# Package: redquack (via r-universe)

March 26, 2025

**Title** Transfer 'REDCap' Data to 'DuckDB'

**Version** 0.1.1.9000

**Description** Provides a single function to transfer 'REDCap' (Research Electronic Data Capture) data to 'DuckDB'. Processes data in chunks to handle large datasets while minimizing memory usage. Features include resuming incomplete transfers, converting column types, tracking progress, and logging operations in the database.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Imports** audio, beepr, cli, DBI, dplyr, duckdb, httr2, readr, utils

**Suggests** arrow, testthat (>= 3.0.0)

**Config/testthat/edition** 3

**Depends** R (>= 4.1.0)

**Config/pak/sysreqs** libssl-dev libx11-dev xz-utils

**Repository** https://dylanpieper.r-universe.dev

**RemoteUrl** https://github.com/dylanpieper/redquack

**RemoteRef** HEAD

**RemoteSha** 725294ca64902a6daaa249f24b9091345000186d

## Contents

`redcap_to_duckdb` *Transfer 'REDCap' Data to 'DuckDB'*

## Description

Transfer REDCap data to DuckDB in chunks to minimize memory usage.

## Usage

```
redcap_to_duckdb(
  redcap_uri,
  token,
  raw_or_label = "raw",
  raw_or_label_headers = "raw",
  export_checkbox_label = FALSE,
  export_survey_fields = FALSE,
  export_data_access_groups = FALSE,
  blank_for_gray_form_status = FALSE,
  filter_logic = "",
  datetime_range_begin = as.POSIXct(NA),
  datetime_range_end = as.POSIXct(NA),
  fields = NULL,
  forms = NULL,
  events = NULL,
  record_id_name = "record_id",
  chunk_size = 1000,
  chunk_delay = 0.5,
  max_retries = 3,
  output_file = "redcap.duckdb",
  optimize_types = TRUE,
  return_duckdb = TRUE,
  verbose = TRUE,
  beep = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `redcap_uri` | Character string specifying the URI (uniform resource identifier) of the REDCap server's API. |
| `token` | Character string containing the REDCap API token specific to your project. This token is used for authentication and must have export permissions. |
| `raw_or_label` | A string (either `'raw'` or `'label'`) that specifies whether to export the raw coded values or the labels for the options of multiple choice fields. Default is `'raw'`. |

| | |
|---|---|
| `raw_or_label_headers` | A string (either `'raw'` or `'label'`) that specifies for the CSV headers whether to export the variable/field names (raw) or the field labels (label). Default is `'raw'`. |
| `export_checkbox_label` | Logical that specifies the format of checkbox field values specifically when exporting the data as labels. If `raw_or_label` is `'label'` and `export_checkbox_label` is TRUE, the values will be the text displayed to the users. Otherwise, the values will be 0/1. Default is FALSE. |
| `export_survey_fields` | Logical that specifies whether to export the survey identifier field (e.g., 'redcap_survey_identifier') or survey timestamp fields. Default is FALSE. |
| `export_data_access_groups` | Logical that specifies whether or not to export the `redcap_data_access_group` field when data access groups are utilized in the project. Default is FALSE. |
| `blank_for_gray_form_status` | Logical that specifies whether or not to export blank values for instrument complete status fields that have a gray status icon. Default is FALSE. |
| `filter_logic` | String of logic text (e.g., `[gender] = 'male'`) for filtering the data to be returned, where the API will only return records where the logic evaluates as TRUE. Default is an empty string. |
| `datetime_range_begin` | To return only records that have been created or modified *after* a given datetime, provide a POSIXct value. Default is NA (no begin time). |
| `datetime_range_end` | To return only records that have been created or modified *before* a given datetime, provide a POSIXct value. Default is NA (no end time). |
| `fields` | Character vector specifying which fields to export. Default is NULL (all fields). |
| `forms` | Character vector specifying which forms to export. Default is NULL (all forms). |
| `events` | Character vector specifying which events to export. Default is NULL (all events). |
| `record_id_name` | Character string specifying the field name that contains record identifiers used for chunking requests. Default is "record_id". |
| `chunk_size` | Integer specifying the number of record IDs to process per chunk. Default is 1000. Consider decreasing this for projects with many fields. |
| `chunk_delay` | Numeric value specifying the delay in seconds between chunked requests. Default is 0.5 seconds. Adjust to respect REDCap server limits. |
| `max_retries` | Integer specifying the maximum number of retry attempts for failed API requests. Default is 3. Set to 0 to disable retries. |
| `output_file` | Character string specifying the file path where the DuckDB database will be created or modified. Default is "redcap.duckdb" in current working directory. |
| `optimize_types` | Logical indicating whether column types should be optimized after all data is inserted. Default is TRUE, which analyzes column content and converts VARCHAR to more appropriate types (INTEGER, DOUBLE, DATE, TIMESTAMP). If FALSE, all columns will remain as VARCHAR regardless of content. |

| | |
|---|---|
| `return_duckdb` | Logical indicating whether to return a DBI connection object. Default is TRUE. If FALSE, return NULL invisibly. |
| `verbose` | Logical indicating whether to show progress and completion messages. Default is TRUE. |
| `beep` | Logical indicating whether to play sound notifications when the process completes or encounters errors. Default is TRUE. |
| `...` | Additional arguments passed to the REDCap API call. |

## Details

This function transfers data from REDCap to DuckDB in chunks, which helps manage memory usage when dealing with large projects. It creates two tables in the DuckDB database:

- `data`: Contains all transferred REDCap records
- `log`: Contains timestamped logs of the transfer process

The function automatically detects existing databases and handles them in three ways:

- If no database exists, starts a new transfer process
- If a database exists but is incomplete, resumes from the last processed record ID
- If a database exists and is complete, returns a connection without reprocessing

The function fetches record IDs first, then processes records in chunks. If any error occurs during processing, the function will stop further processing to prevent incomplete data. Memory is explicitly managed to handle large datasets.

All data is initially stored as VARCHAR type for consistent handling across chunks. When `optimize_types = TRUE` (the default), column types are automatically converted after all data is inserted, based on content analysis:

- Columns containing only integers are converted to INTEGER
- Columns containing numeric values are converted to DOUBLE
- Columns with valid date strings are converted to DATE
- Columns with valid timestamp strings are converted to TIMESTAMP
- All other columns remain as VARCHAR

When `optimize_types = FALSE`, all columns remain as VARCHAR type. This can be useful when:

- You need consistent string-based handling of all data
- You're working with complex mixed-type data
- You plan to handle type conversions manually in subsequent SQL queries
- Import speed is prioritized over storage efficiency or query optimization

## Value

If `return_duckdb` is TRUE, returns a DBI connection object to the DuckDB database, whether newly created, partially completed and resumed, or already complete. Connection has attributes:

- `had_errors`: Logical indicating if errors occurred during the transfer
- `error_chunks`: Vector of chunk numbers that failed processing (if any)

If `return_duckdb` is FALSE, returns invisibly.

## Database Connection

The function returns an open connection to the DuckDB database when `return_duckdb = TRUE`. You must explicitly close this connection with `DBI::dbDisconnect()` when finished.

## See Also

`dbConnect` for database connection details `duckdb` for DuckDB database information `req_retry` for retry functionality details

## Examples

```
## Not run:
# Basic usage with API token
con <- redcap_to_duckdb(
  redcap_uri = "https://redcap.example.org/api/",
  token = "YOUR_API_TOKEN",
  record_id_name = "record_id",
  chunk_size = 1000
  # Increase chunk size for memory-efficient systems (faster)
  # Decrease chunk size for memory-constrained systems (slower)
)

# Query the resulting database
data <- DBI::dbGetQuery(con, "SELECT * FROM data LIMIT 10")

# View transfer logs
logs <- DBI::dbGetQuery(con, "SELECT * FROM log")

# Remember to close the connection
DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

# Index