Package: RTrino (via r-universe)

October 15, 2026

Title 'DBI' Backend for 'Trino'

Version 0.1.0

Description A 'DBI' compliant backend for 'Trino' <https://trino.io>, the distributed SQL query engine (successor of 'PrestoSQL'). Connects to a 'Trino' cluster over its HTTP REST API, executes queries with automatic pagination, maps 'Trino' types to R types, and integrates with 'dplyr' through 'dbplyr' SQL dialects. Supports basic, JSON Web Token (JWT) bearer and OAuth2 client credentials authentication, as well as custom certificate authorities.

License BSD_3_clause + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports bit64, DBI (>= 1.2.0), httr2 (>= 1.0.0), jsonlite, methods, rlang, tibble, utils

Suggests dbplyr (>= 2.6.0), dplyr, knitr, rmarkdown, testthat (>= 3.0.0), webfakes, withr

URL https://angelcasasbl.github.io/RTrino/, https://github.com/AngelCasasbl/RTrino

BugReports https://github.com/AngelCasasbl/RTrino/issues

VignetteBuilder knitr

Config/testthat/edition 3

Collate 'RTrino-package.R' 'TrinoDriver.R' 'TrinoConnection.R' 'TrinoResult.R' 'utils.R' 'request_headers.R' 'auth.R' 'ssl.R' 'dbConnect.R' 'dbDisconnect.R' 'dbIsValid.R' 'type_map.R' 'paginate.R' 'dbSendQuery.R' 'dbFetch.R' 'dbGetQuery.R' 'dbHasCompleted.R' 'dbClearResult.R' 'table_name.R' 'dbListTables.R' 'dbExistsTable.R' 'dbListFields.R' 'dbGetInfo.R' 'dbQuote.R' 'dbDataType.R' 'dbRemoveTable.R' 'dbRenameTable.R' 'dbCreateTable.R' 'dbAppendTable.R'

'dbWriteTable.R' 'dbBegin.R' 'dbplyr-dialect.R' 'dbplyr-db.R' 'dbplyr-simulate.R' 'zzz.R'

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Luis A Casas Ballestas [aut, cre, cph]

Maintainer Luis A Casas Ballestas <luisangelcasasbl@gmail.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-15 10:00:02 UTC

RemoteUrl https://github.com/cran/RTrino

RemoteRef HEAD

RemoteSha 3e3deb835484d6722ec37017c6fe93e249e08021

Contents

dbAppendTable,TrinoConnection-method . . . 3
dbBegin,TrinoConnection-method . . . 4
dbBind,TrinoResult-method . . . 5
dbClearResult,TrinoResult-method . . . 5
dbConnect,TrinoDriver-method . . . 6
dbCreateTable,TrinoConnection-method . . . 8
dbDataType,TrinoConnection-method . . . 9
dbDisconnect,TrinoConnection-method . . . 10
dbExecute,TrinoConnection,character-method . . . 10
dbExistsTable,TrinoConnection,character-method . . . 11
dbFetch,TrinoResult-method . . . 12
dbGetInfo,TrinoConnection-method . . . 13
dbGetQuery,TrinoConnection,character-method . . . 13
dbHasCompleted,TrinoResult-method . . . 14
dbIsValid,TrinoConnection-method . . . 15
dbListFields,TrinoConnection,character-method . . . 15
dbListTables,TrinoConnection-method . . . 16
dbQuoteIdentifier,TrinoConnection,character-method . . . 17
dbQuoteLiteral,TrinoConnection-method . . . 18
dbRemoveTable,TrinoConnection,character-method . . . 19
dbRenameTable . . . 20
dbSendQuery,TrinoConnection,character-method . . . 20
dbSendStatement,TrinoConnection,character-method . . . 21
dbWriteTable,TrinoConnection,ANY-method . . . 22
Trino . . . 23
trino_auth . . . 24
trino_ssl . . . 25
trino_type_to_r . . . 26

Index **27**

dbAppendTable,TrinoConnection-method

Insert rows into an existing table

Description

Builds one `INSERT INTO ... VALUES (...)` statement per batch of rows, with values quoted as literals by `DBI::sqlAppendTable()` (which calls `dbQuoteIdentifier()` and `dbQuoteLiteral()` for Trino) — there is no parameter binding to fall back on.

Usage

```
## S4 method for signature 'TrinoConnection'
dbAppendTable(conn, name, value, ..., row.names = NULL)
```

Arguments

`conn`	A TrinoConnection object.
`name`	Table name, in any form `dbExistsTable()` accepts.
`value`	A data frame with the rows to insert.
`...`	Takes `chunk_size`: how many rows go into a single `INSERT` statement, 1000 by default. Every row is written into the statement as a literal, so a data frame much larger than this is split into several statements rather than one that may exceed the coordinator's query length limit.
`row.names`	Unused: RTrino never writes row names as a column.

Value

The number of rows inserted.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "memory", schema = "default")
DBI::dbCreateTable(con, "sales", c(id = "integer", label = "varchar"))
DBI::dbAppendTable(con, "sales", data.frame(id = 1:3, label = letters[1:3]))
DBI::dbGetQuery(con, "SELECT * FROM sales ORDER BY id")
DBI::dbRemoveTable(con, "sales")
DBI::dbDisconnect(con)
```

dbBegin,TrinoConnection-method

Start, commit and roll back a transaction

Description

Every statement runs inside Trino's engine-level transaction mechanism: `dbBegin()` sends `START TRANSACTION` and keeps the transaction id Trino answers with, which is then carried on every later request until `dbCommit()` or `dbRollback()` sends `COMMIT` or `ROLLBACK` and clears it.

Usage

```
## S4 method for signature 'TrinoConnection'
dbBegin(conn, ...)

## S4 method for signature 'TrinoConnection'
dbCommit(conn, ...)

## S4 method for signature 'TrinoConnection'
dbRollback(conn, ...)
```

Arguments

`conn`	A TrinoConnection object.
`...`	Unused, for compatibility with the generic.

Details

Whether a `ROLLBACK` actually undoes a write depends on the connector behind the active catalog, not on RTrino: Iceberg and Delta Lake tables support it, most others (including Hive, strictly) only partially, and Trino's own synthetic catalogs (`memory`, `tpch`, ...) not at all. `dbBegin()` warns when the active catalog is one of the latter.

Value

`TRUE`, invisibly.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")

# tpch holds no real storage, so dbBegin() warns that a ROLLBACK would
# undo nothing. On an Iceberg or Delta Lake catalog it would not.
DBI::dbBegin(con)
DBI::dbGetQuery(con, "SELECT count(*) AS n FROM nation")
DBI::dbCommit(con)
```

```
DBI::dbDisconnect(con)
```

---

dbBind,TrinoResult-method

Parameters are not supported

Description

Trino's client protocol has no parameter binding of its own, so rather than ignoring the values, `dbBind()` fails.

Usage

```
## S4 method for signature 'TrinoResult'
dbBind(res, params, ...)
```

Arguments

`res`	A TrinoResult object.
`params`	Unused.
`...`	Unused.

Value

Never returns.

---

dbClearResult,TrinoResult-method

Release a Trino result

Description

Frees the buffered rows and marks the cursor cleared. If the query is still running on the server, a `DELETE` is sent to its `nextUri` so the coordinator stops working on it instead of letting it run to completion for nobody.

Usage

```
## S4 method for signature 'TrinoResult'
dbClearResult(res, ...)
```

Arguments

`res`	A TrinoResult object.
`...`	Unused, for compatibility with the generic.

Value

TRUE, invisibly.

---

dbConnect,TrinoDriver-method

Connect to a Trino cluster

Description

Creates a TrinoConnection. Trino's REST API is stateless, so no session is opened on the server; the connection object holds the coordinator address and the per-request settings. To fail early on a wrong address or bad credentials, dbConnect() probes GET /v1/info before returning.

Usage

```
## S4 method for signature 'TrinoDriver'
dbConnect(
  drv,
  host = "http://localhost",
  port = 8080L,
  user = trino_default_user(),
  catalog = NULL,
  schema = NULL,
  source = "RTrino",
  session.timezone = "UTC",
  bigint = c("integer64", "numeric", "character"),
  auth = NULL,
  ssl_options = trino_ssl(),
  extra.headers = list(),
  timeout = 60,
  query_max_run_time = NULL,
  allow_http_auth = FALSE,
  ...
)
```

Arguments

drv	A TrinoDriver object, from Trino().
host	Coordinator URL. A missing scheme defaults to http://; a trailing slash is dropped. It may include the port ("https://trino.example.com:8443"), in which case port need not be given.
port	Coordinator port, a whole number between 1 and 65535.
user	Trino user name, sent as X-Trino-User.
catalog	Trino catalog. Required.
schema	Trino schema. Required.

`source` Value of the `X-Trino-Source` header; shows up in the coordinator's query history.

`session.timezone` Session time zone, used when reading `TIMESTAMP` columns.

`bigint` How to return `BIGINT` columns: `"integer64"` (the default, via `bit64`, exact except for the lowest value, -9223372036854775808, which bit64 reserves for `NA`), `"numeric"` (a double, exact only up to 2^53) or `"character"` (the exact digits).

`auth` An authentication closure from `trino_auth_basic()`, `trino_auth_jwt()` or `trino_auth_oauth2()`, or `NULL` for a cluster with no authentication.

`ssl_options` TLS options from `trino_ssl()`.

`extra.headers` Named list of extra HTTP headers added to every request, each a single string. Useful for Trino session properties set through `X-Trino-Session`.

`timeout` Seconds allowed for each HTTP request to the coordinator, not for the whole query; `Inf` for no limit. Connecting gives up after 10 seconds, or `timeout` if shorter.

`query_max_run_time` Longest a query may run on the cluster before Trino stops it, as a Trino duration such as `"30m"` or `"2h"`, or a number of seconds. `NULL`, the default, leaves the cluster's own limit. Sent as the `query_max_run_time` session property.

`allow_http_auth` Set to `TRUE` to send `auth` credentials to an `http://` host, which carries them unencrypted. Only for a connection that is encrypted by other means, such as an SSH tunnel.

`...` Unused, for compatibility with the generic.

Value

A TrinoConnection object.

Timeouts

A query is a series of HTTP requests, so two limits apply. `timeout` bounds each request: a coordinator or network that stops answering fails with an error instead of blocking the R session. `query_max_run_time` bounds the query as a whole on the server: past it, Trino stops the query and RTrino raises a `trino_query_error` whose `error_name` is `"EXCEEDED_TIME_LIMIT"`. In a Shiny app, where a blocked session blocks its user, set both.

Examples

```
# The examples in this package run against the coordinator that
# RTRINO_TEST_URL points at, such as "http://localhost:8080" for
# `docker run -p 8080:8080 trinodb/trino`. It has no authentication.
con <- DBI::dbConnect(
  RTrino::Trino(),
  host = Sys.getenv("RTRINO_TEST_URL"),
  catalog = "tpch", schema = "tiny"
```

```
)
con
DBI::dbDisconnect(con)

# LDAP over TLS, with limits for use in a Shiny app
## Not run:
con <- DBI::dbConnect(
  RTrino::Trino(),
  host    = "https://trino.example.com",
  port    = 443,
  user    = Sys.getenv("TRINO_USER"),
  catalog = "hive",
  schema  = "analytics",
  auth    = trino_auth_basic(Sys.getenv("TRINO_USER"),
                             Sys.getenv("TRINO_PASSWORD")),
  timeout = 30,
  query_max_run_time = "5m"
)
DBI::dbDisconnect(con)

## End(Not run)
```

`dbCreateTable,TrinoConnection-method`

Create a table

Description

Trino has no temporary tables, so `temporary = TRUE` is an error rather than being silently ignored.

Usage

```
## S4 method for signature 'TrinoConnection'
dbCreateTable(conn, name, fields, ..., row.names = NULL, temporary = FALSE)
```

Arguments

`conn`	A TrinoConnection object.
`name`	Table name, in any form `dbExistsTable()` accepts.
`fields`	Either a named character vector of Trino types, one per column, or a data frame whose columns' R types are mapped to Trino types with `dbDataType()`.
`...`	Unused, for compatibility with the generic.
`row.names`	Unused: RTrino never writes row names as a column.
`temporary`	Must be `FALSE`.

Value

`TRUE`, invisibly.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "memory", schema = "default")
DBI::dbCreateTable(con, "sales", c(id = "bigint", label = "varchar"))
DBI::dbListFields(con, "sales")
DBI::dbRemoveTable(con, "sales")

# The column types can also follow a data frame's
DBI::dbCreateTable(con, "sales", data.frame(id = 1L, label = "a"))
DBI::dbRemoveTable(con, "sales")

DBI::dbDisconnect(con)
```

---

dbDataType,TrinoConnection-method

Map an R type to a Trino type

Description

The inverse of `trino_type_to_r()`: used by `dbCreateTable()` to pick a column type when the caller does not give one explicitly.

Usage

```
## S4 method for signature 'TrinoConnection'
dbDataType(dbObj, obj, ...)
```

Arguments

`dbObj`	A TrinoConnection object.
`obj`	An R vector, or a data frame (one call per column).
`...`	Unused, for compatibility with the generic.

Value

A string naming a Trino type.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbDataType(con, 1L)
DBI::dbDataType(con, Sys.Date())
DBI::dbDataType(con, data.frame(id = 1L, label = "a"))
DBI::dbDisconnect(con)
```

dbDisconnect,TrinoConnection-method

Disconnect from Trino

Description

Marks the connection as closed. Trino's REST API is stateless, so nothing is sent to the server; any later use of conn raises an error.

Usage

```
## S4 method for signature 'TrinoConnection'
dbDisconnect(conn, ...)
```

Arguments

conn	A TrinoConnection object.
...	Unused, for compatibility with the generic.

Value

TRUE, invisibly.

dbExecute,TrinoConnection,character-method

Execute a statement that returns no rows

Description

Runs the statement to completion and reports how many rows Trino says it changed: the count of an INSERT, an UPDATE, a DELETE, a MERGE or a CREATE TABLE ... AS SELECT, and 0 for a statement that changes no rows, such as CREATE TABLE or DROP TABLE.

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbExecute(conn, statement, ...)
```

Arguments

conn	A TrinoConnection object.
statement	A single SQL statement.
...	Unused, for compatibility with the generic.

Value

The number of affected rows, invisibly: a double, or NA for a data change whose count Trino did not report.

---

dbExistsTable,TrinoConnection,character-method

Does a table exist?

Description

Asks information_schema for a count, so the answer costs the same whatever the schema holds. Trino folds every identifier to lower case, so the name is matched the same way: "SALES" exists if sales does, as SELECT * FROM SALES would find it.

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbExistsTable(conn, name, ...)

## S4 method for signature 'TrinoConnection,Id'
dbExistsTable(conn, name, ...)

## S4 method for signature 'TrinoConnection,ANY'
dbExistsTable(conn, name, ...)
```

Arguments

conn	A TrinoConnection object.
name	Table name, required. A string: "table" resolves against the connection's catalog and schema, "schema.table" keeps its catalog, and "catalog.schema.table" is used as given. Also a DBI::Id(), with components named catalog, schema and table, or a quoted identifier from DBI::SQL() or DBI::dbQuoteIdentifier().
...	Unused, for compatibility with the generic.

Value

A logical scalar. A name in a catalog or schema that does not exist is simply FALSE, not an error.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbExistsTable(con, "nation")
DBI::dbExistsTable(con, "tpch.sf1.nation")
DBI::dbExistsTable(con, DBI::Id(catalog = "tpch", schema = "sf1",
                                table = "nation"))
DBI::dbExistsTable(con, "no_such_table")
DBI::dbDisconnect(con)
```

dbFetch,TrinoResult-method

Fetch rows from a Trino result

Description

Pulls rows from the cursor, following Trino's `nextUri` chain as needed, and converts each column to its R type according to the column's declared Trino type and the connection's `bigint` setting.

Usage

```
## S4 method for signature 'TrinoResult'
dbFetch(res, n = -1, ...)
```

Arguments

`res`	A TrinoResult object.
`n`	Number of rows to fetch. `-1` or `Inf` (the default is `-1`) fetches every remaining row, paginating until the query finishes. A positive `n` returns at most that many rows, pulling as many pages as needed and keeping the remainder buffered on the cursor for the next call. `0` returns no rows but the result's columns, with their types.
`...`	Unused, for compatibility with the generic.

Value

A tibble. Once the result is exhausted, a zero-row tibble with the right columns and types.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
res <- DBI::dbSendQuery(con, "SELECT * FROM orders")
while (!DBI::dbHasCompleted(res)) {
  chunk <- DBI::dbFetch(res, n = 4000)
  # process chunk; here, just count its rows
  cat(nrow(chunk), "rows\n")
}
DBI::dbClearResult(res)
DBI::dbDisconnect(con)
```

dbGetInfo,TrinoConnection-method

Metadata about a Trino connection

Description

Reports the connection's settings plus the coordinator's version, read from `/v1/info`. If that endpoint cannot be reached, `db.version` is `NA` rather than an error, so the rest of the metadata stays available.

Usage

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

Arguments

`dbObj`	A TrinoConnection object.
`...`	Unused, for compatibility with the generic.

Value

A list with `db.version`, `host`, `port`, `user`, `catalog` and `schema`.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbGetInfo(con)
DBI::dbDisconnect(con)
```

dbGetQuery,TrinoConnection,character-method

Run a query and return all of its rows

Description

Submits the statement, fetches every row and clears the result, even if fetching fails.

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbGetQuery(conn, statement, n = -1, ...)
```

Arguments

conn	A TrinoConnection object.
statement	A single SQL statement.
n	Number of rows to return; -1 (the default) returns all of them.
...	Unused, for compatibility with the generic.

Value

A tibble.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbGetQuery(con, "SELECT * FROM nation LIMIT 5")
DBI::dbGetQuery(con, "SELECT * FROM nation", n = 3)
DBI::dbDisconnect(con)
```

---

dbHasCompleted,TrinoResult-method

Have all rows of a result been consumed?

Description

TRUE once Trino has stopped handing out pages *and* the rows already pulled have been handed to the caller. A result that has finished on the server but still holds buffered rows is not complete, so a while (!dbHasCompleted(res)) loop over dbFetch() sees every row.

Usage

```
## S4 method for signature 'TrinoResult'
dbHasCompleted(res, ...)
```

Arguments

res	A TrinoResult object.
...	Unused, for compatibility with the generic.

Value

A logical scalar.

dbIsValid,TrinoConnection-method

Is this connection or result still usable?

Description

A connection is valid until `dbDisconnect()` is called. A result is valid until it is cleared by `dbClearResult()` - a result whose rows have all been fetched is still valid, so that `dbColumnInfo()` and friends keep working.

Usage

```
## S4 method for signature 'TrinoConnection'
dbIsValid(dbObj, ...)

## S4 method for signature 'TrinoResult'
dbIsValid(dbObj, ...)
```

Arguments

`dbObj`	A TrinoConnection or TrinoResult object.
`...`	Unused, for compatibility with the generic.

Value

A logical scalar.

dbListFields,TrinoConnection,character-method

List a table's columns

Description

List a table's columns

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbListFields(conn, name, ...)

## S4 method for signature 'TrinoConnection,Id'
dbListFields(conn, name, ...)

## S4 method for signature 'TrinoConnection,ANY'
dbListFields(conn, name, ...)
```

Arguments

conn	A TrinoConnection object.
name	Table name, required, in any form dbExistsTable() accepts: a string with one to three dot-separated parts, a `DBI::Id()` or a quoted identifier from `DBI::SQL()`. An unqualified name is resolved against the connection's catalog and schema.
...	Unused, for compatibility with the generic.

Value

A character vector of column names.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbListFields(con, "nation")
DBI::dbListFields(con, DBI::Id(schema = "sf1", table = "region"))
DBI::dbDisconnect(con)
```

---

dbListTables,TrinoConnection-method

List the tables in the connection's schema

Description

List the tables in the connection's schema

Usage

```
## S4 method for signature 'TrinoConnection'
dbListTables(conn, ...)
```

Arguments

conn	A TrinoConnection object.
...	Unused, for compatibility with the generic.

Value

A character vector of table names.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbListTables(con)
DBI::dbDisconnect(con)
```

dbQuoteIdentifier,TrinoConnection,character-method

Quote identifiers and strings for Trino

Description

Trino follows the SQL standard: identifiers are wrapped in double quotes with embedded double quotes doubled, and string literals are wrapped in single quotes with embedded single quotes doubled. A value that is already `DBI::SQL()` is returned unchanged rather than quoted a second time.

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbQuoteIdentifier(conn, x, ...)

## S4 method for signature 'TrinoConnection,SQL'
dbQuoteIdentifier(conn, x, ...)

## S4 method for signature 'TrinoConnection,character'
dbQuoteString(conn, x, ...)

## S4 method for signature 'TrinoConnection,SQL'
dbQuoteString(conn, x, ...)
```

Arguments

conn	A TrinoConnection object.
x	A character vector, or an identifier for `dbQuoteIdentifier()`.
...	Unused, for compatibility with the generic.

Value

A DBI::SQL object.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbQuoteIdentifier(con, "my column")
DBI::dbQuoteString(con, "O'Brien")
DBI::dbDisconnect(con)
```

dbQuoteLiteral,TrinoConnection-method

Quote R values as Trino literals

Description

Turns R values into SQL literals Trino accepts where a value of the matching type is expected. Used by `DBI::sqlInterpolate()` and `glue::glue_sql()`, and so the way to put values into a statement, since RTrino has no parameter binding. Missing values become `NULL`.

Usage

```
## S4 method for signature 'TrinoConnection'
dbQuoteLiteral(conn, x, ...)
```

Arguments

`conn`	A TrinoConnection object.
`x`	A vector to quote.
`...`	Unused, for compatibility with the generic.

Details

- character and factor: `'text'`, with embedded quotes doubled;
- logical: `TRUE` or `FALSE`;
- integer and `integer64`: the integer's digits;
- double: a `DOUBLE` literal that reads back as the same double, such as `0.30000000000000004E0`, or `nan()`, `infinity()` or `-infinity()`;
- `Date`: `DATE '2026-01-15'`;
- `POSIXct`: `TIMESTAMP '2026-01-15 10:30:00.123456'`, the instant as a wall-clock time in the connection's `session.timezone`, which is how Trino reads it back;
- a list of raw vectors: `X'0102FF'`.

Other classes are quoted by DBI's default method.

Value

A DBI::SQL object.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
DBI::dbQuoteLiteral(con, as.Date("2026-01-15"))
DBI::sqlInterpolate(
  con, "SELECT * FROM orders WHERE orderdate >= ?since AND urgent = ?flag",
```

```
  since = as.Date("2026-01-01"), flag = TRUE
)
DBI::dbDisconnect(con)
```

---

dbRemoveTable,TrinoConnection,character-method

Drop a table

Description

Drop a table

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbRemoveTable(conn, name, ...)

## S4 method for signature 'TrinoConnection,Id'
dbRemoveTable(conn, name, ...)

## S4 method for signature 'TrinoConnection,ANY'
dbRemoveTable(conn, name, ...)
```

Arguments

conn	A TrinoConnection object.
name	Table name, in any form `dbExistsTable()` accepts.
...	Unused, for compatibility with the generic.

Value

`TRUE`, invisibly.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "memory", schema = "default")
DBI::dbWriteTable(con, "sales", data.frame(id = 1:3))
DBI::dbRemoveTable(con, "sales")
DBI::dbExistsTable(con, "sales")
DBI::dbDisconnect(con)
```

dbRenameTable *Rename a table*

Description

Not a DBI generic: DBI has no `dbRenameTable()`, so RTrino defines its own, the way RPresto does for Presto.

Usage

```
dbRenameTable(conn, name, new_name, ...)

## S4 method for signature 'TrinoConnection'
dbRenameTable(conn, name, new_name, ...)
```

Arguments

`conn`	A TrinoConnection object.
`name`	Current table name, in any form `dbExistsTable()` accepts.
`new_name`	New table name, in the same forms.
`...`	Unused, for compatibility with the generic.

Value

`TRUE`, invisibly.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "memory", schema = "default")
DBI::dbWriteTable(con, "sales", data.frame(id = 1:3))
dbRenameTable(con, "sales", "sales_old")
DBI::dbExistsTable(con, "sales_old")
DBI::dbRemoveTable(con, "sales_old")
DBI::dbDisconnect(con)
```

dbSendQuery,TrinoConnection,character-method *Submit a statement to Trino*

Description

Posts the statement to `/v1/statement` and returns immediately with a TrinoResult cursor; the rows are pulled by `dbFetch()`. A statement that fails during planning raises an error here, because Trino reports it in the response to the initial `POST`.

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbSendQuery(conn, statement, ...)
```

Arguments

conn	A TrinoConnection object.
statement	A single SQL statement, without a trailing semicolon.
...	Unused, for compatibility with the generic.

Details

Query parameters are not supported: passing `params` is an error rather than being silently ignored. Put values into the statement with `DBI::sqlInterpolate()` or `glue::glue_sql()`, which quote them with `DBI::dbQuoteLiteral()` as Trino literals.

Value

A TrinoResult object.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "tpch", schema = "tiny")
res <- DBI::dbSendQuery(con, "SELECT 1 AS n")
DBI::dbFetch(res)
DBI::dbClearResult(res)

# Values go into the SQL as literals, quoted for Trino
sql <- DBI::sqlInterpolate(
  con, "SELECT count(*) AS n FROM orders WHERE orderdate >= ?since",
  since = as.Date("1998-01-01")
)
DBI::dbGetQuery(con, sql)

DBI::dbDisconnect(con)
```

---

dbSendStatement,TrinoConnection,character-method

Submit a statement that changes something

Description

Unlike dbSendQuery(), runs the statement to completion before returning, so that `dbGetRowsAffected()` answers straight away, as DBI requires: Trino only reports the count once the statement has finished.

Usage

```
## S4 method for signature 'TrinoConnection,character'
dbSendStatement(conn, statement, ...)
```

Arguments

conn	A TrinoConnection object.
statement	A single SQL statement, without a trailing semicolon.
...	Unused, for compatibility with the generic.

Value

A TrinoResult object whose statement has finished.

---

dbWriteTable,TrinoConnection,ANY-method

Write a table

Description

Creates the table with dbCreateTable() and fills it with dbAppendTable(). CREATE TABLE cannot be undone by dbRollback() on most connectors, so overwrite = TRUE plays it safe on its own: it renames the existing table rather than dropping it, and restores it if anything fails before the new table is fully written.

Usage

```
## S4 method for signature 'TrinoConnection,ANY'
dbWriteTable(
  conn,
  name,
  value,
  ...,
  overwrite = FALSE,
  append = FALSE,
  field.types = NULL,
  temporary = FALSE,
  row.names = NULL,
  chunk_size = 1000L
)

## S4 method for signature 'TrinoConnection,Id'
dbWriteTable(conn, name, value, ...)
```

Arguments

conn	A TrinoConnection object.
name	Table name, in any form dbExistsTable() accepts.
value	A data frame to write.
...	Unused, for compatibility with the generic.
overwrite	Drop and recreate the table if it already exists.
append	Add to the table if it already exists, instead of creating it.
field.types	A named character vector of Trino types, overriding the types dbDataType() would otherwise infer. Not allowed together with append.
temporary	Must be FALSE: Trino has no temporary tables.
row.names	Unused: RTrino never writes row names as a column.
chunk_size	Passed to dbAppendTable().

Value

TRUE, invisibly.

Examples

```
con <- DBI::dbConnect(RTrino::Trino(), host = Sys.getenv("RTRINO_TEST_URL"),
                      catalog = "memory", schema = "default")
DBI::dbWriteTable(con, "sales", data.frame(id = 1:3, label = letters[1:3]))
DBI::dbWriteTable(con, "sales", data.frame(id = 4L, label = "d"),
                  append = TRUE)
DBI::dbGetQuery(con, "SELECT * FROM sales ORDER BY id")

# Replace the table and its contents
DBI::dbWriteTable(con, "sales", data.frame(id = 9L, label = "z"),
                  overwrite = TRUE)
DBI::dbGetQuery(con, "SELECT * FROM sales")

DBI::dbRemoveTable(con, "sales")
DBI::dbDisconnect(con)
```

Trino — *Instantiate the Trino driver*

Description

Creates the driver object passed as the first argument to DBI::dbConnect(). The call has no side effects and always succeeds.

Usage

```
Trino()
```

Value

A TrinoDriver object.

Examples

```
drv <- Trino()
drv
```

---

trino_auth *Authentication methods for Trino*

Description

Each helper returns a *closure* that takes an `httr2` request and returns it with the appropriate credentials attached. The credentials live only in that closure's environment, so they are never written to the global environment and never stored on the connection object in plain sight. Pass the result as the `auth` argument of `dbConnect()`.

Usage

```
trino_auth_basic(user, password)

trino_auth_jwt(token)

trino_auth_oauth2(client_id, client_secret, token_url, scope = NULL, ...)
```

Arguments

`user`	User name.
`password`	Password. Read it from the environment (`Sys.getenv("TRINO_PASSWORD")`) rather than writing it in a script.
`token`	A JWT, as a string.
`client_id`	OAuth2 client id.
`client_secret`	OAuth2 client secret.
`token_url`	Token endpoint of the OAuth2 provider.
`scope`	Optional OAuth2 scope.
`...`	Further arguments passed on to `httr2::req_oauth_client_credentials()`.

Details

- `trino_auth_basic()` — HTTP basic authentication, used by Trino's LDAP and password-file authenticators.
- `trino_auth_jwt()` — a bearer JWT, for service accounts and unattended pipelines, or for a cluster fronted by an SSO proxy that mints its own tokens.
- `trino_auth_oauth2()` — OAuth2 client credentials, for corporate identity providers such as Okta or Entra ID. `httr2` caches the token and renews it when it expires.

Trino rejects basic and bearer credentials over plain HTTP, so use an `https://` host with all three.

Value

A function of one argument (an `httr2` request) returning a modified request.

Examples

```
# Only builds the credential; nothing is sent until a connection uses it
auth <- trino_auth_basic("analyst", Sys.getenv("TRINO_PASSWORD"))

# Needs a cluster that authenticates with a password
## Not run:
con <- DBI::dbConnect(
  RTrino::Trino(),
  host    = "https://trino.example.com",
  port    = 443,
  user    = "analyst",
  catalog = "hive",
  schema  = "default",
  auth    = auth
)

## End(Not run)
```

---

trino_ssl — *TLS options for a Trino connection*

Description

Collects the TLS settings applied to every request made on a connection. Pass the result as the `ssl_options` argument of `dbConnect()`.

Usage

```
trino_ssl(verify = TRUE, ca_bundle = NULL)
```

Arguments

`verify`	Whether to verify the server's certificate. Defaults to `TRUE`.
`ca_bundle`	Path to a PEM file holding the certificate authorities to trust, or `NULL` to use the system store.

Details

A cluster whose certificate is signed by an internal authority does not need verification turned off: point `ca_bundle` at the authority's PEM file and certificates keep being checked. `verify = FALSE` accepts any certificate, including an attacker's, and so `dbConnect()` warns once when it opens a connection with it.

Value

A list of TLS options.

Examples

```
# Secure default
trino_ssl()

# Trust an internal certificate authority
## Not run:
trino_ssl(ca_bundle = "/etc/ssl/certs/internal-ca.pem")

## End(Not run)
```

---

trino_type_to_r *Map a Trino type to the name of an R type*

Description

Reports the R type a column of the given Trino type is converted to. Mostly useful for documentation and testing; `dbFetch()` converts columns through `trino_cast_column()`.

Usage

```
trino_type_to_r(type, bigint = c("integer64", "numeric", "character"))
```

Arguments

`type`	A Trino type name, with or without parameters, as a single string.
`bigint`	How `BIGINT` is handled; see `dbConnect()`.

Value

A string naming an R class.

Examples

```
trino_type_to_r("varchar(10)")
trino_type_to_r("bigint")
trino_type_to_r("bigint", bigint = "numeric")
trino_type_to_r("timestamp(3) with time zone")
```

Index

dbAppendTable(), *22, 23*
dbAppendTable,TrinoConnection-method, 3
dbBegin,TrinoConnection-method, 4
dbBind,TrinoResult-method, 5
dbClearResult(), *15*
dbClearResult,TrinoResult-method, 5
dbColumnInfo(), *15*
dbCommit(), *4*
dbCommit,TrinoConnection-method (dbBegin,TrinoConnection-method), 4
dbConnect(), *24–26*
dbConnect,TrinoDriver-method, 6
dbCreateTable(), *9*, *22*
dbCreateTable,TrinoConnection-method, 8
dbDataType(), *8*, *23*
dbDataType,TrinoConnection-method, 9
dbDisconnect(), *15*
dbDisconnect,TrinoConnection-method, 10
dbExecute,TrinoConnection,character-method, 10
dbExistsTable(), *3*, *8*, *16*, *19, 20*, *23*
dbExistsTable,TrinoConnection,ANY-method (dbExistsTable,TrinoConnection,character-method), 11
dbExistsTable,TrinoConnection,character-method, 11
dbExistsTable,TrinoConnection,Id-method (dbExistsTable,TrinoConnection,character-method), 11
dbFetch(), *14*, *20*, *26*
dbFetch,TrinoResult-method, 12
dbGetInfo,TrinoConnection-method, 13
dbGetQuery,TrinoConnection,character-method, 13
dbGetRowsAffected(), *21*
dbHasCompleted,TrinoResult-method, 14
DBI::dbConnect(), *23*
DBI::dbQuoteIdentifier(), *11*
DBI::dbQuoteLiteral(), *21*
DBI::Id(), *11*, *16*
DBI::SQL, *17, 18*
DBI::SQL(), *11*, *16, 17*
DBI::sqlAppendTable(), *3*
DBI::sqlInterpolate(), *18*, *21*
dbIsValid,TrinoConnection-method, 15
dbIsValid,TrinoResult-method (dbIsValid,TrinoConnection-method), 15
dbListFields,TrinoConnection,ANY-method (dbListFields,TrinoConnection,character-method), 15
dbListFields,TrinoConnection,character-method, 15
dbListFields,TrinoConnection,Id-method (dbListFields,TrinoConnection,character-method), 15
dbListTables,TrinoConnection-method, 16
dbQuoteIdentifier(), *3*
dbQuoteIdentifier,TrinoConnection,character-method, 17
dbQuoteIdentifier,TrinoConnection,SQL-method (dbQuoteIdentifier,TrinoConnection,character-metho 17
dbQuoteLiteral(), *3*
dbQuoteLiteral,TrinoConnection-method, 18
dbQuoteString,TrinoConnection,character-method (dbQuoteIdentifier,TrinoConnection,character-metho 17
dbQuoteString,TrinoConnection,SQL-method (dbQuoteIdentifier,TrinoConnection,character-metho 17
dbRemoveTable,TrinoConnection,ANY-method

(dbRemoveTable,TrinoConnection,character-method), 19
dbRemoveTable,TrinoConnection,character-method, 19
dbRemoveTable,TrinoConnection,Id-method (dbRemoveTable,TrinoConnection,character-method), 19
dbRenameTable, 20
dbRenameTable,TrinoConnection-method (dbRenameTable), 20
dbRollback(), *4*, *22*
dbRollback,TrinoConnection-method (dbBegin,TrinoConnection-method), 4
dbSendQuery(), *21*
dbSendQuery,TrinoConnection,character-method, 20
dbSendStatement,TrinoConnection,character-method, 21
dbWriteTable,TrinoConnection,ANY-method, 22
dbWriteTable,TrinoConnection,Id-method (dbWriteTable,TrinoConnection,ANY-method), 22

httr2::req_oauth_client_credentials(), *24*

tibble, *12*, *14*
Trino, 23
Trino(), *6*
trino_auth, 24
trino_auth_basic (trino_auth), 24
trino_auth_basic(), *7*
trino_auth_jwt (trino_auth), 24
trino_auth_jwt(), *7*
trino_auth_oauth2 (trino_auth), 24
trino_auth_oauth2(), *7*
trino_ssl, 25
trino_ssl(), *7*
trino_type_to_r, 26
trino_type_to_r(), *9*
TrinoConnection, *3*, *4*, *6–11*, *13–23*
TrinoDriver, *6*, *24*
TrinoResult, *5*, *12*, *14*, *15*, *20–22*