Package 'acdcquery'

February 19, 2026

Title Query the Attentional Control Data Collection or the Truth Effect Database

Version 1.2.3

Description
Interact with the Attentional Control Data Collection (ACDC) or the Truth Effect Database (TED). Download the databases using download_acdc() or download_ted(), connect to the database via connect_to_db(), set filter arguments via add_argument() and query the database via query_db().

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports DBI, RSQLite, httr, jsonlite, digest

URL `https://github.com/SLesche/acdc-query`

BugReports `https://github.com/SLesche/acdc-query/issues`

NeedsCompilation no

Author Sven Lesche [aut, cre, cph],
Julia M. Haaf [ctb, ths],
Madlen Hoffstadt [ctb]

Maintainer Sven Lesche `<sven.lesche@psychologie.uni-heidelberg.de>`

Repository CRAN

Date/Publication 2026-02-19 15:40:08 UTC

Contents

`add_argument` *Add a filter argument to a list*

Description

This function adds an argument to a list containing filter arguments later used to select data from the database. When supplying the variable used for filtering, the operator and the value, an SQL query will be constructed for the user and added as the next object to the list of arguments.#' When supplying only variable, operator and value, a SQL query will be constructed for the user and added as the next object to a list. Alternatively, the user may specify an SQL query manually.

Usage

```
add_argument(list, conn, variable, operator, values, statement = NULL)
```

Arguments

`list`	The list to which the argument will be added.
`conn`	The connection object or database connection string.
`variable`	The variable name to be used in the argument.
`operator`	The operator to be used in the argument (i.e., "greater", "between", "equal", "less").
`values`	The values to be used in the argument.
`statement`	The manual SQL query to be used.

Value

A list object with the new argument (SQL query) added.

Examples

```
## Not run:
conn <- connect_to_db("path/to/db")

# Initializing argument list
arguments = list()

# Using "equal" operator
arguments = add_argument(
 list = arguments,
 conn = conn,
 variable = "cyl",
 operator = "equal",
 values = c(4, 6)
)

# Using "greater" operator
arguments = add_argument(
 list = arguments,
 conn = conn,
 variable = "cyl",
 operator = "greater",
 values = 2
)

# Using "between" operator
arguments = add_argument(
 list = arguments,
 conn = conn,
 variable = "cyl",
 operator = "between",
 values = c(2, 8)
)

# Manully constructing a filter statement
manual_arguments = add_argument(
 list = arguments,
 conn = conn,
 statement = "SELECT mtcars_id FROM mtcars WHERE cyl = 4 OR cyl = 6)"
)

## End(Not run)
```

---

add_join_paths_to_query *Add Join Paths to Query*

Description

This function generates an SQL query based on a specified connection, argument, and join path list. It constructs a query that performs joins on multiple tables according to the provided join path, incorporating requested variables and filter conditions as needed.

Usage

```
add_join_paths_to_query(
  conn,
  filter_statements,
  join_path_list,
  argument_sequence,
  requested_vars = NULL,
  filter_variables = NULL
)
```

Arguments

`conn`	The connection object or database connection string.
`filter_statements`	The SQL-Filter statements extracted from the filter arguments list via 'get_filter_statement()'.
`join_path_list`	A list representing the join path. Each element of the list should be a data frame describing a step in the join path with columns: "table_to_join", "method", and "common_var".
`argument_sequence`	A numeric vector representing the AND/OR sequence of arguments.
`requested_vars`	A character vector specifying the variables to be selected from the final query result. If `NULL`, all variables are selected.
`filter_variables`	A character vector specifying the variables needed for filtering

Value

A SQL query string that represents the joined tables and requested variables.

`check_acdc` *Check local ACDC database against GitHub release*

Description

Compares the SHA256 hash of a local ACDC SQLite database (`acdc.db`) with the hash stored in the latest (or specified) release on `jstbcs/acdc-database`.

Usage

```
check_acdc(
  local_sqlite_path,
  tag = NULL,
  token = Sys.getenv("GITHUB_PAT"),
  algo = "sha256"
)
```

Arguments

`local_sqlite_path`	Path to the local `acdc.db` file.
`tag`	Optional release tag. If NULL, uses latest release.
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`algo`	Hash algorithm used for comparison (default: `"sha256"`).

Value

Returns TRUE if the local database is up to date, FALSE if it is outdated.

Examples

```
## Not run:
check_acdc("acdc.db")

## End(Not run)
```

`check_operator` *Check validity of operator and values*

Description

This function checks the validity of the operator and values used in a condition.

Usage

```
check_operator(operator, values)
```

Arguments

`operator`	The operator to be checked.
`values`	The values to be checked.

Value

NULL (no explicit return value).

check_ted — *Check local TED database against GitHub release*

Description

Compares the SHA256 hash of a local TED SQLite database (`ted.db`) with the hash stored in the latest (or specified) release on `SLesche/truth-effect-database`.

Usage

```
check_ted(
  local_sqlite_path,
  tag = NULL,
  token = Sys.getenv("GITHUB_PAT"),
  algo = "sha256"
)
```

Arguments

`local_sqlite_path`	Path to the local `ted.db` file.
`tag`	Optional release tag. If NULL, uses latest release.
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`algo`	Hash algorithm used for comparison (default: `"sha256"`).

Value

Returns TRUE if the local database is up to date, FALSE if it is outdated.

Examples

```
## Not run:
check_ted("ted.db")

## End(Not run)
```

compare_sqlite_to_release — *Compare local SQLite database to GitHub release hash*

Description

Compares the SHA256 hash of a local SQLite database file with the hash file stored in a GitHub release.

Usage

```
compare_sqlite_to_release(
  owner,
  repo,
  asset_name,
  local_sqlite_path,
  tag = NULL,
  token = Sys.getenv("GITHUB_PAT"),
  algo = "sha256"
)
```

Arguments

`owner`	Repository owner.
`repo`	Repository name.
`asset_name`	Name of the SQLite asset (used to derive hash filename).
`local_sqlite_path`	Path to local SQLite database.
`tag`	Optional release tag. If NULL, uses latest release.
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`algo`	Hash algorithm (default: `"sha256"`).

Details

This avoids downloading the full database when checking for updates.

Value

TRUE if hashes match, FALSE otherwise.

Examples

```
## Not run:
compare_sqlite_to_release(
  owner = "myuser",
  repo = "myrepo",
  asset_name = "database.sqlite",
  local_sqlite_path = "database.sqlite"
)

## End(Not run)
```

`connect_to_db` *Connect to an SQLite database*

Description

This function establishes a connection to an SQLite database file located at the specified path using the DBI and RSQLite packages.

Usage

```
connect_to_db(path_to_db)
```

Arguments

`path_to_db`	The path to the SQLite database file.

Value

A database connection object.

Examples

```
# When connecting to a specific file, like the downloaded ACDC-Database
# just use the path to the database
# If the specified database file does not exist, an error will be thrown.
# Consider downloading the database using `download_acdc()` or `download_ted()`.
## Not run: conn <- connect_to_db("path/to/database.db")
```

`create_sqlite_hash_file` *Create SHA256 hash file for a SQLite database*

Description

Computes a SHA256 hash for a SQLite database file and writes it to a `.sha256` file. The hash file can be uploaded alongside the database in a GitHub release to allow version comparison without downloading the full database.

Usage

```
create_sqlite_hash_file(sqlite_path, output_file = NULL, algo = "sha256")
```

Arguments

`sqlite_path`	Path to the SQLite file.
`output_file`	Optional output path for the hash file. Defaults to `<sqlite_path>.sha256`.
`algo`	Hash algorithm (default: `"sha256"`).

Value

Invisibly returns the path to the hash file.

Examples

```
## Not run:
create_sqlite_hash_file("database.sqlite")

## End(Not run)
```

`discover_id_introduction_steps`

Discover ID Introduction Steps

Description

This function identifies the steps in a join path where new IDs are introduced, allowing you to determine at which join steps each ID variable is added to the query. It returns a data frame with information about newly discovered IDs and the corresponding join step in the path.

Usage

```
discover_id_introduction_steps(conn, full_path_dataframe)
```

Arguments

`conn`	The connection object or database connection string.
`full_path_dataframe`	A data frame representing the full join path, including columns: "table_to_join", "method", and "common_var".

Value

A data frame with information about newly discovered IDs and the corresponding join step.

download_acdc *Download ACDC database from GitHub release*

Description

Downloads the `acdc.db` SQLite database from the latest (or specified) release on `jstbcs/acdc-database` to a given directory.

Usage

```
download_acdc(dest_dir, tag = NULL, token = Sys.getenv("GITHUB_PAT"))
```

Arguments

`dest_dir`	Directory where the database should be saved.
`tag`	Optional release tag. If NULL, uses latest release.
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.

Value

Invisibly returns the full path to the downloaded file.

Examples

```
## Not run:
download_acdc(tempdir())

## End(Not run)
```

download_sqlite_release *Download SQLite file from GitHub release*

Description

Downloads a SQLite database file from the latest or a specific GitHub release.

Usage

```
download_sqlite_release(
  owner,
  repo,
  asset_name,
  dest_dir,
  tag = NULL,
  algo = "sha256",
  token = Sys.getenv("GITHUB_PAT"),
  overwrite = TRUE
)
```

Arguments

`owner`	Repository owner.
`repo`	Repository name.
`asset_name`	Name of the SQLite asset in the release.
`dest_dir`	Local directory where the file should be saved.
`tag`	Optional release tag (e.g., `"v1.2.0"`). If NULL, the latest release is used.
`algo`	Hash algorithm (default: `"sha256"`).
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`overwrite`	Whether to overwrite existing file (default: TRUE).

Value

Invisibly returns the full path to the downloaded file.

Examples

```
## Not run:
download_sqlite_release(
  owner = "myuser",
  repo = "myrepo",
  asset_name = "database.sqlite",
  dest_dir = tempdir()
)

## End(Not run)
```

`download_ted` *Download TED database from GitHub release*

Description

Downloads the `ted.db` SQLite database from the latest (or specified) release on `SLesche/truth-effect-database` to a given directory.

Usage

```
download_ted(dest_dir, tag = NULL, token = Sys.getenv("GITHUB_PAT"))
```

Arguments

`dest_dir`	Directory where the database should be saved.
`tag`	Optional release tag. If NULL, uses latest release.
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.

Value

Invisibly returns the full path to the downloaded file.

Examples

```
## Not run:
download_ted(tempdir())

## End(Not run)
```

`find_relevant_tables` *Find relevant tables based on column name*

Description

This function finds the relevant database tables that contain a specified column.

Usage

```
find_relevant_tables(conn, column_name, info = NULL, strict = FALSE)
```

Arguments

`conn`	The connection object or database connection string.
`column_name`	The name of the column to search for in the database tables.
`info`	Optional. The information data frame obtained from `get_column_names()` function. If not provided, it will be obtained within the function.
`strict`	Should only one table be returned? Relevant for id variables

Value

A character vector containing the names of the relevant tables.

`get_argument_sequence` *Get argument sequence based on argument relation*

Description

This function returns the sequence of arguments based on the specified argument relation. The argument relation determines the logical relationship between the arguments (e.g., "and", "or").

Usage

```
get_argument_sequence(arguments, argument_relation)
```

Arguments

`arguments`	The list of arguments.
`argument_relation`	The specified argument relation. If "and", the sequence will be 1:length(arguments). If "or", the sequence will be rep(1, length(arguments)). If a vector is provided, it should have the same length as the number of arguments.

Value

A numeric vector representing the sequence of arguments.

`get_column_names` *Get column names from database tables*

Description

This function retrieves the column names from all tables in the specified database connection.

Usage

```
get_column_names(conn)
```

Arguments

`conn`	The connection object or database connection string.

Value

A data frame containing the column names and corresponding table names.

`get_filter_statement` *Get Filter Statement*

Description

This function constructs a SQL filter statement based on the provided filter statements and argument sequence.

Usage

```
get_filter_statement(filter_statements, argument_sequence, introduction_table)
```

Arguments

`filter_statements`	A character vector of SQL filter statements, one for each argument in the argument sequence.
`argument_sequence`	A numeric vector representing the argument sequence for constructing the filter statement.
`introduction_table`	A data frame containing information about table prefixes for ID variables.

Value

A character string representing the constructed SQL filter statement.

`make_valid_sql` *Create a valid SQL statement based on variable, operator, and values*

Description

This function creates a valid SQL statement based on the specified variable, operator, and values. It handles different operators such as "greater", "less", "equal", and "between".

Usage

```
make_valid_sql(conn, variable, operator, values)
```

Arguments

`conn`	The connection object or database connection string.
`variable`	The variable for which the SQL statement is created.
`operator`	The operator to be used in the SQL statement.
`values`	The values to be used in the SQL statement.

Value

A character string representing the valid SQL statement.

`precompute_table_join_paths` *Precompute Table Join Paths*

Description

This function precomputes join paths for all tables in a given database using a combination of forward and backward joins. It generates a list of data frames representing the join paths for each table, including information about tables to join, walk approaches (forward or backward), and common variables used for joining.

Usage

```
precompute_table_join_paths(conn, input_table = NULL, relevant_tables = NULL)
```

Arguments

`conn`	The connection object or database connection string.
`input_table`	The table from which the join path is computed.
`relevant_tables`	A vector of tables that are relevant to the query.

Value

A list of join paths for each table in the database.

`query_db` *Query Database*

Description

This function performs targeted queries on an SQLite database using specified filtering arguments and returns the query results. It extracts information about which tables of the database are relevant for the query and then joins these relevant tables to the target table. The function constructs an SQL query which incorporates both the joining and filtering target variables. This SQL statement is then applied to the database and the resulting dataframe is returned to the user.

Usage

```
query_db(
  conn,
  arguments,
  target_vars = "default",
  target_table = "observation_table",
  argument_relation = "and"
)
```

Arguments

`conn`	The connection object to an SQLite database.
`arguments`	A list of filtering arguments for the query. The list must have only one filter argument per list-entry.
`target_vars`	A character vector specifying the variables to be included in the query results.
`target_table`	The target table in the database for querying.
`argument_relation`	A character string specifying the relation between filtering arguments ("and" or "or" or a numerical vector with the same length as the number of arguments). Arguments with equal numbers in their index are joined using the OR operator, others using AND. To represent (A OR B) AND C AND D use the vector c(1, 1, 2, 3).

Value

A data frame containing the query results.

Examples

```
## Not run:
conn <- connect_to_db("path/to/database.db")

arguments <- add_argument(list(), "publication_id", "greater", 0)

target_vars <- c("default", "n_participants")

target_table <- "study_table"

results <- query_db(conn, arguments, target_vars, target_table)

## End(Not run)
```

`return_id_name_from_table` *Return ID column name from table name*

Description

This function generates the ID column name based on the provided table name. It replaces the "table" suffix with "id" to obtain the ID column name.

Usage

```
return_id_name_from_table(table_name)
```

Arguments

`table_name`	The name of the table.

Value

The generated ID column name.

`return_table_name_from_id` *Return table name from ID column name*

Description

This function generates the table name based on the provided ID column name. It replaces the "id" suffix with "table" to obtain the table name.

Usage

```
return_table_name_from_id(id_name)
```

Arguments

`id_name`	The name of the ID column.

Value

The generated table name.

update_acdc *Update local ACDC database from GitHub release*

Description

Checks whether the local ACDC SQLite database (`acdc.db`) is up to date with the release on `jstbcs/acdc-database` and optionally downloads the latest version.

Usage

```
update_acdc(
  local_sqlite_path,
  tag = NULL,
  update = FALSE,
  token = Sys.getenv("GITHUB_PAT"),
  algo = "sha256"
)
```

Arguments

`local_sqlite_path`	Path to the local `acdc.db` file.
`tag`	Optional release tag. If NULL, uses latest release.
`update`	Logical; if TRUE, will download the latest database when the local copy is outdated (default: FALSE).
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`algo`	Hash algorithm (default: `"sha256"`).

Value

Invisibly returns TRUE if up to date, FALSE if outdated.

Examples

```
## Not run:
update_acdc("acdc.db")
update_acdc("acdc.db", update = TRUE)

## End(Not run)
```

`update_database` *Update a local SQLite database from a GitHub release*

Description

Internal helper function that compares a local SQLite database with a GitHub release and optionally downloads the latest version.

Usage

```
update_database(
  owner,
  repo,
  asset_name,
  local_sqlite_path,
  tag = NULL,
  token = Sys.getenv("GITHUB_PAT"),
  algo = "sha256",
  update = FALSE
)
```

Arguments

`owner`	GitHub repository owner.
`repo`	GitHub repository name.
`asset_name`	Name of the SQLite file in the release.
`local_sqlite_path`	Path to the local SQLite database.
`tag`	Optional release tag. If NULL, uses latest release.
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`algo`	Hash algorithm to use (default: `"sha256"`).
`update`	Logical; if TRUE, will download the latest database when the local copy is outdated (default: FALSE).

Value

Invisibly returns TRUE if the local database is up to date, FALSE if it is outdated (and possibly updated if `update = TRUE`).

`update_ted` *Update local TED database from GitHub release*

Description

Checks whether the local TED SQLite database (`ted.db`) is up to date with the release on `SLesche/truth-effect-database` and optionally downloads the latest version.

Usage

```
update_ted(
  local_sqlite_path,
  tag = NULL,
  update = FALSE,
  token = Sys.getenv("GITHUB_PAT"),
  algo = "sha256"
)
```

Arguments

`local_sqlite_path`	Path to the local `ted.db` file.
`tag`	Optional release tag. If NULL, uses latest release.
`update`	Logical; if TRUE, will download the latest database when the local copy is outdated (default: FALSE).
`token`	Optional GitHub token. Defaults to `Sys.getenv("GITHUB_PAT")`.
`algo`	Hash algorithm (default: `"sha256"`).

Value

Invisibly returns TRUE if up to date, FALSE if outdated.

Examples

```
## Not run:
update_ted("ted.db")
update_ted("ted.db", update = TRUE)

## End(Not run)
```

Index