Package 'DBmaps'

August 31, 2026

Title An R Tool for Streamlining Database Joins

Version 0.1.1

Description Simplifies and automates the process of exploring and merging data from relational databases. This package allows users to discover table relationships, create a map of all possible joins, and generate executable plans to merge data based on a structured metadata framework.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0), DiagrammeR

Config/testthat/edition 3

VignetteBuilder knitr

Imports data.table

Depends R (>= 3.5)

LazyData true

NeedsCompilation no

Author Akshat Maurya [aut, cre],
David Shilane [aut]

Maintainer Akshat Maurya <codingmaster902@gmail.com>

Repository CRAN

Date/Publication 2026-08-31 16:40:02 UTC

Contents

`add_table` *Add a Table's Metadata to a Registry*

Description

A generic function to add new table metadata to a registry object.

Usage

```
add_table(registry, table_metadata)
```

Arguments

`registry`	The registry object to which metadata will be added.
`table_metadata`	A data.table object created by `table_info()`.

Value

The updated registry object.

`create_join_plan` *Create a Plan for Aggregating and Merging Tables*

Description

Create a Plan for Aggregating and Merging Tables

Usage

```
create_join_plan(
  base_table,
  selections,
  metadata_dt,
  join_map = NULL,
  tables_dis = NULL
)
```

Arguments

base_table	A character string specifying the main table.
selections	A named list specifying the columns or aggregations to include.
metadata_dt	The master metadata data.table.
join_map	An optional "Join Map" data.table produced by map_join_paths(). If NULL (the default), the map will be generated automatically from the metadata.
tables_dis	An optional named list of data.tables used for data-driven (inferred) join discovery.

Value

A data.table representing the ordered plan steps.

---

create_metadata_registry *Create a Metadata Registry*

Description

Initializes an empty data.table with a custom class "MetadataRegistry" to store and manage metadata definitions.

Usage

```
create_metadata_registry()
```

Value

An empty data.table with the class "MetadataRegistry".

---

customers *Sample Customer Data*

Description

A sample dataset containing demographic information for customers included with the DBmaps package.

Usage

```
customers
```

Format

A data.table with 5 variables:

customer_id A unique identifier for each customer.

age The age of the customer in years.

gender The gender of the customer.

income The income level of the customer.

region The geographical region where the customer resides.

Source

Generated for package examples.

`execute_join_plan` *Execute a Join Plan*

Description

Takes a plan generated by `create_join_plan()` and executes it sequentially to produce a final, merged data.table.

Usage

```
execute_join_plan(join_plan, data_list)
```

Arguments

`join_plan`	A data.table created by `create_join_plan()`.
`data_list`	A named list of the source data.tables.

Value

A final, merged data.table.

`generate_aggregation_code` *Generate data.table Aggregation Code from Metadata*

Description

Reads metadata from a master data.table and generates executable data.table code strings for performing aggregations.

Usage

```
generate_aggregation_code(
  table_name_filter,
  metadata_dt,
  inferred_join_keys = NULL
)
```

Arguments

`table_name_filter`	Character string, the name of the table for which to generate aggregation code.
`metadata_dt`	A data.table containing the master metadata, created by calling `table_info()` for multiple tables and combining them.
`inferred_join_keys`	An optional character vector of column names discovered by the join engine. If provided, overrides the metadata's grouping variables.

Value

A named character vector where each element is a runnable `data.table` code string, and the names correspond to the grouping variables.

Examples

```
# First, create some metadata
customers_info <- table_info(
  table_name = "customers",
  source_identifier = "customers.csv",
  identifier_columns = "customer_id",
  key_outcome_specs = list(
    list(OutcomeName = "CustomerCount", ValueExpression = 1, AggregationMethods = list(
      list(AggregatedName = "CountByRegion", AggregationFunction = "sum",
           GroupingVariables = "region")
    ))
))

transactions_info <- table_info(
  table_name = "transactions",
```

```
  source_identifier = "transactions.csv",
  identifier_columns = "transaction_id",
  key_outcome_specs = list(
   list(OutcomeName = "Revenue", ValueExpression = quote(amount), AggregationMethods = list(
      list(AggregatedName = "RevenueByCustomer", AggregationFunction = "sum",
           GroupingVariables = "customer_id"),
      list(AggregatedName = "RevenueByProduct", AggregationFunction = "sum",
           GroupingVariables = "product_id")
    )),
    list(OutcomeName = "Transactions", ValueExpression = 1, AggregationMethods = list(
      list(AggregatedName = "TransactionsByCustomer", AggregationFunction = "sum",
           GroupingVariables = "customer_id")
    ))
))

master_metadata <- data.table::rbindlist(list(customers_info, transactions_info))

# Now, generate the code for the "transactions" table
generated_code <- generate_aggregation_code("transactions", master_metadata)
print(generated_code)

# To demonstrate execution:
# 1. Create the sample data
transactions <- data.table::data.table(
  transaction_id = c("T001", "T002", "T003"),
  customer_id = c("C001", "C002", "C001"),
  product_id = c("P001", "P002", "P001"),
  amount = c(10.0, 20.0, 15.0)
)

# 2. Parse and evaluate the first generated statement
revenue_by_customer_code <- generated_code["customer_id"]
cat("Executing code:\n", revenue_by_customer_code)
revenue_by_customer_dt <- eval(parse(text = revenue_by_customer_code))
print(revenue_by_customer_dt)
```

`map_join_paths` *Discover Potential Join Paths from Metadata and Data*

Description

Analyzes metadata for explicit joins and optionally scans data to infer additional joins. Handles single- and multi-variable join keys.

Usage

```
map_join_paths(metadata_dt, data_list = NULL)
```

Arguments

metadata_dt	A data.table containing the master metadata.
data_list	A named list of data.tables (names match table_name in metadata_dt). If provided, scans data to find inferred join paths. Defaults to NULL.

Value

A data.table representing the "Join Map" with columns: table_from, table_to, key_from, key_to

---

plot_join_plan *Plot a Join Plan as a Flowchart*

Description

Takes a plan generated by create_join_plan() and creates a flowchart visualizing the sequence of aggregations and merges.

Usage

```
plot_join_plan(join_plan)
```

Arguments

join_plan	A data.table created by create_join_plan().

Value

A DiagrammeR graph object that can be printed to the RStudio Viewer pane.

---

products *Sample Product Data*

Description

A sample dataset containing product information included with the DBmaps package.

Usage

```
products
```

Format

A data.table with 3 variables:

product_id A unique identifier for each product.

category The category to which the product belongs.

original_price The original price of the product.

Source

Generated for package examples.

`table_info` *Define Metadata for a Data Table in a Tidy data.table*

Description

Takes descriptive information about a table and returns a tidy data.table.

Usage

```
table_info(
  table_name,
  source_identifier,
  identifier_columns,
  key_outcome_specs
)
```

Arguments

`table_name`	Character string, the conceptual name of the table.
`source_identifier`	Character string, the file name or DB table identifier.
`identifier_columns`	Character vector, names of column(s) acting as primary key(s).
`key_outcome_specs`	A list of 'OutcomeSpec' lists.

Value

A tidy data.table with the table's metadata. The `identifier_columns` and `grouping_variables` columns are list-columns.

Examples

```
transactions_info <- table_info(
  table_name = "transactions",
  source_identifier = "transactions.csv",
  identifier_columns = c("customer_id", "product_id", "time"),
  key_outcome_specs = list(
    list(
      OutcomeName = "Revenue",
      ValueExpression = quote(price * quantity),
      AggregationMethods = list(
        list(AggregatedName = "RevenueByCustomer", AggregationFunction = "sum",
             GroupingVariables = "customer_id"),
        list(AggregatedName = "RevenueByProduct", AggregationFunction = "sum",
             GroupingVariables = "product_id")
      )
    )
  )
)
# Note the structure of the list-columns
print(transactions_info)
str(transactions_info[, .(identifier_columns, grouping_variable)])
```

`transactions` *Sample Transaction Data*

Description

A sample dataset of transaction events, linking customers and products. This is a typical "fact" table in a relational schema.

Usage

```
transactions
```

Format

A data.table with 5 variables:

customer_id Identifier for the customer making the transaction.

product_id Identifier for the product being purchased.

time The timestamp of the transaction (POSIXct format).

quantity The number of units of the product purchased.

price The price per unit at the time of transaction.

Source

Generated for package examples.

views *Sample Product View Data*

Description

A sample dataset of product view events, linking customers and products. This is a smaller, sampled version of a potentially very large event log.

Usage

```
views
```

Format

A data.table with 3 variables:

customer_id Identifier for the customer viewing the product.

product_id Identifier for the product being viewed.

time The timestamp of the view event (POSIXct format).

Source

Generated for package examples.

Index