

Package: certedb (via r-universe)

October 1, 2026

Title A Certe R Package for Connecting to Databases

Version 1.15.0

Description A Certe R Package for connecting to internal Certe cBases and local DuckDB databases. This package is part of the 'certedata' universe.

URL <https://certe-medical-epidemiology.github.io/certedb>,
<https://github.com/certe-medical-epidemiology/certedb>

Depends R (>= 4.1.0)

Imports certestyle, certetoolbox, AMR (>= 2.0.0), cli (>= 3.4.0), DBI (>= 1.0.0), dbplyr (>= 2.1.0), dplyr (>= 1.0.0), DT (>= 0.3.0), duckdb (>= 0.8.0), knitr (>= 1.40), odbc (>= 1.3.0), pillar (>= 1.8.0), shiny (>= 1.9.0), RSQLite (>= 2.3.0), rlang (>= 1.1.0), tidyr (>= 1.1.0), yaml (>= 2.2.1)

Suggests callr, clipr (>= 0.8.0), connections, RMariaDB, rstudioapi, sf, stringr, testthat (>= 2.0.0), utils

License GPL-2

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 2

Config/pak/sysreqs cmake libicu-dev libuv1-dev libxml2-dev unixodbc-dev libssl-dev libx11-dev xz-utils zlib1g-dev

Repository <https://certe-medical-epidemiology.r-universe.dev>

Date/Publication 2026-07-16 10:29:49 UTC

RemoteUrl <https://github.com/certe-medical-epidemiology/certedb>

RemoteRef HEAD

RemoteSha 4cf17befef8fbc044b7a2dd80fe6c7de14a4d610

Contents

cbase_explorer	2
db_connect	3
get_diver_data	4
mysql_init	10
presets	10
read_query_log	13
Index	15

cbase_explorer	<i>Search cBase Interactively</i>
----------------	-----------------------------------

Description

Use this Shiny app to search a cBase. There is also an RStudio add-in.

Usage

```
cbase_explorer(  
  preset = read_secret("db.preset_default_shiny"),  
  diver_cbase = NULL,  
  diver_project = read_secret("db.diver_project"),  
  diver_dsn = if (diver_testserver == FALSE) read_secret("db.diver_dsn") else  
    read_secret("db.diver_dsn_test"),  
  diver_testserver = FALSE,  
  diver_tablename = "data"  
)
```

Arguments

- preset** a preset to choose from [presets\(\)](#). Will be ignored if diver_cbase is set, even if it is set to NULL. Be sure to read [the documentation](#) on how to use presets, and to see in which order the YAML keys will be run.
- diver_cbase, diver_project, diver_dsn, diver_testserver**
 properties to set in [db_connect\(\)](#). The diver_cbase argument will be based on preset, but can also be set to blank NULL to manually select a cBase in a popup window.
- diver_tablename**
 name of the database table to download data from. This is hard-coded by DI and should normally never be changed.

db_connect*Connect to a Certe Database*

Description

Connect to an internal (yet remote) Certe database server using `DBI::dbConnect()`, or any other database.

Usage

```
db_connect(driver, ..., print = TRUE)
```

```
db_close(conn, ..., print = TRUE)
```

Arguments

driver	database driver to use, such as <code>odbc::odbc()</code> and <code>duckdb::duckdb()</code>
...	arguments passed on to <code>DBI::dbDisconnect()</code>
print	a logical to indicate whether info about the connection should be printed
conn	connection to close, such as the output of <code>db_connect()</code>

Examples

```
# create a local duckdb database
db <- db_connect(duckdb::duckdb(), "~/my_duck.db")
db

db |> db_write_table("my_iris_table", values = iris)
db |> db_list_tables()
db |> db_has_table("my_iris_table")

if (require(dplyr, warn.conflicts = FALSE)) {
  db |>
    tbl("my_iris_table") |>
    filter(Species == "setosa", Sepal.Width > 3) |>
    collect() |>
    as_tibble()
}

db |> db_drop_table("my_iris_table")
db |> db_list_tables()

db |> db_close()

# remove the database
unlink("~/my_duck.db")
```

get_diver_data

*Download Data from a Local or Remote Database***Description**

These functions can be used to download local or remote database data, e.g. Spectre data from DiveLine on a Diver server (from [Dimensional Insight](#)). The `get_diver_data()` function sets up an ODBC connection (using `db_connect()`), which requires their quite limited [DI-ODBC driver](#).

Usage

```
get_diver_data(
  date_range = this_year(),
  where = NULL,
  post_where = NULL,
  review_qry = interactive(),
  antibiogram_type = "sir",
  distinct = TRUE,
  auto_transform = TRUE,
  snake_case = TRUE,
  preset = read_secret("db.preset_default"),
  add_cols = NULL,
  date_column = NULL,
  diver_cbase = NULL,
  diver_project = read_secret("db.diver_project"),
  diver_dsn = if (diver_testserver == FALSE) read_secret("db.diver_dsn") else
    read_secret("db.diver_dsn_test"),
  diver_testserver = FALSE,
  diver_tablename = "data",
  info = interactive(),
  log_file = read_secret("db.query_log_sqlite"),
  limit = Inf,
  only_real_patients = TRUE,
  only_conducted_tests = TRUE,
  only_validated = FALSE,
  only_requested = FALSE,
  only_relevant_rows = TRUE,
  convert_numeric_where = TRUE,
  in_background = FALSE,
  ...
)

certedb_query(query, where = NULL, auto_transform = TRUE, info = interactive())

get_duckdb_data(
  date_range = this_year(),
  where = NULL,
```

```
    preset = NULL,
    review_qry = interactive(),
    duckdb_path = read_secret("db.duckdb_path"),
    duckdb_table = read_secret("db.duckdb_table"),
    auto_transform = TRUE,
    info = interactive()
)

certedb_getmmb(
  dates = NULL,
  where = NULL,
  select_preset = "mmb",
  preset = "mmb",
  select = NULL,
  add_cols = NULL,
  info = interactive(),
  first_isolates = FALSE,
  eucast_rules = "all",
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  mic = FALSE,
  disk = FALSE,
  zipcodes = FALSE,
  ziplength = 4,
  tat_hours = FALSE,
  log_file = read_secret("db.query_log_sqlite"),
  only_real_patients = TRUE,
  only_conducted_tests = TRUE,
  only_validated = FALSE,
  only_show_query = FALSE,
  review_where = interactive(),
  auto_transform = TRUE,
  query = NULL,
  ...
)

certedb_getmmb_tat(
  dates = NULL,
  where = NULL,
  add_cols = NULL,
  info = interactive(),
  only_real_patients = TRUE,
  only_conducted_tests = TRUE,
  only_validated = TRUE,
  only_show_query = FALSE,
  ...
)

get_glims10_data(
```

```

date_range = this_year(),
where = NULL,
review_qry = interactive(),
antibiogram_type = "sir",
distinct = TRUE,
auto_transform = TRUE,
diver_tablename = "glims10",
info = interactive(),
limit = Inf,
...
)

```

Arguments

date_range	date range, can be length 1 or 2 (or more to use the min/max) to filter on the column specified in the YAML file, see presets . Defaults to <code>certetoolbox::this_year()</code> . Use NULL to set no date filter. Can also be years, or functions such as <code>last_month()</code> . Date-time objects will be converted to dates, so using times as input is useless. It is supported to filter on a date-time column though.
where, post_where	arguments to filter data on, will be passed on to <code>filter()</code> . Do not use && or but only & or in filtering. The post_where will be run after all downloading and post-processing, but before the auto-transform.
review_qry	a logical to indicate whether the query must be reviewed first, defaults to TRUE in interactive mode and FALSE otherwise. This will always be FALSE in Quarto / R Markdown, since the output of <code>knitr::pandoc_to()</code> must be NULL.
antibiogram_type	antibiotic transformation mode. Leave blank to strip antibiotic results from the data, "sir" to keep SIR values, "mic" to keep MIC values or "disk" to keep disk diffusion values. Values will be cleaned with <code>as.sir()</code> , <code>as.mic()</code> or <code>as.disk()</code> .
distinct	logical to apply <code>dplyr::distinct()</code> to the resulting data set
auto_transform	logical to apply <code>certetoolbox::auto_transform()</code> to the resulting data set
snake_case	logical to convert column names to snake case , only when auto_transform = TRUE
preset	a preset to choose from <code>presets()</code> . Will be ignored if diver_cbase is set, even if it is set to NULL. Be sure to read the documentation on how to use presets, and to see in which order the YAML keys will be run.
add_cols	extra column to include in the selection. Can be named to set a new column name. Use <code>gl</code> or <code>di</code> to quickly pick from a list.
date_column	column name of data set to query. Normally this should be set in a preset, but this argument can be used to override that.
diver_cbase, diver_project, diver_dsn, diver_testserver	properties to set in <code>db_connect()</code> . The diver_cbase argument will be based on preset, but can also be set to blank NULL to manually select a cBase in a popup window.

diver_tablename	name of the database table to download data from. This is hard-coded by DI and should normally never be changed.
info	settings for old <code>certedb_getmmb()</code> function
log_file	path to a SQLite file to which the query will be logged, after collection and before any transformation. Use NULL or "" to prevent logging. The log can be read with <code>read_query_log()</code> .
limit	maximum number of rows to return.
only_real_patients,	<code>only_conducted_tests,</code>
only_requested,	<code>only_validated,</code>
only_relevant_rows	
	These can all be set to TRUE or FALSE, and if TRUE, run scripts in the folder <code>read_secrets("db.datafilters")</code> :
	• <code>only_real_patients.R</code> to include only real patients, i.e., remove test and Q&A samples • <code>only_conducted_tests.R</code> to include only tests that were not stopped • <code>only_validated.R</code> to include only validated tests • <code>only_requested.R</code> to include only requested tests • <code>only_relevant_rows.R</code> to include only relevant rows
convert_numeric_where	<code>logical</code> to convert numeric vectors in the WHERE to character vectors. For example, <code>where = PatientID %in% pat_numbers</code> would update the WHERE clause to make the numeric vector <code>pat_numbers</code> a character vector. Using <code>%in%</code> with numeric vectors in a Diver query has been known to return insufficient rows.
in_background	run data collection in the background using <code>callr::r_bg()</code> . Use <code>...\$get_result()</code> to retrieve results, or <code>...\$is_active()</code> to check whether the background process still runs.
...	not used anymore, previously settings for old <code>certetools::certedb_getmmb()</code> function
query	a <code>data.frame</code> to view the query of, or a <code>character</code> string to run as query in <code>certedb_getmmb()</code> (which will ignore all other arguments, except for <code>where</code> , <code>auto_transform</code> and <code>info</code>).
duckdb_path	path to the local DuckDB database
duckdb_table	name of the DuckDB database to retrieve data from
dates	date range, can be length 1 or 2 (or more to use the min/max) to filter on the column <code>Ontvangstdatum</code> . Defaults to <code>certetoolbox::this_year()</code> . Use NULL to set no date filter. Can also be years, or functions such as <code>last_month()</code> .
select_preset	settings for old <code>certedb_getmmb()</code> function
select	settings for old <code>certedb_getmmb()</code> function
first_isolates	settings for old <code>certedb_getmmb()</code> function
eucast_rules	settings for old <code>certedb_getmmb()</code> function

keep_synonyms	A logical to indicate if outdated, previously valid taxonomic names must be preserved and not be corrected to currently accepted names. Do note that the term "synonym" is in this case jargon from the field of microbial taxonomy - it is not in place to denote that e.g. "Streptococcus Group A" is a synonym of <i>S. pyogenes</i>. Though this is practically the case, taxonomically it is not as "Streptococcus Group A" is not even a valid taxonomic name. The default is FALSE, which will return a note if outdated taxonomic names were processed. The default can be set with the package option AMR_keep_synonyms, i.e. <code>options(AMR_keep_synonyms = TRUE)</code> or <code>options(AMR_keep_synonyms = FALSE)</code>.
mic	settings for old <code>certedb_getmmb()</code> function
disk	settings for old <code>certedb_getmmb()</code> function
zipcodes	settings for old <code>certedb_getmmb()</code> function
ziplength	settings for old <code>certedb_getmmb()</code> function
tat_hours	settings for old <code>certedb_getmmb()</code> function
only_show_query	settings for old <code>certedb_getmmb()</code> function
review_where	a logical to indicate whether the query must be reviewed first, defaults to TRUE in interactive mode and FALSE otherwise. This will always be FALSE in Quarto / R Markdown, since the output of <code>knitr::pandoc_to()</code> must be NULL.

Details

These functions return a 'certedb tibble' from Diver or the certemmb MySQL database, which prints information in the tibble header about the used source and current user.

Use [certedb_query\(\)](#) to retrieve the original query that was used to download the data.

Using `certedb_query("your qry here")` is identical to using `certedb_getmmb(query = "your qry here")`.

Examples

```
## Not run:

# peek-preview of a cBase:
get_diver_data(diver_cbase = "models/MedEpi/GLIMS10_Resistentties.cbase",
               date_range = NULL, limit = 10, review_qry = FALSE, auto_transform = FALSE)

# these two work identical:
get_diver_data(date_range = 2024, where = BepalingCode == "PXNCOV")
get_diver_data(2024, BepalingCode == "PXNCOV")

# use gl$ to pull a list with column names while you type
get_diver_data(2024, gl$BepalingCode == "PXNCOV")

# gl$ can also be used for adding columns, which can also be named
get_diver_data(2024, gl$BepalingCode == "PXNCOV", add_cols = gl$AfnameJaar)
get_diver_data(2024, gl$BepalingCode == "PXNCOV", add_cols = c(jaar = gl$AfnameJaar))
```

```

# for the `where`, use `&` or `|`:
get_diver_data(last_month(),
  gl$BepalingCode == "PXNCOV" & gl$Zorglijn == "2e lijn")
get_diver_data(c(2020:2024),
  where = gl$BepalingCode == "PXNCOV" | gl$Zorglijn == "2e lijn")

# use %like%, %unlike%, %like_case% or %unlike_case% for regular expressions
get_diver_data(2024, where = gl$MateriaalNaam %like% "Bloed")
get_diver_data(2024, where = gl$MateriaalNaam %unlike% "Bloed")
get_diver_data(2024, where = gl$MateriaalNaam %like_case% "bloed")
get_diver_data(2024, where = gl$MateriaalNaam %unlike_case% "Bloed")

get_diver_data(2024,
  where = gl$BepalingNaam %like% "Noro" &
    gl$PatientLeeftijd >= 75)

# R objects will be converted
materialen <- c("A", "B", "C")
get_diver_data(2024, where = gl$MateriaalNaam %in% materialen)
leeftijden <- 65:85
get_diver_data(2024, where = gl$PatientLeeftijd %in% leeftijden)

# USING DIVER INTEGRATOR LANGUAGE -----

# See the website for an overview of allowed functions:
# https://www.dimins.com/online-help/workbench\_help/Content/ODBC/di-odbc-sql-reference.html

# Use Diver Integrator functions within EVAL():
get_diver_data(2024, where = EVAL('regexp(value("MateriaalCode"), "^B")'))

get_diver_data(
  2024,
  where = EVAL('rolling(12, value("OntvangstDatum"), date("2024/11/27"))')
)

## End(Not run)

# USING DUCKDB DATABASE -----

# create a local duckdb database and write a table
db <- db_connect(duckdb::duckdb(), "~/my_duck.db")
db |> db_write_table("iris", values = iris)
db |> db_close()

df <- get_duckdb_data(date_range = NULL,
  where = Species == "setosa" & Sepal.Width > 3,
  # not needed in production environment:
  duckdb_path = "~/my_duck.db",
  duckdb_table = "iris",

```

```
        review_qry = FALSE,
        info = TRUE)

df
certedb_query(df)

# remove the database
unlink("~/my_duck.db")
```

mysql_init	<i>Initialise MySQL Connection for Connections Pane</i>
------------	---

Description

This sets up a MySQL connection to be viewable in the Connections pane.

Usage

```
mysql_init()
```

presets	<i>Available Presets for get_diver_data()</i>
---------	---

Description

Work with presets for [get_diver_data\(\)](#). This automates selecting, filtering, and joining cBases.

Usage

```
presets()

get_preset(preset)
```

Arguments

preset	name of the preset
--------	--------------------

Details

The function [presets\(\)](#) returns a data.frame with available presets, as defined in the secrets YAML file under db.presets.

The function [get_preset\(\)](#) will return all the details of a preset as a [list](#).

Required YAML Format

This YAML information should be put into the YAML file that is read using `read_secret()`. Afterwards, the name of the preset can be used as `get_diver_data(preset = "...")`.

The most basic YAML form:

```
name_new_preset:
  cbase: "location/to/name.cbase"
  date_col: "ColumnNameDate"
```

The most extensive YAML form:

```
name_new_preset:
  cbase: "location/to/name.cbase"
  date_col: "ColumnNameDate"
  filter: ColumnName1 %in% c("Filter1", "Filter2") & ColumnName2 %in% c("Filter3", "Filter4")
  select: ColumnName1, ColumnName2, ColumnNameReceiptDate, new_name = OldName
  join:
    cbase: "location/to/another.cbase"
    by: ColumnName1, ColumnName2
    type: "left"
    filter: ColumnName1 == "abc"
  select: ColumnName1, col_name_2 = ColumnName2, ColumnName3, everything(), !starts_with("abc")
  wide_names_from: ColumnName3
  join2:
    cbase: "location/to/yet_another.cbase"
    by: ColumnName1, ColumnName2
    type: "left"
    select: ColumnName1, ColumnName2, ColumnName3
    wide_names_from: ColumnName3
    wide_name_trans: gsub("_", "..", .x)
  post-processing:
    - "file 1.R"
    - "file 2.R"
```

For all presets, `cbase` and `date_col` are required.

Order of running:

The YAML keys run in this order:

1. Download cBase and filter (applied in WHERE statement)
2. Select
3. Join(s)
4. Post-processing

After this, the arguments in `get_diver_data()` will run:

1. Post-WHERE if `post_where` is set, using `filter()`
2. Distinct if `distinct = TRUE`, using `distinct()`
3. Auto-transform if `autotransform = TRUE`, using `auto_transform()`

cBase (cbase):

This cBase must be a filepath and must exist on the Diver server. For joins, this can also be another type of file, see *Joins*

Select (select):

Input for select will be passed on to `select()`, meaning that column names can be used, but also tidyselect functions such as `everything()`.

Filters (filter):

Input for filter will be passed on to `filter()`.

Joins (join):

For joins, you must set at least cbase, by, and type ("left", "right", "inner", etc., [see here](#)).

Other files than a cBase in the field cbase will be imported using `certetoolbox::import()`, such as an Excel or CSV file. This can be any file on any local or remote location (even live internet files). For example:

join:

```
cbase: "location/to/excel_file.xlsx"
by: ColumnName1
type: "left"
```

join:

```
cbase: "https://github.com/certe-medical-epidemiology/certegis/blob/main/data/geo_gemeenten.rda"
by: ColumnName1
type: "left"
```

An unlimited number of joins can be used, but all so-called 'keys' **must be unique** and start with join, e.g. joinA / joinB or join / join2 / join3.

If wide_names_from is set, the dataset is first transformed to long format using the columns specified in by, and then reshaped to wide format with values in wide_names_from.

Use wide_name_trans to transform the values in wide_names_from before the reshaping to wide format is applied. Use .x for the column values. As this will be applied before the data is transformed to a wide format, it allows to refine the values in wide_names_from using e.g., `case_when()`.

Post-processing (post-processing):

Any data transformation can be done after the data have been downloaded and processed according to all previous steps. Use x to indicate the data set.

name_new_preset:

```
cbase: "location/to/name.cbase"
date_col: "ColumnNameDate"
post-processing: |
  x |>
    mutate(Column1 = case_when(Column2 = "A" ~ 1,
                               Column3 = "B" ~ 2,
                               TRUE ~ 3))
```

As shown, in YAML the `|` character can be used to start a multi-line statement.

The post-processing field can also be one or more R file paths, which will be read using `readLines()`. Each of these chunks is valid:

```
name_new_preset:
  cbase: "location/to/name.cbase"
  date_col: "ColumnNameDate"
  post-processing: "location/to/file.R"

name_new_preset:
  cbase: "location/to/name.cbase"
  date_col: "ColumnNameDate"
  post-processing:
    - "location/to/file 1.R"
    - "location/to/file 2.R"

name_new_preset:
  cbase: "location/to/name.cbase"
  date_col: "ColumnNameDate"
  post-processing1: "location/to/file 1.R"
  post-processing2: "location/to/file 2.R"
```

Note that all post-processing steps will run directly after the querying the data and thus **before auto-transformation** if `autotransform = TRUE` in `get_diver_data()`.

read_query_log	<i>Read and Write the Query Log</i>
----------------	-------------------------------------

Description

The `read_query_log()` function reads the query log from a SQLite database. The `write_query_log()` function writes a query log entry. Join queries from presets are stored in a separate table and linked to their parent query; use `read_query_log()` to retrieve them together.

Usage

```
read_query_log(
  log_file = read_secret("db.query_log_sqlite"),
  date_range = this_year(),
  with_joins = TRUE,
  joins_only = FALSE
)

write_query_log(
  log_file,
  user,
  source_type,
```

```

    source_dsn,
    source_project,
    source_cbase = NA_character_,
    query,
    rows,
    columns,
    duration_secs = NA_real_,
    print = TRUE
)

```

Arguments

log_file	path to the SQLite query log file. Defaults to the path set in <code>read_secret("db.query_log_sqlite")</code> .
date_range	date range to filter log entries on, can be length 1 or 2 (or more to use the min/max). Defaults to <code>certetoolbox::this_year()</code> . Use NULL to return all entries. Can also be years, or functions such as <code>last_month()</code> .
with_joins	if TRUE, add the joins to the table
joins_only	if TRUE, return only the joins table instead of the main query log
user	user name that ran the query
source_type	type of source, e.g. "Diver" or "MySQL"
source_dsn	the DSN used to connect
source_project	the project used to connect
source_cbase	the cBase used to connect (if applicable)
query	the SQL query that was run
rows	number of rows returned
columns	number of columns returned
duration_secs	duration of the query in seconds
print	a logical to indicate whether info should be printed

Examples

```

## Not run:

read_query_log() # this year (default)
read_query_log(date_range = 2024) # all of 2024
read_query_log(date_range = last_month()) # last month
read_query_log(date_range = c(2023, 2025)) # range of years
read_query_log(date_range = NULL) # everything

## End(Not run)

```

Index

AMR_keep_synonyms, 8
as.disk(), 6
as.mic(), 6
as.sir(), 6
auto_transform(), 11

callr::r_bg(), 7
case_when(), 12
cbase_explorer, 2
certedb_getmmb (get_diver_data), 4
certedb_getmmb(), 7
certedb_getmmb_tat (get_diver_data), 4
certedb_query (get_diver_data), 4
certedb_query(), 8
certetoolbox::auto_transform(), 6
certetoolbox::import(), 12
certetoolbox::this_year(), 6, 7, 14
character, 7

data.frame, 7
db_close (db_connect), 3
db_connect, 3
db_connect(), 2–4, 6
DBI::dbConnect(), 3
DBI::dbDisconnect(), 3
di, 6
distinct(), 11
dplyr::distinct(), 6
duckdb::duckdb(), 3

everything(), 12

filter(), 6, 11, 12

get_diver_data, 4
get_diver_data(), 4, 10, 11, 13
get_duckdb_data (get_diver_data), 4
get_glms10_data (get_diver_data), 4
get_preset (presets), 10
get_preset(), 10
gl, 6

knitr::pandoc_to(), 6, 8

last_month(), 6, 7, 14
list, 10
logical, 6–8, 14

mysql_init, 10

odbc::odbc(), 3

presets, 6, 10
presets(), 2, 6, 10

read_query_log, 13
read_query_log(), 7, 13
read_secret(), 11
readLines(), 13

see here, 12
select(), 12

the documentation, 2, 6

write_query_log (read_query_log), 13
write_query_log(), 13