Package: writer (via r-universe)

February 20, 2025

Type Package

Title Write from Multiple Sources to a Database Table

Version 0.2.1

Description Provides unified syntax to write data from lazy 'dplyr' 'tbl' or 'dplyr' 'sql' query or a dataframe to a database table with modes such as create, append, insert, update, upsert, patch, delete, overwrite, overwrite_schema.

License LGPL (>= 3)

Imports DBI (>= 1.2.3), dplyr (>= 1.1.0), dbplyr (>= 2.3.1), glue (>= 1.5.1), cli (>= 3.4.0), rlang (>= 1.1.5),

Suggests RSQLite, testthat (>= 3.0.0),

URL `https://github.com/talegari/writer`

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Repository https://talegari.r-universe.dev

RemoteUrl https://github.com/talegari/writer

RemoteRef HEAD

RemoteSha c726434590ceb2b631744082bffb5167a27ee27e

Contents

write_table *Write from multiple sources to a database table*

Description

Provides unified syntax to write data from `dplyr::tbl()` (lazy dplyr query via a DBI connection) or a `dplyr::sql()` or a data.frame to a database table with modes such as: `create`, `append`, `insert`, `update`, `upsert`, `patch`, `delete`, `overwrite`, `overwrite_schema`.

Usage

```
write_table(
  x,
  table_name,
  mode,
  con = NULL,
  use_transaction = TRUE,
  verbose = TRUE,
  ...
)

## S3 method for class 'tbl_lazy'
write_table(
  x,
  table_name,
  mode = c("create", "append", "insert", "update", "upsert", "patch", "delete",
    "overwrite", "overwrite_schema"),
  con = NULL,
  use_transaction = TRUE,
  verbose = TRUE,
  ...
)

## S3 method for class 'sql'
write_table(
  x,
  table_name,
  mode = c("create", "append", "insert", "update", "upsert", "patch", "delete",
    "overwrite", "overwrite_schema"),
  con = NULL,
  use_transaction = TRUE,
  verbose = TRUE,
  ...
)

## S3 method for class 'data.frame'
write_table(
  x,
  table_name,
  mode = c("create", "append", "insert", "update", "upsert", "patch", "delete",
    "overwrite", "overwrite_schema"),
  con = NULL,
  use_transaction = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	(`dplyr::tbl()` or `dplyr::sql()` or data.frame) The data to write to the database. Input need not have any rows.
`table_name`	(`string` or `AsIs` or `Id`) The name of the table to write to. This has to be one among: string (typically `"catalog.schema.table"`), Object of class "AsIs" generated by wrapping `I()` (typically, `I("catalog.schema.table")`) or Object of class `Id` generated by `DBI::Id` (typically, `DBI::Id("catalog", "schema", "table")`).
`mode`	(`string`) Writing mode. Possible values are one among: `create`, `append`, `insert`, `update`, `upsert`, `patch`, `delete`, `overwrite`, `overwrite_schema`.
`con`	(default: NULL) DBI-connection object to use to write operation. When con is `NULL` and source is a `tbl_lazy`, then DBI connection object from the source tbl is used. When source is a `data.frame`, then `con` should be provided.
`use_transaction`	(flag, default: `TRUE`) Whether the operation should be run within a transaction.
`verbose`	(default: `TRUE`) Whether the progress message should be shown
`...`	Arguments passed to specific function based on `mode`. See details.

Details

The `DBI-dplyr-dbplyr` combination provides a great workflow to handle database operations from R. When saving the output from analysis notebooks or scripts, different functions need to be called based on the type of the object we intend to write. `writer` package solves the problem by exporting one generic `write_table` to handle multiple input types and multiple modes. Further, `overwrite` and `overwrite_schema` refine the idea of table overwrite so that schema of the table is not changed inadvertently.

Modes:

- `create`: Creates a new table only if table with same name does not exist and writes data.
- `append`: Appends data only if table exists and schema matches. See `dplyr::rows_append()`.
- `insert`: Inserts data only if table exists and key values do not exist. See `dplyr::rows_insert()`.
- `update`: Updates data only if table exists and key values match. See `dplyr::rows_update()`
.
- `upsert`: Inserts or Updates only if table exists and depending on whether or not the key value already exists. See `dplyr::rows_upsert()`.
- `patch`: Updates only missing values if table exists and key values match. See `dplyr::rows_patch()`.

- delete: Deletes rows for matching key values if table exists. See dplyr::rows_delete()
- overwrite: Overwrites data only if table exists and schema matches.
- overwrite_schema: Creates a new table with data irrespective of whether table exists or not.

Failures:

The failures are mostly due to unavailable or wrong sql translation to the specific backend. Please raise issues in dbplyr repo.

Errors are raised with a class (using rlang::abort()). These are the error classes:

```
 * `error_input`: Related to wrong or unexpected input.
 * `error_connection`: Raised when connection is inactive.
 * `error_table`: Related to table existence or non-existence.
 * `error_operation`: Related to core write operation.
```

Permissions:

```
 * `create`: create new table
 * `append`, `insert`, `update`, `upsert`, `patch`, `delete`: modify existing table
 * `overwrite`: modify existing table
 * `overwrite_schema`: delete, create and rename table
```

Using transaction:

The use_transaction is always switched on. User should opt-out to achieve an operation if the specific database connection does not permit transactions.

Copying data with dataframe input:

When input is a dataframe, dplyr::copy_to() is used. Supply use_inline = TRUE to use dbplyr::copy_inline() instead.

Value

When successful, returns the output table name as a string. Else, throws an error with informative messages.

Examples

```
## Not run:
#' Create an in-memory SQLite database connection
con = DBI::dbConnect(RSQLite::SQLite(), ":memory:")

remove_new_table = function(){
  df = data.frame(id = 1:3, name = c("Alice", "Bob", "Charlie"))
  DBI::dbRemoveTable(con, "new_table", fail_if_missing = FALSE)
}

create_new_table = function(){
  df = data.frame(id = 1:3, name = c("Alice", "Bob", "Charlie"))
  DBI::dbWriteTable(con, "new_table", df, overwrite = TRUE)
}

#' Create a sample data.frame
```

```
df = data.frame(id = 1:3, name = c("Alice", "Bob", "Charlie"))
df

#' Create a new table
write_table(df, "new_table", mode = "create", con = con)
dplyr::tbl(con, "new_table")


intermediate = dplyr::tbl(con, "new_table") |>
  dplyr::filter(id >= 2)

write_table(intermediate, "new_filtered_table", mode = "create")
dplyr::tbl(con, "new_filtered_table")

#' Append data to an existing table
create_new_table()
append_df = data.frame(id = 4:5, name = c("Dave", "Eve"))

append_df |>
  write_table("new_table", mode = "append", con = con)
dplyr::tbl(con, "new_table")

create_new_table()
write_table(append_df, "append_table", mode = "create", con)
dplyr::tbl(con, "append_table")

dplyr::tbl(con, "append_table") |>
  write_table("new_table", mode = "append")
dplyr::tbl(con, "new_table")

#' Insert data into an existing table, only if key values do not exist
create_new_table()
dplyr::tbl(con, "new_table")
insert_df = data.frame(id = 3:4, name = c("Dave", "Eve"))
insert_df

insert_df |>
  write_table("new_table",
              mode = "insert",
              con = con,
              by = "id",
              conflict = "ignore"
              )
dplyr::tbl(con, "new_table")

create_new_table()
insert_df |>
  write_table("insert_table", mode = "create", con = con)
dplyr::tbl(con, "insert_table")

dplyr::tbl(con, "insert_table") |>
  write_table("new_table",
              mode = "insert",
              by = "id",
              conflict = "ignore"
              )
dplyr::tbl(con, "new_table")

#' Update data in an existing table, only if key values match
create_new_table()
update_df = data.frame(id = c(1, 3), name = c("Alicia", "Charles"))
update_df
write_table(update_df,
            "new_table",
            mode = "update",
            con = con,
            unmatched = "ignore"
            )
dplyr::tbl(con, "new_table")

create_new_table()
write_table(update_df, "update_table", mode = "create", con = con)
dplyr::tbl(con, "update_table")
write_table(dplyr::tbl(con, "update_table"),
            "new_table",
            mode = "update",
            unmatched = "ignore"
            )
dplyr::tbl(con, "new_table")

#' upsert
create_new_table()
upsert_df = data.frame(id = c(2, 6), name = c("Bobby", "Frank"))
upsert_df
write_table(upsert_df,
            "new_table",
            mode = "upsert",
            con = con,
            by = "id"
            )
dplyr::tbl(con, "new_table")

create_new_table()
write_table(upsert_df,
            "upsert_table",
            mode = "create",
            con = con
            )
dplyr::tbl(con, "upsert_table")
write_table(dplyr::tbl(con, "upsert_table"),
            "new_table",
            mode = "upsert"
            )
dplyr::tbl(con, "new_table")
```

```
#' Patch data, updating only missing values
create_new_table()
patch_df = data.frame(id = c(1, 2), name = c("alice", NA))
patch_df
write_table(df_patch, "table_with_na", mode = "create", con)
dplyr::tbl(con, "table_with_na")
df
write_table(df,
            "table_with_na",
            mode = "patch",
            con = con,
            unmatched = "ignore"
            )
dplyr::tbl(con, "table_with_na")

DBI::dbRemoveTable(con, "table_with_na")
write_table(df_patch, "table_with_na", mode = "create", con)
dplyr::tbl(con, "new_table")
write_table(dplyr::tbl(con, "new_table"),
            "table_with_na",
            mode = "patch",
            con = con,
            unmatched = "ignore"
            )
dplyr::tbl(con, "table_with_na")

#' Delete rows for matching key values
create_new_table()
delete_df = data.frame(id = c(3, 4))
delete_df
write_table(df_delete,
            "new_table",
            mode = "delete",
            con = con,
            unmatched = "ignore"
            )
dplyr::tbl(con, "new_table")

create_new_table()
write_table(delete_df,
            "delete_table",
            mode = "create",
            con = con
            )
dplyr::tbl(con, "delete_table")
write_table(df_delete,
            "new_table",
            mode = "delete",
            con = con,
            unmatched = "ignore"
            )
dplyr::tbl(con, "new_table")
```

```
#' Overwrite data in an existing table, schema must match
create_new_table()
overwrite_df = data.frame(id = c(2, 6), name = c("Bobby", "Frank"))
overwrite_df
write_table(overwrite_df,
            "new_table",
            mode = "overwrite",
            con = con
            )
dplyr::tbl(con, "new_table")

create_new_table()
overwrite_df = data.frame(id = c(2, 6), name = c("Bobby", "Frank"))
overwrite_df
write_table(overwrite_df,
            "new_table",
            mode = "overwrite",
            con = con
            )
dplyr::tbl(con, "new_table")

#' Overwrite schema
overwrite_schema_df = data.frame(id = c(2, 6),
                                 name = c("Bobby", "Frank"),
                                 age = c(30, 40)
                                 )
write_table(overwrite_schema_df,
            "new_table",
            mode = "overwrite_schema",
            con = con
            )
dplyr::tbl(con, "new_table")

create_new_table()
write_table(overwrite_schema_df,
            "overwrite_schema_table",
            mode = "overwrite_schema",
            con = con
            )
write_table(dplyr::tbl(con, "overwrite_schema_table"),
            "new_table",
            mode = "overwrite_schema",
            con = con
            )
dplyr::tbl(con, "new_table")

#' Disconnect from the database
DBI::dbDisconnect(con)

## End(Not run)
```

Index