function.

Examples

```
library(shiny)

ui <- fluidPage(
  ggsqlOutput("chart")
)

server <- function(input, output, session) {
  ggsql_session_reader(duckdb_reader())

  output$chart <- renderGgsql({
    "SELECT * FROM r:mtcars VISUALISE mpg AS x, disp AS y DRAW point"
  })
}

shinyApp(ui, server)
```

`ggsql_execute` *Execute a ggsql query*

Description

Parses the query, and execute it against the reader's database. Returns either a visualization specification ready for rendering (`ggsql_execute`) or a data frame with the query result (`ggsql_execute_sql`).

Usage

```
ggsql_execute(reader, query)

ggsql_execute_sql(reader, query)
```

Arguments

`reader`	A `Reader` object created by e.g. `duckdb_reader()` or `odbc_reader()`.
`query`	A ggsql query string (SQL + VISUALISE clause).

Value

`ggsql_execute()` returns `Spec` object. `ggsql_execute_sql()` returns a data frame or `NULL`

Examples

```
reader <- duckdb_reader()
ggsql_register(reader, mtcars, "cars")
spec <- ggsql_execute(reader,
  "SELECT * FROM cars VISUALISE mpg AS x, disp AS y DRAW point"
)
```

ggsql_register *Register and unregisters a data frame as a queryable table*

Description

After registration, the data frame can be queried by name in SQL statements. You can use `ggsql_table` to extract tables from the reader (both registered ones and those native to the backend) and `ggsql_table_names` to get a vector of all the tables in reader.

Usage

```
ggsql_register(reader, df, name, replace = FALSE)

ggsql_unregister(reader, name)

ggsql_table(reader, name)

ggsql_table_names(reader)
```

Arguments

`reader`	A `Reader` object created by e.g. `duckdb_reader()`.
`df`	A data frame to register.
`name`	The name of the table.
`replace`	If `TRUE`, replace an existing table with the same name. Defaults to `FALSE`.

Value

`reader` for `ggsql_register()` and `ggsql_unregister()`. `ggsql_table` returns a data.frame if the table exists and `NULL` if not. `ggsql_table_names` return a character vector.

Examples

```
reader <- duckdb_reader()
ggsql_register(reader, mtcars, "cars")

ggsql_table_names(reader)

ggsql_table(reader, "cars")

ggsql_unregister(reader, "cars")

ggsql_table_names(reader)
```

ggsql_render *Render a spec with a writer*

Description

This function takes a Spec object as returned by ggsql_execute() and renders it with the provided writer.

Usage

```
ggsql_render(writer, spec)
```

Arguments

writer	A Writer object created by e.g. vegalite_writer().
spec	A Spec object returned by ggsql_execute().

Value

Writer dependent:

- vegalite_writer: A string holding the vegalite JSON representation of the visualization

Examples

```
reader <- duckdb_reader()
ggsql_register(reader, mtcars, "cars")
spec <- ggsql_execute(reader,
  "SELECT * FROM cars VISUALISE mpg AS x DRAW histogram"
)

ggsql_render(vegalite_writer(), spec)
```

ggsql_save *Save a ggsql spec to a file*

Description

This function renders a specification and returns it either as a Vegalite json string, an SVG or a PNG. For the latter two, the Vegalite JSON is rendered to SVG using the V8 package and, potentially, converted to PNG using the rsvg package.

Usage

```
ggsql_save(spec, file, width = 600, height = 400)
```

Arguments

spec	A Spec object returned by ggsql_execute().
file	Output file path. Extension determines format: .svg, .png, or .json.
width	Width in pixels.
height	Height in pixels.

Value

file, invisibly.

Examples

```
reader <- duckdb_reader()
ggsql_register(reader, mtcars, "cars")
spec <- ggsql_execute(reader,
  "SELECT * FROM cars VISUALISE mpg AS x, disp AS y DRAW point"
)
spec_file <- tempfile(fileext = ".json")
ggsql_save(spec, spec_file)
```

---

ggsql_session_reader *Set the ggsql reader for the current Shiny session*

Description

Registers a duckdb_reader() for use by all renderGgsql() outputs in the current Shiny session. Must be called from within a Shiny server function (i.e., while a session is active). The reader is automatically cleaned up when the session ends.

Usage

```
ggsql_session_reader(reader, session = shiny::getDefaultReactiveDomain())
```

Arguments

reader	A Reader object created by duckdb_reader().
session	The Shiny session object. Defaults to the current session.

Value

The reader, invisibly.

Examples

```
library(shiny)

ui <- fluidPage(
  ggsqlOutput("chart")
)

server <- function(input, output, session) {
  ggsql_session_reader(duckdb_reader())

  output$chart <- renderGgsql({
    "SELECT * FROM r:mtcars VISUALISE mpg AS x, disp AS y DRAW point"
  })
}

shinyApp(ui, server)
```

`ggsql_validate` *Validate a ggsql query*

Description

Checks query syntax and semantics without executing SQL. Returns a validation result that can be inspected for errors and warnings.

Usage

```
ggsql_validate(query)

ggsql_has_visual(x)

ggsql_is_valid(x)
```

Arguments

`query`	A ggsql query string.
`x`	A `ggsql_validated` object

Value

A `ggsql_validated` object for `ggsql_validate()`. A boolean for `ggsql_has_visual()` and `ggsql_is_valid()`

Examples

```
result <- ggsql_validate("SELECT 1 AS x, 2 AS y VISUALISE x, y DRAW point")
result
```

odbc_reader *Create an ODBC reader*

Description

Creates a connection to a database through an ODBC driver. Can execute SQL queries and, where the backend supports it, register R data frames as temporary tables. Requires an ODBC driver manager (unixODBC/iODBC on Unix, built into Windows) and the appropriate ODBC driver for the target database to be installed.

Usage

```
odbc_reader(
  dsn = NULL,
  driver = NULL,
  server = NULL,
  database = NULL,
  uid = NULL,
  pwd = NULL,
  ...,
  connection_string = NULL
)
```

Arguments

dsn	Name of a data source configured in odbc.ini / odbcinst.ini.
driver	ODBC driver name (e.g. "{PostgreSQL}", "{Snowflake}"). Curly brackets are optional.
server, database, uid, pwd	Common ODBC parameters.
...	Additional named key = value parameters appended to the connection string (e.g. Port = 5432, Warehouse = "COMPUTE_WH").
connection_string	A full ODBC connection string, e.g. "Driver={PostgreSQL};Server=localhost;Database=mydb;UID A leading odbc:// is accepted and stripped.

Details

Either pass a full ODBC connection string via connection_string, or supply named components (dsn, driver, server, database, uid, pwd, plus any extra key = value pairs in ...) and they will be assembled into a connection string. If connection_string is supplied, the other named arguments are ignored.

Value

A Reader object.

Credentials

Connection strings are stored in-memory for the life of the reader. Prefer configuring credentials through a DSN, `~/.odbc.ini`, or environment variables rather than hard-coding passwords in scripts.

See Also

Other readers: `custom_reader()`, `duckdb_reader()`, `snowflake_reader()`

Examples

```
## Not run:
# Using a preconfigured DSN
reader <- odbc_reader(dsn = "mydsn")

# Building a connection string from components
reader <- odbc_reader(
  driver = "{PostgreSQL}",
  server = "localhost",
  database = "mydb",
  uid = "user",
  pwd = "secret",
  Port = 5432
)

# Passing a full connection string
reader <- odbc_reader("Driver={SQLite3};Database=:memory:")

## End(Not run)
```

---

snowflake_reader *Create a Snowflake reader*

Description

Convenience constructor for Snowflake connections. Uses the ODBC reader under the hood with the `Snowflake` driver, and takes advantage of the dedicated Snowflake handling in the ggsql Rust core:

Usage

```
snowflake_reader(
  account = NULL,
  warehouse = NULL,
  database = NULL,
  schema = NULL,
  role = NULL,
```

```
  user = NULL,
  password = NULL,
  authenticator = NULL,
  connection_name = NULL,
  driver = NULL,
  ...,
  connection_string = NULL
)
```

Arguments

account	Snowflake account identifier (e.g. "xy12345" or "xy12345.us-east-1"). Translated to Server={account}.snowflakecomputing.com in the connection string.
warehouse, database, schema, role	Snowflake session defaults.
user, password	User credentials. Prefer a DSN, connection_name, or authenticator = "externalbrowser" over hard-coded passwords.
authenticator	Snowflake authenticator (e.g. "externalbrowser", "snowflake_jwt", "oauth").
connection_name	Named entry in ~/.snowflake/connections.toml whose fields will fill in the remaining connection parameters.
driver	Override the ODBC driver name (defaults to "Snowflake").
...	Additional named key = value parameters appended to the connection string.
connection_string	A full raw connection string, bypassing the named arguments. Driver={Snowflake}; is prepended if it isn't already present.

Details

Special handling of Snowflake includes:

- If connection_name is supplied (or ConnectionName= appears in the connection string), it is resolved against ~/.snowflake/connections.toml.
- When running inside Posit Workbench, an OAuth token is automatically injected if one is available.
- Schema introspection uses Snowflake's SHOW DATABASES / SCHEMAS / TABLES commands rather than information_schema.

Requires the Snowflake ODBC driver to be installed on the system.

Value

A Reader object.

See Also

Other readers: custom_reader(), duckdb_reader(), odbc_reader()

Examples

```
## Not run:
# Using a named connection from ~/.snowflake/connections.toml
reader <- snowflake_reader(connection_name = "my_workbench")

# Browser-based SSO
reader <- snowflake_reader(
  account = "xy12345.us-east-1",
  user = "alice@example.com",
  authenticator = "externalbrowser",
  warehouse = "COMPUTE_WH",
  database = "ANALYTICS",
  schema = "PUBLIC",
  role = "ANALYST"
)

## End(Not run)
```

---

spec_utility *Utility functions for visualization specifications*

Description

These functions allow you to extract various information from a Spec object returned by ggsql_execute().

Usage

```
ggsql_metadata(spec)

ggsql_sql(spec)

ggsql_visual(spec)

ggsql_layer_count(spec)

ggsql_layer_data(spec, index = 1L)

ggsql_stat_data(spec, index = 1L)

ggsql_layer_sql(spec, index = 1L)

ggsql_stat_sql(spec, index = 1L)

ggsql_warnings(spec)
```

Arguments

spec	A Spec object as returned by ggsql_execute()
index	Layer index

Value

- ggsql_metadata: A list with elements rows, columns, and layer_count
- ggsql_sql: A character string with the SQL portion of the query
- ggsql_visual: A character string with the visual portion of the query
- ggsql_layer_count: An integer giving the number of layers
- ggsql_layer_data: A data frame, or NULL if no data is available for this layer
- ggsql_stat_data: A data frame, or NULL if the layer doesn't use a stat transform
- ggsql_layer_sql: A character string with the SQL query used by the layer to fetch its data, or NULL if the layer doesn't have any data.
- ggsql_stat_sql: A character string with the SQL query used by the layers stat transform, or NULL if the layer doesn't have a stat transform.
- ggsql_warnings: A data.frame with columns message, line, and column giving the validation warnings for the spec

Examples

```
reader <- duckdb_reader()
ggsql_register(reader, mtcars, "cars")
spec <- ggsql_execute(reader,
  "SELECT * FROM cars VISUALISE mpg AS x DRAW histogram"
)

ggsql_metadata(spec)

ggsql_visual(spec)
```

---

vegalite_writer *Create a Vega-Lite writer*

Description

This function creates a vegalite writer which is currently the only writer type for ggsql

Usage

```
vegalite_writer()
```

Value

A Writer object.

Examples

```
vegalite_writer()
```

Index