Package ‘rlowdb’

April 16, 2025

Type Package

Title Lightweight JSON-Based Database

Version 0.2.0

Description The goal of 'rlowdb' is to provide a lightweight, file-based JSON database.
Inspired by 'LowDB' in 'JavaScript', it generates an intuitive interface for
storing, retrieving, updating, and querying structured data without requiring a full-fledged database system.
Ideal for prototyping, small-scale applications, and lightweight data management needs.

License MIT + file LICENSE

Encoding UTF-8

URL `https://github.com/feddelegrand7/rlowdb`

BugReports `https://github.com/feddelegrand7/rlowdb/issues`

Imports cli (>= 3.6.4), purrr (>= 1.0.2), R6 (>= 2.5.1), rlang (>= 1.1.3), yyjsonr (>= 0.1.20)

Suggests testthat (>= 3.2.3)

Config/testthat/edition 3

RoxygenNote 7.3.1

NeedsCompilation no

Author Mohamed El Fodil Ihaddaden [aut, cre],
lowdb developers [ctb, cph] (developers of the lowdb package)

Maintainer Mohamed El Fodil Ihaddaden <`ihaddaden.fodeil@gmail.com`>

Repository CRAN

Date/Publication 2025-04-16 07:30:02 UTC

Contents

`rlowdb` *rlowdb: A Simple JSON-Based Database in R*

Description

The 'rlowdb' class provides a lightweight, JSON-based database solution for storing and managing structured data in R. It supports CRUD operations (Create, Read, Update, Delete) and enables querying with custom functions.

Methods

Public methods:

- `rlowdb$new()`
- `rlowdb$commit()`
- `rlowdb$get_data()`
- `rlowdb$get_data_collection()`
- `rlowdb$get_data_key()`
- `rlowdb$insert()`
- `rlowdb$find()`
- `rlowdb$update()`
- `rlowdb$upsert()`
- `rlowdb$delete()`
- `rlowdb$query()`
- `rlowdb$filter()`
- `rlowdb$drop()`
- `rlowdb$drop_all()`
- `rlowdb$clear()`
- `rlowdb$count()`
- `rlowdb$list_collections()`
- `rlowdb$exists_collection()`
- `rlowdb$exists_key()`
- `rlowdb$exists_value()`
- `rlowdb$transaction()`
- `rlowdb$restore()`
- `rlowdb$backup()`
- `rlowdb$search()`
- `rlowdb$bulk_insert()`
- `rlowdb$status()`
- `rlowdb$set_auto_commit()`
- `rlowdb$set_verbose()`
- `rlowdb$rename_collection()`
- `rlowdb$list_keys()`

- `rlowdb$count_values()`
- `rlowdb$insert_default_values()`
- `rlowdb$clone_collection()`
- `rlowdb$sample_records()`
- `rlowdb$set_schema()`
- `rlowdb$get_schema()`
- `rlowdb$clone()`

Method `new()`**:** Initialize the database, loading data from a JSON file. If the file does not exist, an empty database is created.

Usage:

```
rlowdb$new(
  file_path,
  default_values = list(),
  auto_commit = TRUE,
  verbose = FALSE,
  pretty = FALSE
)
```

Arguments:

`file_path` The path to the JSON file that stores the database.

`default_values` A list of named list with the format: list(collection_name = list(key_name_1 = value, key_name_2 = value, ..., key_name_n = value)) containing the default values that will be inserted each time the 'insert' method is called. Note that the default_values will not override the existing records. Default is an empty list ('list()').

`auto_commit` whether to update the DB automatically each time there's an insertion, an update or a deletion. Defaults to TRUE.s Note that you can use the 'commit' method to update the DB manually.

`verbose` If TRUE, will print informative messages to the console. Defaults to FALSE

`pretty` Use pretty = FALSE for compact JSON, which is more efficient for data transmission and storage. TRUE for a human readable format. Defaults to FALSE.

Method `commit()`**:** Update the DB with the operated changes.

Usage:

```
rlowdb$commit()
```

Method `get_data()`**:** Retrieve all stored data.

Usage:

```
rlowdb$get_data()
```

Returns: A list containing all database records.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$get_data()
unlink("database.json")
```

Method `get_data_collection()`**:** Retrieve data from a specific collection.

Usage:

```
rlowdb$get_data_collection(collection)
```

Arguments:

`collection` The name of the collection

Returns: A list containing a specific collection's records.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$get_data_collection("users")
unlink("database.json")
```

Method `get_data_key()`**:** Retrieve the records of a specific key within a collection

Usage:

```
rlowdb$get_data_key(collection, key)
```

Arguments:

`collection` The name of the collection

`key` The key name

Returns: A vector/list containing a specific key's records.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$insert("users", list(id = 2, name = "Omar"))
db$get_data_key("users", "name")
unlink("database.json")
```

Method `insert()`**:** Insert a new record into a specified collection.

Usage:

```
rlowdb$insert(collection, record)
```

Arguments:

`collection` The collection name (a string).

`record` A named list representing the record to insert.

Examples:

```
db <- rlowdb$new("database.json", default_values = list(
  "users" = list("active" = TRUE)
 )
)
db$insert("users", list(id = 1, name = "Alice"))
unlink("database.json")
```

Method `find()`**:** Find records in a collection that match a given key-value pair.

Usage:

```
rlowdb$find(collection, key, value)
```

Arguments:

`collection` The collection name (a string).

`key` The field name to search for.

`value` The value to match.

Returns: A list of matching records. Returns an empty list if no match is found.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$find("users", "id", 1)
unlink("database.json")
```

Method `update()`**:** Update existing records in a collection.

Usage:

```
rlowdb$update(collection, key, value, new_data)
```

Arguments:

`collection` The collection name.

`key` The field name to search for.

`value` The value to match.

`new_data` A named list containing the updated data.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$update("users", "id", 1, list(name = "Alice Updated"))
unlink("database.json")
```

Method `upsert()`**:** If a record exists, update it; otherwise, insert a new record. Note that in order to use the method, the 'collection' has to exist

Usage:

```
rlowdb$upsert(collection, key, value, new_data)
```

Arguments:

`collection` The collection name.

`key` The field name to search for.

`value` The value to match.

`new_data` A named list containing the updated data.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 100, name = "Coconut"))
db$upsert("users", "id", 1, list(name = "Alice Updated"))
unlink("database.json")
```

Method `delete()`**:** Delete records from a collection that match a given key-value pair.

Usage:

```
rlowdb$delete(collection, key, value)
```

Arguments:

`collection` The collection name.

`key` The field name to search for.

`value` The value to match.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$delete("users", "id", 1)
db$get_data()
unlink("database.json")
```

Method `query()`**:** Query a collection using a condition string. This function allows filtering records from a collection using a condition string that is evaluated dynamically. The condition supports multiple logical expressions using standard R operators (e.g., '>', '<', '==', '&', '|').

Usage:

```
rlowdb$query(collection, condition = NULL)
```

Arguments:

`collection` The collection name (a string).

`condition` A string representing a logical condition for filtering records. - Supports comparisons ('>', '<', '>=', '<=', '==', '!='). - Allows logical operators ('&' for AND, '|' for OR). - Example: '"views > 200 & id > 2"'. - If 'NULL' or an empty string ('""'), returns all records.

Returns: A list of records that satisfy the condition. If no records match, returns an empty list.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("posts", list(id = 1, title = "LowDB in R", views = 100))
db$insert("posts", list(id = 2, title = "Data Management", views = 250))
db$insert("posts", list(id = 3, title = "Advanced R", views = 300))

# Query posts with views > 200 AND id > 2
db$query("posts", "views > 200 & id > 2")

# Query posts with views > 100 OR id == 1
db$query("posts", "views > 100 | id == 1")

# Query all posts (no condition)
db$query("posts", "")

unlink("database.json")
```

Method `filter()`**:** Filter Records Using a Custom Function This method applies a user-defined function to filter records in a specified collection. The function should take a record as input and return 'TRUE' for records that should be included in the result and 'FALSE' for records that should be excluded.

Usage:

```
rlowdb$filter(collection, filter_fn)
```

Arguments:

`collection` A character string specifying the name of the collection.

`filter_fn` A function that takes a record (a list) as input and returns 'TRUE' or 'FALSE'.

Returns: A list of records that satisfy the filtering condition.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("users", list(name = "Gamma", age = 36))
# Find users older than 30
db$filter("users", function(record) record$age > 30)
unlink("database.json")
```

Method `drop()`**:** Just like DROP TABLE in SQL, drops a complete collection.

Usage:

```
rlowdb$drop(collection)
```

Arguments:

`collection` The collection name.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$drop("users")
db$get_data()
unlink("database.json")
```

Method `drop_all()`**:** Drop all the collections available in your JSON file DB

Usage:

```
rlowdb$drop_all()
```

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$drop_all()
db$get_data()
unlink("database.json")
```

Method `clear()`**:** Removes all records from a collection without deleting the collection itself

Usage:

```
rlowdb$clear(collection)
```

Arguments:

`collection` the collection name

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$clear("users")
db$get_data()
unlink("database.json")
```

Method `count()`**:** Count the number of records in a collection

Usage:

```
rlowdb$count(collection)
```

Arguments:

`collection` the collection name

Returns: numeric

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("users", list(name = "Gamma", age = 36))
db$count("users")
unlink("database.json")
```

Method `list_collections()`**:** List the available collections

Usage:

```
rlowdb$list_collections()
```

Returns: character

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$list_collections()
unlink("database.json")
```

Method `exists_collection()`**:** Check if a collection exists.

Usage:

```
rlowdb$exists_collection(collection)
```

Arguments:

`collection` The collection name

Returns: TRUE if the collection exists, FALSE otherwise

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$exists_collection("users")
unlink("database.json")
```

Method `exists_key()`**:** Check if a key exists within a specific collection.

Usage:

```
rlowdb$exists_key(collection, key)
```

Arguments:

`collection` The collection name

`key` The key name

Returns: TRUE if the key exists, FALSE otherwise

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$exists_key("users", "name")
unlink("database.json")
```

Method `exists_value()`**:** Check if a value exists within a specific collection/key combination.

Usage:

```
rlowdb$exists_value(collection, key, value)
```

Arguments:

`collection` The collection name

`key` The key name

`value` The value to look for

Returns: TRUE if the value exists, FALSE otherwise

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$exists_value("users", "name", "Delta")
unlink("database.json")
```

Method `transaction()`**:** Perform a Transaction with Rollback on Failure

This method executes a sequence of operations as a transaction. If any operation fails, it rolls back all changes to maintain data integrity.

Usage:

```
rlowdb$transaction(transaction_fn)
```

Arguments:

`transaction_fn` A function that performs operations on ‘self‘. It should not return a value.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$count("users")
db$transaction(function() {
  db$insert("users", list(name = "Zlatan", age = 40))
  db$insert("users", list(name = "Neymar", age = 28))
  # if an error is raised, a rollback will happen and
  # the records won't be inserted
})
db$count("users")
unlink("database.json")
```

Method `restore()`**:** Load a JSON backup and replace the current database.

Usage:

```
rlowdb$restore(backup_path)
```

Arguments:

`backup_path` The path of the backup JSON file. Allow users to quickly backup their database.

Method `backup()`**:**

Usage:

```
rlowdb$backup(backup_path)
```

Arguments:

`backup_path` The path of the backup JSON file

Method `search()`**:** Search Records in a Collection

This method searches for records in a collection where a specified key's value contains a given search term.

Usage:

```
rlowdb$search(collection, key, term, ignore.case = FALSE)
```

Arguments:

`collection` A character string specifying the name of the collection.

`key` A character string specifying the field to search within.

`term` A character string specifying the term to search for.

`ignore.case` A logical value indicating whether the search should be case-insensitive (default: ‘FALSE‘).

Returns: A list of matching records. Returns an empty list if no matches are found.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$insert("users", list(id = 2, name = "Bob"))
db$insert("users", list(id = 3, name = "alice"))

# Case-sensitive search
db$search("users", "name", "Alice", ignore.case = FALSE)
```

```
# Case-insensitive search
db$search("users", "name", "alice", ignore.case = TRUE)
unlink("database.json")
```

Method `bulk_insert()`**:** Insert Multiple Records into a Collection

This method inserts multiple records into a specified collection at once. Each record should be a named list representing an entry in the collection.

Usage:

```
rlowdb$bulk_insert(collection, records)
```

Arguments:

`collection` A character string specifying the name of the collection.

`records` A list of named lists, where each named list represents a record to insert.

Examples:

```
db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = "Charlie", age = 40)
))
db$count("users")
unlink("database.json")
```

Method `status()`**:** Provides some useful information about the database

Usage:

```
rlowdb$status()
```

Method `set_auto_commit()`**:** Set the auto_commit value

Usage:

```
rlowdb$set_auto_commit(auto_commit)
```

Arguments:

`auto_commit` TRUE will update automatically the JSON by each insertion/update/delete. If FALSE, you'll need to use the commit method whenever you want to commit your changes to the JSON DB.

Method `set_verbose()`**:** Set the verbose value.

Usage:

```
rlowdb$set_verbose(verbose)
```

Arguments:

`verbose` If TRUE, informative messages will be printed to the console.

Method `rename_collection()`**:** Allows you to rename an existing collection in the database. It checks if the specified collection exists before attempting to rename it.

Usage:

```
rlowdb$rename_collection(collection_name, new_collection_name)
```

Arguments:

`collection_name` A character string representing the current name of the collection to be renamed.

`new_collection_name` A character string representing the new name for the collection.

Examples:

```
db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = "Charlie", age = 40)
))
db$list_collections()
db$rename_collection("users", "customers")
db$list_collections()
unlink("database.json")
```

Method `list_keys()`**:** Retrieves the names (keys) of all keys within a given collection.

Usage:

```
rlowdb$list_keys(collection)
```

Arguments:

`collection` The collection name

Returns: A character vector containing the unique keys (field names) present across all records in the collection.

Examples:

```
db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = "Charlie")
))
db$list_keys("users")
unlink("database.json")
```

Method `count_values()`**:** Count Occurrences of a Key's Values in a Collection

Usage:

```
rlowdb$count_values(collection, key)
```

Arguments:

`collection` The collection name

`key` The key name

Returns: A table (a frequency count) of the values associated with the specified key, showing the number of occurrences of each unique value. If the key does not exist, an error is thrown.

Examples:

```
db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = NA),
  list(id = 4, name = NA)
))
db$count_values("users", "name")
unlink("database.json")
```

Method `insert_default_values()`**:** Add Default Values to Records in a Collection

Ensures that all records in a collection have specific default values for certain keys. If a key is missing, the default value is added. Optionally, existing values can be replaced with defaults.

Usage:

```
rlowdb$insert_default_values(collection, defaults, replace_existing = FALSE)
```

Arguments:

`collection` The collection name.

`defaults` A named list of default values to add.

`replace_existing` Logical; if ‘TRUE‘, replaces existing values with defaults. If ‘FALSE‘, only adds missing keys. Defaults to ‘FALSE‘.

Returns: Updates the collection in place, ensuring consistency of data.

Examples:

```
db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(name = "Alice", age = 30),
  list(name = "Bob"),
  list(name = "Charlie", age = 25, role = "admin")
))

# Add defaults without replacing existing values
db$insert_default_values("users", list(role = "guest", active = TRUE))

# Add defaults and replace existing values
db$insert_default_values("users", list(role = "guest", active = TRUE), replace_existing = TRUE)

unlink("database.json")
```

Method `clone_collection()`**:** Clone an existing collection to a new collection with a different name. This creates an exact copy of the original collection's records under the new name.

Usage:

```
rlowdb$clone_collection(from, to)
```

Arguments:

`from` The name of the source collection to clone.

`to` The name of the new collection.

`overwrite` If FALSE (default), will abort if target collection exists. If TRUE, will overwrite existing target collection.

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$clone_collection("users", "users_backup")
db$list_collections()
unlink("database.json")
```

Method `sample_records()`: Randomly sample records from a collection

Usage:

```
rlowdb$sample_records(collection, n = 1, replace = FALSE, seed = NULL)
```

Arguments:

`collection` The name of the collection to sample from

`n` Number of records to sample. If n > collection size, returns all records with a warning.

`replace` Should sampling be with replacement? Default FALSE

`seed` Optional random seed for reproducible sampling

Returns: A list of sampled records

Examples:

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$insert("users", list(id = 2, name = "Bob"))
db$insert("users", list(id = 3, name = "Charlie"))

# Sample 2 records without replacement
db$sample_records("users", n = 2)

# Sample with replacement
db$sample_records("users", n = 5, replace = TRUE)

# Reproducible sampling with seed
db$sample_records("users", n = 2, seed = 123)
unlink("database.json")
```

Method `set_schema()`: Set schema for a collection to validate future inserts/updates

Usage:

```
rlowdb$set_schema(collection, schema)
```

Arguments:

`collection` The collection name

`schema` A named list where: - Names are field names - Values can be: * A type string ("character", "numeric", etc.) * A function that returns TRUE/FALSE * A vector of allowed values * NULL to make the field optional

Examples:

```
db <- rlowdb$new("database.json")

# Define schema before inserting
db$set_schema("users", list(
  id = "numeric",
  name = function(x) is.character(x) && nchar(x) > 0,
  age = function(x) is.numeric(x) && x >= 0,
  email = NULL  # Optional
))

# This will fail validation:
try(db$insert("users", list(id = "1", name = "")))
```

Method `get_schema()`**:** Retrieve the schema for a specific collection

Usage:

```
rlowdb$get_schema(collection)
```

Arguments:

`collection` The collection name

Returns: list

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
rlowdb$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `rlowdb$get_data`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$get_data()
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$get_data_collection`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$get_data_collection("users")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$get_data_key`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$insert("users", list(id = 2, name = "Omar"))
db$get_data_key("users", "name")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$insert`
## ------------------------------------------------

db <- rlowdb$new("database.json", default_values = list(
  "users" = list("active" = TRUE)
 )
)
db$insert("users", list(id = 1, name = "Alice"))
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$find`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$find("users", "id", 1)
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$update`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$update("users", "id", 1, list(name = "Alice Updated"))
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$upsert`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 100, name = "Coconut"))
db$upsert("users", "id", 1, list(name = "Alice Updated"))
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$delete`
## ------------------------------------------------
```

```
db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$delete("users", "id", 1)
db$get_data()
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$query`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("posts", list(id = 1, title = "LowDB in R", views = 100))
db$insert("posts", list(id = 2, title = "Data Management", views = 250))
db$insert("posts", list(id = 3, title = "Advanced R", views = 300))

# Query posts with views > 200 AND id > 2
db$query("posts", "views > 200 & id > 2")

# Query posts with views > 100 OR id == 1
db$query("posts", "views > 100 | id == 1")

# Query all posts (no condition)
db$query("posts", "")

unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$filter`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("users", list(name = "Gamma", age = 36))
# Find users older than 30
db$filter("users", function(record) record$age > 30)
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$drop`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$drop("users")
db$get_data()
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$drop_all`
## ------------------------------------------------
```

```
db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$drop_all()
db$get_data()
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$clear`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$clear("users")
db$get_data()
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$count`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("users", list(name = "Gamma", age = 36))
db$count("users")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$list_collections`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$list_collections()
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$exists_collection`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$exists_collection("users")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$exists_key`
## ------------------------------------------------

db <- rlowdb$new("database.json")
```

```
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$exists_key("users", "name")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$exists_value`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$insert("consumers", list(name = "Teta", age = 22))
db$exists_value("users", "name", "Delta")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$transaction`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(name = "Delta", age = 25))
db$count("users")
db$transaction(function() {
  db$insert("users", list(name = "Zlatan", age = 40))
  db$insert("users", list(name = "Neymar", age = 28))
  # if an error is raised, a rollback will happen and
  # the records won't be inserted
})
db$count("users")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$search`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$insert("users", list(id = 2, name = "Bob"))
db$insert("users", list(id = 3, name = "alice"))

# Case-sensitive search
db$search("users", "name", "Alice", ignore.case = FALSE)

# Case-insensitive search
db$search("users", "name", "alice", ignore.case = TRUE)
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$bulk_insert`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = "Charlie", age = 40)
))
db$count("users")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$rename_collection`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = "Charlie", age = 40)
))
db$list_collections()
db$rename_collection("users", "customers")
db$list_collections()
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$list_keys`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = "Charlie")
))
db$list_keys("users")
unlink("database.json")


## ------------------------------------------------
## Method `rlowdb$count_values`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(id = 1, name = "Alice", age = 25),
  list(id = 2, name = "Bob", age = 32),
  list(id = 3, name = NA),
  list(id = 4, name = NA)
))
db$count_values("users", "name")
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$insert_default_values`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$bulk_insert("users", list(
  list(name = "Alice", age = 30),
  list(name = "Bob"),
  list(name = "Charlie", age = 25, role = "admin")
))

# Add defaults without replacing existing values
db$insert_default_values("users", list(role = "guest", active = TRUE))

# Add defaults and replace existing values
db$insert_default_values("users", list(role = "guest", active = TRUE), replace_existing = TRUE)

unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$clone_collection`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$clone_collection("users", "users_backup")
db$list_collections()
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$sample_records`
## ------------------------------------------------

db <- rlowdb$new("database.json")
db$insert("users", list(id = 1, name = "Alice"))
db$insert("users", list(id = 2, name = "Bob"))
db$insert("users", list(id = 3, name = "Charlie"))

# Sample 2 records without replacement
db$sample_records("users", n = 2)

# Sample with replacement
db$sample_records("users", n = 5, replace = TRUE)

# Reproducible sampling with seed
db$sample_records("users", n = 2, seed = 123)
unlink("database.json")

## ------------------------------------------------
## Method `rlowdb$set_schema`
## ------------------------------------------------

db <- rlowdb$new("database.json")

# Define schema before inserting
```

```
db$set_schema("users", list(
  id = "numeric",
  name = function(x) is.character(x) && nchar(x) > 0,
  age = function(x) is.numeric(x) && x >= 0,
  email = NULL  # Optional
))

# This will fail validation:
try(db$insert("users", list(id = "1", name = "")))
```

Index