

Package ‘csdb’

May 8, 2026

Title An Abstracted System for Easily Working with Databases with Large Datasets

Version 2026.2.2

Description

Provides object-oriented database management tools for working with large datasets across multiple database systems. Features include robust connection management for 'PostgreSQL' databases, advanced table operations with bulk data loading and upsert functionality, comprehensive data validation through customizable field type and content validators, efficient index management, and cross-database compatibility. Designed for high-performance data operations in surveillance systems and large-scale data processing workflows.

Depends R (>= 4.1.0)

License MIT + file LICENSE

URL <https://niphr.github.io/csdb/>, <https://github.com/niphr/csdb>

BugReports <https://github.com/niphr/csdb/issues>

Encoding UTF-8

LazyData true

Imports csutil, data.table, DBI, dplyr, fs, ggplot2, glue, methods, odbc, R6, S7, stringr, uuid

Suggests testthat, knitr, rmarkdown, rstudioapi, digest, crayon

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Richard Aubrey White [aut, cre] (ORCID: <https://orcid.org/0000-0002-6747-1726>), August Sørli Mathisen [aut]

Maintainer Richard Aubrey White <hello@rwhite.no>

Repository CRAN

Date/Publication 2026-03-31 11:10:02 UTC

Contents

csdb_get_auth_hook	2
csdb_set_auth_hook	2
DBConnection_v9	3
DBTable_v9	6
get_table_names_and_info	12
nor_covid19_cases_by_time_location	14
validator_field_contents_blank	15
validator_field_contents_csfmt_rts_data_v1	15
validator_field_contents_csfmt_rts_data_v2	16
validator_field_types_blank	17
validator_field_types_csfmt_rts_data_v1	18
validator_field_types_csfmt_rts_data_v2	19

Index	20
--------------	-----------

csdb_get_auth_hook	<i>Get the current authentication hook</i>
--------------------	--

Description

Returns the currently registered authentication hook function.

Usage

```
csdb_get_auth_hook()
```

Value

The current auth hook function, or NULL if none is set.

csdb_set_auth_hook	<i>Set authentication hook for database connections</i>
--------------------	---

Description

Register a function to be called when a database connection fails. This is useful for refreshing Kerberos tickets or other authentication credentials before retrying the connection.

Usage

```
csdb_set_auth_hook(hook)
```

Arguments

hook	A function with no arguments that performs authentication, or NULL to clear the hook.
------	---

Value

Invisibly returns the previous hook (if any).

Examples

```
## Not run:
# Set an auth hook to refresh Kerberos credentials
csdb_set_auth_hook(function() {
  system2("/bin/authenticate.sh", stdout = NULL)
})

# Clear the hook
csdb_set_auth_hook(NULL)

## End(Not run)
```

DBConnection_v9

R6 Class representing a database connection

Description

A robust database connection manager that handles connections to various database systems including Microsoft SQL Server and PostgreSQL. This class provides connection management, authentication, and automatic reconnection capabilities.

Details

The `DBConnection_v9` class encapsulates database connection logic and provides a consistent interface for connecting to different database systems. It supports both trusted connections and user/password authentication, handles connection failures gracefully, and provides automatic reconnection functionality.

Key features:

- Support for multiple database systems (SQL Server, PostgreSQL)
- Automatic connection management with retry logic
- Secure credential handling
- Connection status monitoring
- Graceful error handling and recovery

Public fields

`config` Configuration details of the database.

Active bindings

`connection` Database connection.

`autoconnection` Database connection that automatically connects if possible.

Methods

Public methods:

- [DBConnection_v9\\$new\(\)](#)
- [DBConnection_v9\\$is_connected\(\)](#)
- [DBConnection_v9\\$print\(\)](#)
- [DBConnection_v9\\$connect\(\)](#)
- [DBConnection_v9\\$disconnect\(\)](#)
- [DBConnection_v9\\$clone\(\)](#)

Method `new()`: Create a new `DBConnection_v9` object.

Usage:

```
DBConnection_v9$new(  
  driver = NULL,  
  server = NULL,  
  port = NULL,  
  db = NULL,  
  schema = NULL,  
  user = NULL,  
  password = NULL,  
  trusted_connection = NULL,  
  sslmode = NULL,  
  role_create_table = NULL  
)
```

Arguments:

`driver` Driver

`server` Server

`port` Port

`db` DB

`schema` Schema (e.g. "dbo")

`user` User

`password` Password

`trusted_connection` NULL or "yes"

`sslmode` NULL or "require"

`role_create_table` NULL or the role to take when creating tables.

Returns: A new 'DBConnection_v9' object.

Method `is_connected()`: Is the DB schema connected?

Usage:

```
DBConnection_v9$is_connected()
```

Returns: TRUE/FALSE

Method `print()`: Class-specific print function.

Usage:

```
DBConnection_v9$print(...)
```

Arguments:

... Not used.

Method connect(): Connect to the database

Usage:

```
DBConnection_v9$connect(attempts = 2)
```

Arguments:

attempts Number of attempts to be made to try to connect

Method disconnect(): Disconnect from the database

Usage:

```
DBConnection_v9$disconnect()
```

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

Usage:

```
DBConnection_v9$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
# Create a SQL Server connection
db_config <- DBConnection_v9$new(
  driver = "ODBC Driver 17 for SQL Server",
  server = "localhost",
  port = 1433,
  db = "mydb",
  user = "myuser",
  password = "mypass"
)

# Connect to the database
db_config$connect()

# Check connection status
db_config$is_connected()

# Use the connection
tables <- DBI::dbListTables(db_config$connection)

# Disconnect when done
db_config$disconnect()

# PostgreSQL example
pg_config <- DBConnection_v9$new(
  driver = "PostgreSQL",
```

```
server = "localhost",
port = 5432,
db = "mydb",
user = "myuser",
password = "mypass"
)

pg_config$connect()
# ... use connection ...
pg_config$disconnect()

## End(Not run)
```

DBTable_v9

R6 Class representing a database table with advanced data management capabilities

Description

A comprehensive database table management class that provides high-level operations for data manipulation, schema validation, and table administration. This class combines database connectivity with data validation and efficient bulk operations.

Details

The DBTable_v9 class is a sophisticated database table abstraction that provides:

Core functionality:

- Table creation and schema management
- Data insertion with bulk loading capabilities
- Upsert operations (insert or update)
- Index management (creation, deletion)
- Data validation through customizable validators
- Integration with dplyr for data queries

Advanced features:

- Automatic table creation based on field specifications
- Schema validation with custom validator functions
- Efficient bulk data loading using database-specific methods
- Index optimization for query performance
- Cross-database compatibility (SQL Server, PostgreSQL)

Data validation: The class supports custom validation functions for both field types and data contents, ensuring data integrity and schema compliance.

Public fields

dbconnection Database connection.

dbconfig Configuration details of the database.

table_name Name of the table in the database.

table_name_short_for_mssql_fully_specified_for_postgres Fully specified name of the table in the database (e.g. \[db\].\[dbo\].\[table_name\]).

table_name_short_for_mssql_fully_specified_for_postgres_text Fully specified name of the table in the database (e.g. \[db\].\[dbo\].\[table_name\]).

table_name_fully_specified Fully specified name of the table in the database (e.g. \[db\].\[dbo\].\[table_name\]).

table_name_fully_specified_text Fully specified name of the table in the database (e.g. \[db\].\[dbo\].\[table_name\]) as a text string.

field_types The types of each column in the database table (INTEGER, DOUBLE, TEXT, BOOLEAN, DATE, DATETIME).

field_types_with_length The same as field_types but with (100) added to the end of all TEXT fields.

keys The combination of variables that uniquely identify each row in the database.

keys_with_length The same as keys but with (100) added to the end of all TEXT fields.

indexes A named list of vectors (generally "ind1", "ind2", etc.) that improves the speed of data retrieval operations on a database table.

validator_field_contents A function that validates the data before it is inserted into the database.

load_folder A temporary folder that is used to write data to before inserting into the database.

censors A named list of censors.

Methods**Public methods:**

- `DBTable_v9$new()`
- `DBTable_v9$print()`
- `DBTable_v9$connect()`
- `DBTable_v9$disconnect()`
- `DBTable_v9$table_exists()`
- `DBTable_v9$create_table()`
- `DBTable_v9$remove_table()`
- `DBTable_v9$insert_data()`
- `DBTable_v9$upsert_data()`
- `DBTable_v9$drop_all_rows()`
- `DBTable_v9$drop_rows_where()`
- `DBTable_v9$keep_rows_where()`
- `DBTable_v9$drop_all_rows_and_then_upsert_data()`
- `DBTable_v9$drop_all_rows_and_then_insert_data()`
- `DBTable_v9$tbl()`

- DBTable_v9\$print_dplyr_select()
- DBTable_v9\$add_indexes()
- DBTable_v9\$drop_indexes()
- DBTable_v9\$confirm_indexes()
- DBTable_v9\$nrow()
- DBTable_v9\$info()
- DBTable_v9\$clone()

Method `new()`: Create a new DBTable_v9 object.

Usage:

```
DBTable_v9$new(
  dbconfig,
  table_name,
  field_types,
  keys,
  indexes = NULL,
  validator_field_types = validator_field_types_blank,
  validator_field_contents = validator_field_contents_blank
)
```

Arguments:

`dbconfig` Configuration details of the database (driver, server, port, db, schema, user, password, trusted_connection, sslmode, role_create_table).

`table_name` Name of the table in the database.

`field_types` The types of each column in the database table (INTEGER, DOUBLE, TEXT, BOOLEAN, DATE, DATETIME).

`keys` The combination of these variables uniquely identifies each row of data in the table.

`indexes` A named list of vectors (generally "ind1", "ind2", etc.) that improves the speed of data retrieval operations on a database table.

`validator_field_types` A function that validates the `field_types` before the DB schema is created.

`validator_field_contents` A function that validates the data before it is inserted into the database.

Returns: A new 'DBTable_v9' object.

Method `print()`: Class-specific print function.

Usage:

```
DBTable_v9$print(...)
```

Arguments:

... Not in use.

Method `connect()`: Connect from the database

Usage:

```
DBTable_v9$connect()
```

Method `disconnect()`: Disconnect from the database

Usage:

```
DBTable_v9$disconnect()
```

Method table_exists(): Does the table exist

Usage:

```
DBTable_v9$table_exists()
```

Method create_table(): Create the database table

Usage:

```
DBTable_v9$create_table()
```

Method remove_table(): Drop the database table

Usage:

```
DBTable_v9$remove_table()
```

Method insert_data(): Inserts data

Usage:

```
DBTable_v9$insert_data(  
  newdata,  
  confirm_insert_via_nrow = FALSE,  
  verbose = TRUE  
)
```

Arguments:

newdata The data to insert.

confirm_insert_via_nrow Checks nrow() before insert and after insert. If nrow() has not increased sufficiently, then attempt an upsert.

verbose Boolean. Inserts data into the database table

Method upsert_data(): Upserts data into the database table

Usage:

```
DBTable_v9$upsert_data(  
  newdata,  
  drop_indexes = names(self$indexes),  
  verbose = TRUE  
)
```

Arguments:

newdata The data to insert.

drop_indexes A vector containing the indexes to be dropped before upserting (can increase performance).

verbose Boolean.

Method drop_all_rows(): Drops all rows in the database table

Usage:

```
DBTable_v9$drop_all_rows()
```

Method `drop_rows_where()`: Drops rows in the database table according to the SQL condition.

Usage:

```
DBTable_v9$drop_rows_where(condition)
```

Arguments:

`condition` SQL text condition.

Method `keep_rows_where()`: Keeps rows in the database table according to the SQL condition.

Usage:

```
DBTable_v9$keep_rows_where(condition)
```

Arguments:

`condition` SQL text condition.

Method `drop_all_rows_and_then_upsert_data()`: Drops all rows in the database table and then upserts data.

Usage:

```
DBTable_v9$drop_all_rows_and_then_upsert_data(
  newdata,
  drop_indexes = names(self$indexes),
  verbose = TRUE
)
```

Arguments:

`newdata` The data to insert.

`drop_indexes` A vector containing the indexes to be dropped before upserting (can increase performance).

`verbose` Boolean.

Method `drop_all_rows_and_then_insert_data()`: Drops all rows in the database table and then inserts data.

Usage:

```
DBTable_v9$drop_all_rows_and_then_insert_data(
  newdata,
  confirm_insert_via_nrow = FALSE,
  verbose = TRUE
)
```

Arguments:

`newdata` The data to insert.

`confirm_insert_via_nrow` Checks `nrow()` before insert and after insert. If `nrow()` has not increased sufficiently, then attempt an upsert.

`verbose` Boolean.

Method `tbl()`: Provides access to the database table via `dplyr::tbl`.

Usage:

```
DBTable_v9$tbl()
```

Method `print_dplyr_select()`: Prints a template `dplyr::select` call that you can easily copy/paste for all your variables.

Usage:

```
DBTable_v9$print_dplyr_select()
```

Method `add_indexes()`: Adds indexes to the database table from 'self\$indexes'

Usage:

```
DBTable_v9$add_indexes()
```

Method `drop_indexes()`: Drops all indexes from the database table

Usage:

```
DBTable_v9$drop_indexes()
```

Method `confirm_indexes()`: Confirms that the names and number of indexes in the database are the same as in the R code. Does not confirm the contents of the indexes!

Usage:

```
DBTable_v9$confirm_indexes()
```

Method `nrow()`: Gets the number of rows in the database table

Usage:

```
DBTable_v9$nrow(use_count = FALSE)
```

Arguments:

`use_count` If true, then uses the count command, which is slow but accurate. If false, then uses summary statistics, which is fast but inaccurate.

Method `info()`: Gets the information about the database table

Usage:

```
DBTable_v9$info()
```

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

Usage:

```
DBTable_v9$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Create database connection
db_config <- list(
  driver = "ODBC Driver 17 for SQL Server",
  server = "localhost",
  db = "mydb",
  user = "myuser",
  password = "mypass"
)
```

```
# Define table schema
field_types <- c(
  "id" = "INTEGER",
  "name" = "TEXT",
  "value" = "DOUBLE",
  "date_created" = "DATE"
)

# Create table object
my_table <- DBTable_v9$new(
  dbconfig = db_config,
  table_name = "my_data_table",
  field_types = field_types,
  keys = c("id"),
  validator_field_types = validator_field_types_blank,
  validator_field_contents = validator_field_contents_blank
)

# Create table in database
my_table$create_table()

# Insert data
sample_data <- data.frame(
  id = 1:3,
  name = c("Alice", "Bob", "Charlie"),
  value = c(10.5, 20.3, 15.7),
  date_created = as.Date("2023-01-01")
)
my_table$insert_data(sample_data)

# Query data using dplyr
result <- my_table$tbl() |>
  dplyr::filter(value > 15) |>
  dplyr::collect()

# Add indexes for performance
my_table$add_indexes(c("name", "date_created"))

# Upsert (insert or update) data
new_data <- data.frame(
  id = 2:4,
  name = c("Bob_Updated", "Charlie", "David"),
  value = c(25.0, 15.7, 30.2),
  date_created = as.Date("2023-01-02")
)
my_table$upsert_data(new_data)

## End(Not run)
```

get_table_names_and_info

Get table names, number of rows, and size information

Description

Retrieves comprehensive information about database tables including their names, row counts, and storage size metrics. This function provides database-specific implementations for different database systems.

Usage

```
get_table_names_and_info(connection)
```

Arguments

connection A database connection object (e.g., from [dbConnect](#))

Value

A data.table containing table information with columns:

table_name Character. Name of the table

nrow Numeric. Number of rows in the table

size_total_gb Numeric. Total size of the table in gigabytes

size_data_gb Numeric. Size of data in gigabytes

size_index_gb Numeric. Size of indexes in gigabytes

Examples

```
## Not run:
# Microsoft SQL Server example
con <- DBI::dbConnect(odbc::odbc(),
                     driver = "ODBC Driver 17 for SQL Server",
                     server = "localhost",
                     database = "mydb")
table_info <- get_table_names_and_info(con)
print(table_info)
DBI::dbDisconnect(con)

# PostgreSQL example
con <- DBI::dbConnect(RPostgres::Postgres(),
                     host = "localhost",
                     dbname = "mydb",
                     user = "user")
table_info <- get_table_names_and_info(con)
print(table_info)
DBI::dbDisconnect(con)

## End(Not run)
```

nor_covid19_cases_by_time_location

Covid-19 data for PCR-confirmed cases in Norway (nation and county)

Description

This data comes from the Norwegian Surveillance System for Communicable Diseases (MSIS). The date corresponds to when the PCR-test was taken.

Usage

nor_covid19_cases_by_time_location

Format

A csfmt_rts_data_v1 with 11028 rows and 18 variables:

granularity_time day/isoweek

granularity_geo nation, county

country_iso3 nor

location_code norge, 11 counties

border 2020

age total

isoyear Isoyear of event

isoweek Isoweek of event

isoyearweek Isoyearweek of event

season Season of event

seasonweek Seasonweek of event

calyear Calyear of event

calmonth Calmonth of event

calyearmonth Calyearmonth of event

date Date of event

covid19_cases_testdate_n Number of confirmed covid19 cases

covid19_cases_testdate_pr100000 Number of confirmed covid19 cases per 100.000 population

Details

The raw number of cases and cases per 100.000 population are recorded.

This data was extracted on 2022-05-04.

Source

https://github.com/folkehelseinstituttet/surveillance_data/blob/master/covid19/_DOCUMENTATION_data_covid19_msis_by_time_location.txt

```
validator_field_contents_blank
```

Blank data contents validator

Description

A pass-through validator that accepts any data without validation. This is useful as a placeholder when no specific data content validation is needed.

Usage

```
validator_field_contents_blank(data)
```

Arguments

`data` A data.frame or data.table containing the data to validate

Value

Always returns TRUE

Examples

```
# This validator always returns TRUE regardless of input
test_data <- data.frame(id = 1:3, name = c("A", "B", "C"), value = c(10, 20, 30))
validator_field_contents_blank(test_data)

# Works with any data structure
empty_data <- data.frame()
validator_field_contents_blank(empty_data)
```

```
validator_field_contents_csfmt_rts_data_v1
```

Field contents validator for csfmt_rts_data_v1 schema

Description

Validates that data contents conform to the csfmt_rts_data_v1 schema specification. This validator checks that `granularity_time` and `granularity_geo` fields contain valid values according to the surveillance data format requirements.

Usage

```
validator_field_contents_csfmt_rts_data_v1(data)
```

Arguments

data A data.frame or data.table containing the data to validate

Value

TRUE if data is valid for csfmt_rts_data_v1, FALSE otherwise (with error attribute)

Examples

```
# Valid data for csfmt_rts_data_v1
valid_data <- data.frame(
  granularity_time = c("date", "isoyearweek", "total"),
  granularity_geo = c("nation", "county", "municip"),
  stringsAsFactors = FALSE
)
validator_field_contents_csfmt_rts_data_v1(valid_data)

# Invalid data (wrong granularity_geo value)
invalid_data <- data.frame(
  granularity_time = "date",
  granularity_geo = "invalid_geo",
  stringsAsFactors = FALSE
)
validator_field_contents_csfmt_rts_data_v1(invalid_data)
```

validator_field_contents_csfmt_rts_data_v2

Field contents validator for csfmt_rts_data_v2 schema

Description

Validates that data contents conform to the csfmt_rts_data_v2 schema specification. This validator checks that granularity_time and granularity_geo fields contain valid values according to the surveillance data format requirements for version 2.

Usage

```
validator_field_contents_csfmt_rts_data_v2(data)
```

Arguments

data A data.frame or data.table containing the data to validate

Value

TRUE if data is valid for csfmt_rts_data_v2, FALSE otherwise (with error attribute)

Examples

```
# Valid data for csfmt_rts_data_v2
valid_data_v2 <- data.frame(
  granularity_time = c("date", "isoyearweek", "total"),
  granularity_geo = c("nation", "county", "municip"),
  stringsAsFactors = FALSE
)
validator_field_contents_csfmt_rts_data_v2(valid_data_v2)
```

validator_field_types_blank
Blank field types validator

Description

A pass-through validator that accepts any field types without validation. This is useful as a placeholder when no specific field type validation is needed.

Usage

```
validator_field_types_blank(db_field_types)
```

Arguments

db_field_types A named character vector of database field types

Value

Always returns TRUE

Examples

```
# This validator always returns TRUE regardless of input
field_types <- c("id" = "INTEGER", "name" = "TEXT", "date" = "DATE")
validator_field_types_blank(field_types)

# Works with any field types
other_types <- c("value" = "DOUBLE", "status" = "BOOLEAN")
validator_field_types_blank(other_types)
```

```
validator_field_types_csfmt_rts_data_v1
```

Field types validator for csfmt_rts_data_v1 schema

Description

Validates that field types conform to the `csfmt_rts_data_v1` schema specification. This validator ensures that the first 16 fields match the expected structure for real-time surveillance data format version 1.

Usage

```
validator_field_types_csfmt_rts_data_v1(db_field_types)
```

Arguments

`db_field_types` A named character vector of database field types

Value

TRUE if field types are valid for `csfmt_rts_data_v1`, FALSE otherwise

Examples

```
# Valid field types for csfmt_rts_data_v1
valid_fields <- c(
  "granularity_time" = "TEXT",
  "granularity_geo" = "TEXT",
  "country_iso3" = "TEXT",
  "location_code" = "TEXT",
  "border" = "INTEGER",
  "age" = "TEXT",
  "sex" = "TEXT",
  "isoyear" = "INTEGER",
  "isoweek" = "INTEGER",
  "isoyearweek" = "TEXT",
  "season" = "TEXT",
  "seasonweek" = "DOUBLE",
  "calyear" = "INTEGER",
  "calmonth" = "INTEGER",
  "calyearmonth" = "TEXT",
  "date" = "DATE",
  "cases_n" = "INTEGER"
)
validator_field_types_csfmt_rts_data_v1(valid_fields)

# Invalid field types (wrong structure)
invalid_fields <- c("id" = "INTEGER", "name" = "TEXT")
validator_field_types_csfmt_rts_data_v1(invalid_fields)
```

`validator_field_types_csfmt_rts_data_v2`*Field types validator for csfmt_rts_data_v2 schema*

Description

Validates that field types conform to the `csfmt_rts_data_v2` schema specification. This validator ensures that the first 18 fields match the expected structure for real-time surveillance data format version 2.

Usage

```
validator_field_types_csfmt_rts_data_v2(db_field_types)
```

Arguments

`db_field_types` A named character vector of database field types

Value

TRUE if field types are valid for `csfmt_rts_data_v2`, FALSE otherwise

Examples

```
# Valid field types for csfmt_rts_data_v2 (includes additional fields)
valid_fields_v2 <- c(
  "granularity_time" = "TEXT",
  "granularity_geo" = "TEXT",
  "country_iso3" = "TEXT",
  "location_code" = "TEXT",
  "border" = "INTEGER",
  "age" = "TEXT",
  "sex" = "TEXT",
  "isoyear" = "INTEGER",
  "isoweek" = "INTEGER",
  "isoyearweek" = "TEXT",
  "season" = "TEXT",
  "seasonweek" = "DOUBLE",
  "calyear" = "INTEGER",
  "calmonth" = "INTEGER",
  "calyearmonth" = "TEXT",
  "date" = "DATE",
  "tag_outcome" = "TEXT",
  "tag_type" = "TEXT",
  "cases_n" = "INTEGER"
)
validator_field_types_csfmt_rts_data_v2(valid_fields_v2)
```

Index

* datasets

nor_covid19_cases_by_time_location,
[14](#)

csdb_get_auth_hook, [2](#)
csdb_set_auth_hook, [2](#)

dbConnect, [13](#)
DBConnection_v9, [3](#)
DBTable_v9, [6](#)

get_table_names_and_info, [12](#)

nor_covid19_cases_by_time_location, [14](#)

validator_field_contents_blank, [15](#)
validator_field_contents_csfmt_rts_data_v1,
[15](#)
validator_field_contents_csfmt_rts_data_v2,
[16](#)
validator_field_types_blank, [17](#)
validator_field_types_csfmt_rts_data_v1,
[18](#)
validator_field_types_csfmt_rts_data_v2,
[19](#)