Package 'lbugr'

July 11, 2026

Title Interface to 'ladybug' Graph Database

Version 0.1.1

Maintainer Manuel Wick-Eckl <manuel.wick@gmail.com>

Description Provides a high-performance 'R' interface to the 'ladybug' graph database. Uses the 'reticulate' package to wrap the official Python 'ladybug' client. Enables seamless interaction with 'Ladybug' from within 'R' for managing database connections, executing 'Cypher' queries, and loading data from 'R' data frames. Converts query results into popular 'R' data structures including 'tibble', 'igraph', 'tidygraph', and 'g6R' objects for analysis and visualization workflows.

URL https://wickm.github.io/lbugr/, https://github.com/WickM/lbugr

BugReports https://github.com/WickM/lbugr/issues

License MIT + file LICENSE

Imports reticulate, digest, tibble

Suggests g6R, igraph, tidygraph, ggraph, ggplot2, jsonlite, testthat (>= 3.0.0), knitr, rmarkdown, spelling, arrow, withr

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Language en-US

NeedsCompilation no

Author Manuel Wick-Eckl [aut, cre, cph]

Repository CRAN

Date/Publication 2026-07-11 09:40:02 UTC

Contents

`as.data.frame.ladybug.query_result.QueryResult`
Convert a Ladybug Query Result to a Data Frame

Description

Provides an S3 method to seamlessly convert a Ladybug query result object into a standard R `data.frame`.

Usage

```
## S3 method for class 'ladybug.query_result.QueryResult'
as.data.frame(x, ...)
```

Arguments

`x`	A Ladybug query result object.
`...`	Additional arguments passed to `as.data.frame`.

Value

An R `data.frame` containing the query results.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")

# Convert the result to a data.frame
df <- as.data.frame(result)
print(df)

## End(Not run)
```

`as_igraph` *Convert a Ladybug Query Result to an igraph Object*

Description

Converts a Ladybug query result into an `igraph` graph object.

Usage

```
as_igraph(query_result)
```

Arguments

`query_result`	A `ladybug_query_result` object from `lb_execute()` that contains a graph.

Details

This function takes a `ladybug_query_result` object and extracts nodes and edges directly from the query results, then constructs an `igraph` object. It is the final step in the `lb_execute -> as_igraph` workflow.

Value

An `igraph` object.

Examples

```
## Not run:
if (requireNamespace("igraph", quietly = TRUE)) {
  conn <- lb_connection(":memory:")
  lb_execute(conn, "CREATE NODE TABLE Person(name STRING,
  PRIMARY KEY (name))")
  lb_execute(conn, "CREATE REL TABLE Knows(FROM Person TO Person)")
  lb_execute(conn, "CREATE (p:Person {name: 'Alice'}),
  (q:Person {name: 'Bob'})")
```

```
  lb_execute(conn, "MATCH (a:Person), (b:Person) WHERE
                                                    a.name='Alice' AND
                                                    b.name='Bob'
                                                    CREATE (a)-[:Knows]->(b)"
)

  res <- lb_execute(conn, "MATCH (p:Person)-[k:Knows]->(q:Person)
  RETURN p, k, q")
  g <- as_igraph(res)
  print(g)
  rm(conn, res, g)
}

## End(Not run)
```

`as_tibble.ladybug.query_result.QueryResult` *Convert a Ladybug Query Result to a Tibble*

Description

Provides an S3 method to convert a Ladybug query result object into a `tibble`. This requires the `tibble` package to be installed.

Usage

```
## S3 method for class 'ladybug.query_result.QueryResult'
as_tibble(x, ...)
```

Arguments

`x`	A Ladybug query result object.
`...`	Additional arguments passed to `as_tibble`.

Value

A `tibble` containing the query results.

Examples

```
## Not run:
if (requireNamespace("tibble", quietly = TRUE)) {
  conn <- lb_connection(":memory:")
  lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
  PRIMARY KEY (name))")
  lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
  result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")

  # Convert the result to a tibble
```

```
  tbl <- tibble::as_tibble(result)
  print(tbl)
}

## End(Not run)
```

`as_tidygraph` *Convert a Ladybug Query Result to a tidygraph Object*

Description

Converts a Ladybug query result into a `tidygraph` `tbl_graph` object.

Usage

```
as_tidygraph(query_result)
```

Arguments

`query_result`	A `ladybug_query_result` object from `lb_execute()` that contains a graph.

Value

A `tbl_graph` object.

Examples

```
## Not run:
if (requireNamespace("tidygraph", quietly = TRUE)) {
  conn <- lb_connection(":memory:")
  lb_execute(conn, "CREATE NODE TABLE Person(name STRING,
  PRIMARY KEY (name))")
  lb_execute(conn, "CREATE (p:Person {name: 'Alice'})")
  res <- lb_execute(conn, "MATCH (p:Person) RETURN p")
  g_tidy <- as_tidygraph(res)
  print(g_tidy)
  rm(conn, res, g_tidy)
}

## End(Not run)
```

check_ladybug_installation *Check for Ladybug Python Dependencies*

Description

This function checks if the required Python package (`ladybug`) is available in the user's `reticulate` environment. If the package is missing, it provides a clear, actionable message guiding the user on how to install it manually.

Usage

```
check_ladybug_installation(quiet = FALSE)
```

Arguments

`quiet`	If TRUE, suppress the success message. Default is FALSE.

Value

`NULL` invisibly. The function is called for its side effect of checking dependencies and printing messages.

Examples

```
## Not run:
check_ladybug_installation()

## End(Not run)
```

lb_connection *Create a Connection to a Ladybug Database*

Description

Establishes a connection to a Ladybug database. If the database does not exist at the specified path, it will be created. This function combines the database initialization and connection steps into a single call.

Usage

```
lb_connection(path)
```

Arguments

`path`	A string specifying the file path for the database. For an in-memory database, use `":memory:"`.

Value

A Python object representing the connection to the Ladybug database.

Examples

```
## Not run:
# Create an in-memory database and connection
conn <- lb_connection(":memory:")

# Create or connect to an on-disk database
temp_db_dir <- file.path(tempdir(), "ladybug_disk_example_db")
db_path <- file.path(temp_db_dir, "ladybug_db")
dir.create(temp_db_dir, recursive = TRUE, showWarnings = FALSE)

# Establish connection
conn_disk <- lb_connection(db_path)

# Ensure the database is shut down and removed on exit
on.exit({
  db <- attr(conn_disk, "lbugr_db")
  if (!is.null(db)) {
    db$shutdown()
  }
  unlink(temp_db_dir, recursive = TRUE)
})

## End(Not run)
```

---

lb_copy_from_csv *Load Data from a CSV File into a Ladybug Table*

Description

Loads data from a CSV file into a specified table in the Ladybug database.

Usage

```
lb_copy_from_csv(conn, file_path, table_name, optional_csv_parameter = NULL)
```

Arguments

conn	A Ladybug connection object.
file_path	A string specifying the path to the CSV file.
table_name	A string specifying the name of the destination table in Ladybug.
optional_csv_parameter	An optional parameter for CSV-specific configurations (e.g., delimiter, header). Refer to Ladybug documentation for available options.

Value

This function is called for its side effect of loading data and does not return a value.

See Also

Ladybug CSV Import

Examples

```
## Not run:
  conn <- lb_connection(":memory:")
  lb_execute(conn, "CREATE NODE TABLE City(name STRING, population INT64,
  PRIMARY KEY (name))")

  # Create a temporary CSV file
  csv_file <- tempfile(fileext = ".csv")
  write.csv(data.frame(name = c("Berlin", "London"),
  population = c(3645000, 8982000)),
            csv_file, row.names = FALSE)

  # Load data from CSV
  lb_copy_from_csv(conn, csv_file, "City")

  # Verify the data
  result <- lb_execute(conn, "MATCH (c:City) RETURN c.name, c.population")
  print(as.data.frame(result))

  # Clean up the temporary file
  unlink(csv_file)

## End(Not run)
```

---

lb_copy_from_df *Load Data from a Data Frame or Tibble into a Ladybug Table*

Description

Efficiently copies data from an R `data.frame` or `tibble` into a specified table in the Ladybug database.

Usage

```
lb_copy_from_df(conn, df, table_name)
```

Arguments

`conn`	A Ladybug connection object.
`df`	A `data.frame` or `tibble` containing the data to load. Column names in the data frame should match the property names in the Ladybug table.
`table_name`	A string specifying the name of the destination table in Ladybug.

Details

When loading into a relationship table, Ladybug assumes the first two columns in the file are: FROM Node Column: The primary key of the FROM nodes. TO Node Column: The primary key of the TO nodes.

Value

This function is called for its side effect of loading data and does not return a value.

See Also

Ladybug Copy from DataFrame

Examples

```
## Not run:
  conn <- lb_connection(":memory:")
  lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
  PRIMARY KEY (name))")
  lb_execute(conn, "CREATE REL TABLE Knows(FROM User TO User)")

  # Load from a data.frame
  users_df <- data.frame(name = c("Carol", "Dan"), age = c(35, 40))
  lb_copy_from_df(conn, users_df, "User")

  # Load from a tibble (requires pre-existing nodes)
  lb_execute(conn, "CREATE (u:User {name: 'Alice'}), (v:User {name: 'Bob'})")
  knows_df <- data.frame(from_person = c("Alice", "Bob"),
  to_person = c("Bob", "Carol"))
  lb_copy_from_df(conn, knows_df, "Knows")

  result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
  print(as.data.frame(result))

  result_rel <- lb_execute(conn, "MATCH (a:User)-[k:Knows]->(b:User)
  RETURN a.name, b.name")
  print(as.data.frame(result_rel))

## End(Not run)
```

---

lb_copy_from_json *Load Data from a JSON File into a Ladybug Table*

Description

Loads data from a JSON file into a specified table in the Ladybug database. This function also ensures the JSON extension is loaded and available.

Usage

```
lb_copy_from_json(conn, file_path, table_name)
```

Arguments

conn	A Ladybug connection object.
file_path	A string specifying the path to the JSON file.
table_name	A string specifying the name of the destination table in Ladybug.

Value

This function is called for its side effect of loading data and does not return a value.

See Also

Ladybug JSON Import, Ladybug JSON Extension

Examples

```
## Not run:
  conn <- lb_connection(":memory:")
  lb_execute(conn, "CREATE NODE TABLE Product(id INT64, name STRING,
  PRIMARY KEY (id))")

  # Create a temporary JSON file
  json_file <- tempfile(fileext = ".json")
  json_data <- '[{"id": 1, "name": "Laptop"}, {"id": 2, "name": "Mouse"}]'
  writeLines(json_data, json_file)

  # Load data from JSON
  lb_copy_from_json(conn, json_file, "Product")

  # Verify the data
  result <- lb_execute(conn, "MATCH (p:Product) RETURN p.id, p.name")
  print(as.data.frame(result))

  # Clean up the temporary file
  unlink(json_file)

## End(Not run)
```

---

lb_copy_from_parquet *Load Data from a Parquet File into a Ladybug Table*

Description

Loads data from a Parquet file into a specified table in the Ladybug database.

Usage

```
lb_copy_from_parquet(conn, file_path, table_name)
```

Arguments

conn	A Ladybug connection object.
file_path	A string specifying the path to the Parquet file.
table_name	A string specifying the name of the destination table in Ladybug.

Value

This function is called for its side effect of loading data and does not return a value.

See Also

Ladybug Parquet Import

Examples

```
## Not run:
  if (requireNamespace("arrow", quietly = TRUE)) {
    conn <- lb_connection(":memory:")
    lb_execute(conn, "CREATE NODE TABLE Country(name STRING, code STRING,
    PRIMARY KEY (name))")

    # Create a temporary Parquet file
    parquet_file <- tempfile(fileext = ".parquet")
    country_df <- data.frame(name = c("USA", "Canada"), code = c("US", "CA"))
    arrow::write_parquet(country_df, parquet_file)

    # Load data from Parquet
    lb_copy_from_parquet(conn, parquet_file, "Country")

    # Verify the data
    result <- lb_execute(conn, "MATCH (c:Country) RETURN c.name, c.code")
    print(as.data.frame(result))

    # Clean up the temporary file
    unlink(parquet_file)
  }

## End(Not run)
```

`lb_create_table_from_df` *Create a Ladybug Table from a Data Frame*

Description

Infers a schema from an R `data.frame` or `tibble` and creates a corresponding NODE table in the Ladybug database.

Usage

```
lb_create_table_from_df(conn, df, table_name, primary_key)
```

Arguments

`conn`	A Ladybug connection object.
`df`	A `data.frame` or `tibble` from which to infer the schema.
`table_name`	A string specifying the name of the new table in Ladybug.
`primary_key`	An optional string specifying the column to be used as the primary key. If not provided, no primary key will be set.

Value

This function is called for its side effect of creating a table and does not return a value.

Examples

```
## Not run:
  conn <- lb_connection(":memory:")

  my_df <- data.frame(
    name = c("Alice", "Bob"),
    age = c(25L, 30L),
    height = c(1.75, 1.80),
    is_student = c(TRUE, FALSE),
    birth_date = as.Date(c("1999-01-01", "1994-05-15"))
  )

  lb_create_table_from_df(conn, my_df, "Person", primary_key = "name")

  # Now you can load data into the created table
  lb_copy_from_df(conn, my_df, "Person")

  result <- lb_execute(conn, "MATCH (p:Person) RETURN *")
  print(as.data.frame(result))

## End(Not run)
```

`lb_execute` *Execute a Cypher Query*

Description

Submits a Cypher query to the Ladybug database for execution. This function is used for all database operations, including schema definition (DDL), data manipulation (DML), and querying (MATCH).

Usage

```
lb_execute(conn, query)
```

Arguments

`conn`	A Ladybug connection object, as returned by `lb_connection()`.
`query`	A string containing the Cypher query to be executed.

Value

A Python object representing the query result.

Examples

```
## Not run:
conn <- lb_connection(":memory:")

# Create a node table
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")

# Insert data
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")

# Query data
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")

## End(Not run)
```

lb_get_all *Retrieve All Rows from a Query Result*

Description

Fetches all rows from a Ladybug query result and returns them as a list of lists.

Usage

```
lb_get_all(result)
```

Arguments

result	A Ladybug query result object.

Value

A list where each element is a list representing a row of results.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
all_results <- lb_get_all(result)

## End(Not run)
```

lb_get_column_data_types *Get Column Data Types from a Query Result*

Description

Retrieves the data types of the columns in a Ladybug query result.

Usage

```
lb_get_column_data_types(result)
```

Arguments

result	A Ladybug query result object.

Value

A character vector of column data types.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
lb_get_column_data_types(result)

## End(Not run)
```

---

lb_get_column_names *Get Column Names from a Query Result*

Description

Retrieves the names of the columns in a Ladybug query result.

Usage

```
lb_get_column_names(result)
```

Arguments

result	A Ladybug query result object.

Value

A character vector of column names.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
lb_get_column_names(result)

## End(Not run)
```

lb_get_n *Retrieve the First N Rows from a Query Result*

Description

Fetches the first n rows from a Ladybug query result.

Usage

```
lb_get_n(result, n)
```

Arguments

result	A Ladybug query result object.
n	The number of rows to retrieve.

Value

A list of the first n rows.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
lb_execute(conn, "CREATE (:User {name: 'Bob', age: 30})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
first_row <- lb_get_n(result, 1)

## End(Not run)
```

lb_get_next *Retrieve the Next Row from a Query Result*

Description

Fetches the next available row from a Ladybug query result. This function can be called repeatedly to iterate through results one by one.

Usage

```
lb_get_next(result)
```

Arguments

result	A Ladybug query result object.

Value

A list representing the next row, or NULL if no more rows are available.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
lb_execute(conn, "CREATE (:User {name: 'Bob', age: 30})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
row1 <- lb_get_next(result)
row2 <- lb_get_next(result)

## End(Not run)
```

---

lb_get_schema *Get Schema from a Query Result*

Description

Retrieves the schema (column names and data types) of a Ladybug query result.

Usage

```
lb_get_schema(result)
```

Arguments

result	A Ladybug query result object.

Value

A named list where names are column names and values are data types.

Examples

```
## Not run:
conn <- lb_connection(":memory:")
lb_execute(conn, "CREATE NODE TABLE User(name STRING, age INT64,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE (:User {name: 'Alice', age: 25})")
result <- lb_execute(conn, "MATCH (a:User) RETURN a.name, a.age")
lb_get_schema(result)
```

```
## End(Not run)
```

`lb_merge_df` *Merge Data from a Data Frame into Ladybug using a Merge Query*

Description

This function is intended for merging data from an R `data.frame` into Ladybug using a specified merge query.

Usage

```
lb_merge_df(conn, df, merge_query)
```

Arguments

`conn`	A Ladybug connection object.
`df`	A `data.frame` or `tibble` containing the data to merge.
`merge_query`	A string representing the Ladybug query for merging data.

Value

This function is called for its side effect of merging data and does not return a value.

See Also

Ladybug Copy from DataFrame

Examples

```
## Not run:
conn <- lb_connection(":memory:")

lb_execute(conn, "CREATE NODE TABLE Person(name STRING, current_city STRING,
PRIMARY KEY (name))")
lb_execute(conn, "CREATE NODE TABLE Item(name STRING, PRIMARY KEY (name))")
lb_execute(conn, "CREATE REL TABLE PURCHASED(FROM Person TO Item)")

my_data <- data.frame(
  name = c("Alice", "Bob"),
  item = c("Book", "Pen"),
  current_city = c("New York", "London")
)

merge_statement <- "
MERGE (p:Person {name: df.name})
MERGE (i:Item {name: df.item})
MERGE (p)-[:PURCHASED]->(i)
```

```
ON MATCH SET p.current_city = df.current_city
ON CREATE SET p.current_city = df.current_city
"

lb_merge_df(conn, my_data, merge_statement)

result <- lb_execute(conn, "MATCH (p:Person)-[:PURCHASED]->(i:Item)
RETURN p.name, i.name, p.current_city")
print(as.data.frame(result))

## End(Not run)
```

Index