Package 'ClickHouseHTTP'

November 12, 2026

Type Package

Title A Simple HTTP Database Interface to 'ClickHouse'

Version 1.1.3

Description 'ClickHouse' (<https://clickhouse.com/>)
is an open-source, high performance columnar
OLAP (online analytical processing of queries) database management system
for real-time analytics using SQL. This 'DBI' backend
relies on the 'ClickHouse' HTTP interface and support HTTPS protocol.

URL https://patzaw.github.io/ClickHouseHTTP/,
https://github.com/patzaw/ClickHouseHTTP/

BugReports https://github.com/patzaw/ClickHouseHTTP/issues

Depends R (>= 4.1)

Imports methods, DBI (>= 0.3.0), httr2, jsonlite, arrow, data.table,
stats

Suggests knitr, rmarkdown, dplyr, dbplyr, stringi, stringr, testthat
(>= 3.0.0), withr

License GPL-3

Encoding UTF-8

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Patrice Godard [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-6257-9730>),
Eusebiu Marcu [ctb]

Maintainer Patrice Godard <patrice.godard@gmail.com>

Repository CRAN

Date/Publication 2026-11-12 13:00:15 UTC

Contents

ClickHouseHTTP 2
ClickHouseHTTPConnection-class 2
ClickHouseHTTPDriver-class 10
ClickHouseHTTPResult-class 13

Index **14**

`ClickHouseHTTP` *Create a ClickHouseHTTP DBI driver*

Description

Create a ClickHouseHTTP DBI driver

Usage

```
ClickHouseHTTP()
```

Value

A ClickHouseHTTPDriver

See Also

ClickHouseHTTPDriver

`ClickHouseHTTPConnection-class`
ClickHouseHTTPConnection class.

Description

ClickHouseHTTPConnection class.

Send SQL query to ClickHouse

Information about the ClickHouse database

List tables in ClickHouse

Does a table exist?

Read database tables as data frames

List field names of a table

Remove a table from the database

Create a table in ClickHouse

Insert rows into a table

Write a table in ClickHouse

Usage

```
## S4 method for signature 'ClickHouseHTTPConnection,character'
dbSendQuery(
  conn,
  statement,
  format = c("Arrow", "TabSeparatedWithNamesAndTypes"),
  file = NA,
  ...
)

## S4 method for signature 'ClickHouseHTTPConnection'
dbGetInfo(dbObj, ...)

## S4 method for signature 'ClickHouseHTTPConnection'
dbListTables(conn, database = NA, ...)

## S4 method for signature 'ClickHouseHTTPConnection,character'
dbExistsTable(conn, name, database = NA, ...)

## S4 method for signature 'ClickHouseHTTPConnection,character'
dbReadTable(conn, name, database = NA, ...)

## S4 method for signature 'ClickHouseHTTPConnection,character'
dbListFields(conn, name, database = NA, ...)

## S4 method for signature 'ClickHouseHTTPConnection,ANY'
dbRemoveTable(conn, name, database = NA, ...)

## S4 method for signature 'ClickHouseHTTPConnection'
dbCreateTable(
  conn,
  name,
  fields,
  database = NA,
  engine = "TinyLog",
  overwrite = FALSE,
  ...,
  row.names = NULL,
  temporary = FALSE
)

## S4 method for signature 'ClickHouseHTTPConnection'
dbAppendTable(conn, name, value, database = NA, ..., row.names = NULL)

## S4 method for signature 'ClickHouseHTTPConnection,ANY'
dbWriteTable(
  conn,
  name,
```

```
  value,
  database = NA,
  overwrite = FALSE,
  append = FALSE,
  engine = "TinyLog",
  ...
)
```

Arguments

conn	a ClickHouseHTTPConnection object created with dbConnect()
statement	the SQL query statement
format	the format used by ClickHouse to send the results. Two formats are supported: "Arrow" (default) and "TabSeparatedWithNamesAndTypes"
file	a path to a file to send along the query (default: NA)
...	Other parameters passed on to methods
dbObj	a ClickHouseHTTPConnection object
database	the database to consider. If NA (default), the default database or the one in use in the session (if a session is defined).
name	the name of the table to create
fields	a character vector with the name of the fields and their ClickHouse type (e.g. `c("text_col String", "num_col Nullable(Float64)", "nul_col Array(Int32)")`)
engine	the ClickHouse table engine as described in ClickHouse documentation. Examples: • "TinyLog" (default) • "MergeTree() ORDER BY (expr)" (expr generally correspond to fields separated by ",")
overwrite	if TRUE and if a table with the same name exists, then it is deleted before creating the new one (default: FALSE)
row.names	unsupported parameter (add for compatibility reason)
temporary	unsupported parameter (add for compatibility reason)
value	a data.frame
append	if TRUE, the values are added to the database table if it exists (default: FALSE).

Details

Both format have their pros and cons:

- **Arrow** (default):
  - fast for long tables but slow for wide tables
  - fast with Array columns
  - Date and DateTime columns are returned as UInt16 and UInt32 respectively: by default, ClickHouseHTTP interpret them as Date and POSIXct columns but cannot make the difference with actual UInt16 and UInt32

- **TabSeparatedWithNamesAndTypes**:
  - in general faster than Arrow
  - fast for wide tables but slow for long tables
  - slow with Array columns
  - Special characters are not well interpreted. In such cases, the function below can be useful but can also take time.

```
.sp_ch_recov <- function(x){
   stringi::stri_replace_all_regex(
      x,
      c(
         "\\n", "\\t",  "\\r", "\\b",
         "\\a", "\\f", "\\'",  "\\\\"
      ),
      c("\n", "\t", "\r", "\b", "\a", "\f", "'", "\\"),
      vectorize_all=FALSE
   )
}
```

Value

A ClickHouseHTTPResult object

A list with the following elements:

- name: "ClickHouseHTTPConnection"
- db.version: the version of ClickHouse
- uptime: ClickHouse uptime
- dbname: the default database
- username: user name
- host: ClickHouse host
- port: ClickHouse port
- https: Is the connection using HTTPS protocol instead of HTTP

A vector of character with table names.

A logical.

A data.frame.

A vector of character with column names.

`invisible(TRUE)`

dbCreateTable() returns TRUE, invisibly.

The number of rows written, invisibly.

TRUE; called for side effects

Note

Time zones: `toTimeZone()` calls in SQL queries are ignored when `format = "Arrow"` and are not correctly interpreted when `format = "TabSeparatedWithNamesAndTypes"`. It is preferable to set the time zone through the `settings` parameter of `DBI::dbConnect()`, for example `settings = list(session_timezone = "CET")`.

See Also

ClickHouseHTTPResult

Examples

```
if (FALSE) {
  ## Connection ----

  library(DBI)
  ### HTTP connection ----

  con <- dbConnect(
    ClickHouseHTTP::ClickHouseHTTP(),
    host = "localhost",
    port = 8123
  )

  ### HTTPS connection (without ssl peer verification) ----

  con <- dbConnect(
    ClickHouseHTTP::ClickHouseHTTP(),
    host = "localhost",
    port = 8443,
    https = TRUE,
    ssl_verifypeer = FALSE
  )

  ## Write a table in the database ----

  library(dplyr)
  data("mtcars")
  mtcars <- as_tibble(mtcars, rownames = "car")
  dbWriteTable(con, "mtcars", mtcars)

  ## Query the database ----

  carsFromDB <- dbReadTable(con, "mtcars")
  dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

  ## By default, ClickHouseHTTP relies on the
  ## Apache Arrow format provided by ClickHouse.
  ## The `format` argument of the `dbGetQuery()` function can be used to
  ## rely on the *TabSeparatedWithNamesAndTypes* format.
  selCars <- dbGetQuery(
    con,
```

```
  "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format = "TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "types")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames = "province")
swiss <- mutate(swiss, "pr letters" = strsplit(province, ""))
dbWriteTable(
  con,
  "swiss",
  swiss,
  engine = "MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))
}
if (FALSE) {
  ## Connection ----

  library(DBI)
  ### HTTP connection ----

  con <- dbConnect(
    ClickHouseHTTP::ClickHouseHTTP(),
    host = "localhost",
    port = 8123
  )

  ### HTTPS connection (without ssl peer verification) ----

  con <- dbConnect(
    ClickHouseHTTP::ClickHouseHTTP(),
    host = "localhost",
    port = 8443,
    https = TRUE,
    ssl_verifypeer = FALSE
  )
```

```
  ## Write a table in the database ----

  library(dplyr)
  data("mtcars")
  mtcars <- as_tibble(mtcars, rownames = "car")
  dbWriteTable(con, "mtcars", mtcars)

  ## Query the database ----

  carsFromDB <- dbReadTable(con, "mtcars")
  dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

  ## By default, ClickHouseHTTP relies on the
  ## Apache Arrow format provided by ClickHouse.
  ## The `format` argument of the `dbGetQuery()` function can be used to
  ## rely on the *TabSeparatedWithNamesAndTypes* format.
  selCars <- dbGetQuery(
    con,
    "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
    format = "TabSeparatedWithNamesAndTypes"
  )
  ## Identifying the original ClickHouse data types
  attr(selCars, "types")

  ## Using alternative databases stored in ClickHouse ----

  dbSendQuery(con, "CREATE DATABASE swiss")
  dbSendQuery(con, "USE swiss")

  ## The chosen database is used until the session expires.
  ## It can also be chosen when connecting using the `dbname` argument of
  ## the `dbConnect()` function.

  ## The example below shows that spaces in column names are supported.
  ## It also shows the support of R `list` using the *Array* ClickHouse type.
  data("swiss")
  swiss <- as_tibble(swiss, rownames = "province")
  swiss <- mutate(swiss, "pr letters" = strsplit(province, ""))
  dbWriteTable(
    con,
    "swiss",
    swiss,
    engine = "MergeTree() ORDER BY (Fertility, province)"
  )
  swissFromDB <- dbReadTable(con, "swiss")

  ## A table from another database can also be accessed as following:
  dbReadTable(con, SQL("default.mtcars"))
}
if (FALSE) {
  ## Connection ----

  library(DBI)
```

```
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(),
  host = "localhost",
  port = 8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(),
  host = "localhost",
  port = 8443,
  https = TRUE,
  ssl_verifypeer = FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames = "car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con,
  "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format = "TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "types")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
```

```
  data("swiss")
  swiss <- as_tibble(swiss, rownames = "province")
  swiss <- mutate(swiss, "pr letters" = strsplit(province, ""))
  dbWriteTable(
    con,
    "swiss",
    swiss,
    engine = "MergeTree() ORDER BY (Fertility, province)"
  )
  swissFromDB <- dbReadTable(con, "swiss")

  ## A table from another database can also be accessed as following:
  dbReadTable(con, SQL("default.mtcars"))
}
```

ClickHouseHTTPDriver-class *Driver for the ClickHouse database using HTTP(S) interface*

Description

Driver for the ClickHouse database using HTTP(S) interface

Connect to a ClickHouse database using the ClickHouseHTTP DBI

Usage

```
## S4 method for signature 'ClickHouseHTTPDriver'
dbConnect(
  drv,
  host = "localhost",
  port = 8123L,
  dbname = "default",
  user = "default",
  password = "",
  https = FALSE,
  ssl_verifypeer = TRUE,
  host_path = NA,
  use_session = FALSE,
  session_timeout = 3600L,
  convert_uint = TRUE,
  extended_headers = list(),
  reset_handle = FALSE,
  settings = list(),
  ...
)
```

Arguments

drv	A driver object created by ClickHouseHTTP()
host	name of the database host (default: "localhost")
port	port on which the database is listening (default: 8123L)
dbname	name of the default database (default: "default")
user	user name (default: "default")
password	user password (default: "")
https	a logical to use the HTTPS protocol (default: FALSE)
ssl_verifypeer	a logical to verify SSL certificate when using HTTPS (default: TRUE)
host_path	a path to use on host (e.g. "ClickHouse/"): it allows to connect on a server behind a reverse proxy for example
use_session	a logical indicating if a session should be created and use in ClickHouse (default: FALSE)
session_timeout	timeout in seconds (default: 3600L seconds)
convert_uint	a logical: if TRUE (default), UInt ClickHouse data types are converted in the following R classes: • UInt8 : logical • UInt16: Date (when using Arrow format in dbSendQuery,ClickHouseHTTPConnection,character-method) • UInt32: POSIXct (when using Arrow format in dbSendQuery,ClickHouseHTTPConnection,characte method)
extended_headers	a named list with other HTTP headers (for example: extended_headers=list("X-Authorization"="Be <token>") can be used for OAuth access delegation)
reset_handle	a logical indicating whether to force a fresh TCP connection for each request (httr2 fresh_connect curl option). If TRUE, connections are never reused from the pool, which can help in long-running sessions where stale connections cause errors (default: FALSE).
settings	list of Clickhouse settings
...	Other parameters passed on to methods

Value

A ClickHouseHTTPConnection

See Also

ClickHouseHTTPConnection

Examples

```
if (FALSE) {
  ## Connection ----

  library(DBI)
  ### HTTP connection ----

  con <- dbConnect(
    ClickHouseHTTP::ClickHouseHTTP(),
    host = "localhost",
    port = 8123
  )

  ### HTTPS connection (without ssl peer verification) ----

  con <- dbConnect(
    ClickHouseHTTP::ClickHouseHTTP(),
    host = "localhost",
    port = 8443,
    https = TRUE,
    ssl_verifypeer = FALSE
  )

  ## Write a table in the database ----

  library(dplyr)
  data("mtcars")
  mtcars <- as_tibble(mtcars, rownames = "car")
  dbWriteTable(con, "mtcars", mtcars)

  ## Query the database ----

  carsFromDB <- dbReadTable(con, "mtcars")
  dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

  ## By default, ClickHouseHTTP relies on the
  ## Apache Arrow format provided by ClickHouse.
  ## The `format` argument of the `dbGetQuery()` function can be used to
  ## rely on the *TabSeparatedWithNamesAndTypes* format.
  selCars <- dbGetQuery(
    con,
    "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
    format = "TabSeparatedWithNamesAndTypes"
  )
  ## Identifying the original ClickHouse data types
  attr(selCars, "types")

  ## Using alternative databases stored in ClickHouse ----

  dbSendQuery(con, "CREATE DATABASE swiss")
  dbSendQuery(con, "USE swiss")
```

```
  ## The chosen database is used until the session expires.
  ## It can also be chosen when connecting using the `dbname` argument of
  ## the `dbConnect()` function.

  ## The example below shows that spaces in column names are supported.
  ## It also shows the support of R `list` using the *Array* ClickHouse type.
  data("swiss")
  swiss <- as_tibble(swiss, rownames = "province")
  swiss <- mutate(swiss, "pr letters" = strsplit(province, ""))
  dbWriteTable(
    con,
    "swiss",
    swiss,
    engine = "MergeTree() ORDER BY (Fertility, province)"
  )
  swissFromDB <- dbReadTable(con, "swiss")

  ## A table from another database can also be accessed as following:
  dbReadTable(con, SQL("default.mtcars"))
}
```

ClickHouseHTTPResult-class *ClickHouseHTTPResult class.*

Description

ClickHouseHTTPResult class.

Index

ClickHouseHTTP, 2
ClickHouseHTTP(), *11*
ClickHouseHTTPConnection, *11*
ClickHouseHTTPConnection-class, 2
ClickHouseHTTPDriver, *2*
ClickHouseHTTPDriver-class, 10
ClickHouseHTTPResult, *6*
ClickHouseHTTPResult-class, 13

dbAppendTable,ClickHouseHTTPConnection-method
(ClickHouseHTTPConnection-class), 2
dbConnect(), *4*
dbConnect,ClickHouseHTTPDriver-method
(ClickHouseHTTPDriver-class), 10
dbCreateTable,ClickHouseHTTPConnection-method
(ClickHouseHTTPConnection-class), 2
dbExistsTable,ClickHouseHTTPConnection,character-method
(ClickHouseHTTPConnection-class), 2
dbGetInfo,ClickHouseHTTPConnection-method
(ClickHouseHTTPConnection-class), 2
DBI::dbConnect(), *6*
dbListFields,ClickHouseHTTPConnection,character-method
(ClickHouseHTTPConnection-class), 2
dbListTables,ClickHouseHTTPConnection-method
(ClickHouseHTTPConnection-class), 2
dbReadTable,ClickHouseHTTPConnection,character-method
(ClickHouseHTTPConnection-class), 2
dbRemoveTable,ClickHouseHTTPConnection,ANY-method
(ClickHouseHTTPConnection-class), 2
dbSendQuery,ClickHouseHTTPConnection,character-method,
11
dbSendQuery,ClickHouseHTTPConnection,character-method
(ClickHouseHTTPConnection-class), 2
dbWriteTable,ClickHouseHTTPConnection,ANY-method
(ClickHouseHTTPConnection-class), 2