Package: RSQLite (via r-universe)

October 28, 2026

Title SQLite Interface for R

Version 3.53.3

Date 2026-06-28

Description Embeds the SQLite database engine in R and provides an interface compliant with the DBI package. The source for the SQLite engine and for various extensions is included. System libraries will never be consulted because this package relies on static linking for the plugins it includes; this also ensures a consistent experience across all installations. Optionally, when libcurl is available at build time, an experimental HTTP/HTTPS virtual file system (VFS) can be enabled to allow read-only access to remote immutable SQLite database files via URIs.

License LGPL (>= 2.1)

URL https://rsqlite.r-dbi.org, https://github.com/r-dbi/RSQLite

BugReports https://github.com/r-dbi/RSQLite/issues

Depends R (>= 3.1.0)

Imports bit64, blob (>= 1.2.0), DBI (>= 1.2.0), memoise, methods, pkgconfig, rlang

Suggests callr, cli, DBItest (>= 1.8.0), decor, gert, gh, hms, httpuv, knitr, magrittr, rmarkdown, rvest, testthat (>= 3.0.0), withr, xml2

LinkingTo cpp11 (>= 0.4.0)

VignetteBuilder knitr

Config/Needs/website r-dbi/dbitemplate

Config/autostyle/scope line_breaks

Config/autostyle/strict false

Config/testthat/edition 3

Encoding UTF-8

Collate 'SQLiteConnection.R' 'SQLKeywords_SQLiteConnection.R'
'SQLiteDriver.R' 'SQLite.R' 'SQLiteResult.R' 'coerce.R'
'compatRowNames.R' 'copy.R' 'cpp11.R' 'datasetsDb.R'
'dbAppendTable_SQLiteConnection.R' 'dbBeginTransaction.R'
'dbBegin_SQLiteConnection.R' 'dbBind_SQLiteResult.R'
'dbClearResult_SQLiteResult.R' 'dbColumnInfo_SQLiteResult.R'
'dbCommit_SQLiteConnection.R' 'dbConnect_SQLiteConnection.R'
'dbConnect_SQLiteDriver.R' 'dbDataType_SQLiteConnection.R'
'dbDataType_SQLiteDriver.R' 'dbDisconnect_SQLiteConnection.R'
'dbExistsTable_SQLiteConnection_Id.R'
'dbExistsTable_SQLiteConnection_character.R'
'dbFetch_SQLiteResult.R' 'dbGetException_SQLiteConnection.R'
'dbGetInfo_SQLiteConnection.R' 'dbGetInfo_SQLiteDriver.R'
'dbGetPreparedQuery.R'
'dbGetPreparedQuery_SQLiteConnection_character_data.frame.R'
'dbGetRowCount_SQLiteResult.R'
'dbGetRowsAffected_SQLiteResult.R'
'dbGetStatement_SQLiteResult.R' 'dbHasCompleted_SQLiteResult.R'
'dbIsValid_SQLiteConnection.R' 'dbIsValid_SQLiteDriver.R'
'dbIsValid_SQLiteResult.R' 'dbListObjects_SQLiteConnection.R'
'dbListResults_SQLiteConnection.R'
'dbListTables_SQLiteConnection.R'
'dbQuoteIdentifier_SQLiteConnection_SQL.R'
'dbQuoteIdentifier_SQLiteConnection_character.R'
'dbReadTable_SQLiteConnection_character.R'
'dbRemoveTable_SQLiteConnection_character.R'
'dbRollback_SQLiteConnection.R' 'dbSendPreparedQuery.R'
'dbSendPreparedQuery_SQLiteConnection_character_data.frame.R'
'dbSendQuery_SQLiteConnection_character.R'
'dbUnloadDriver_SQLiteDriver.R'
'dbUnquoteIdentifier_SQLiteConnection_SQL.R'
'dbWriteTable_SQLiteConnection_character_character.R'
'dbWriteTable_SQLiteConnection_character_data.frame.R'
'db_bind.R' 'deprecated.R' 'export.R' 'fetch_SQLiteResult.R'
'httpvfs.R' 'httpvfs_config.R'
'import-standalone-check_suggested.R'
'import-standalone-purrr.R' 'initExtension.R' 'initRegExp.R'
'isSQLKeyword_SQLiteConnection_character.R'
'make.db.names_SQLiteConnection_character.R' 'pkgconfig.R'
'show_SQLiteConnection.R' 'sqlData_SQLiteConnection.R'
'table.R' 'transactions.R' 'utils.R' 'version.R' 'zzz.R'

Config/roxygen2/version 8.0.0.9000

NeedsCompilation yes

Author Kirill Müller [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1416-3412>), Hadley Wickham [aut],
David A. James [aut], Seth Falcon [aut], D. Richard Hipp [ctb]
(for the included SQLite sources), Dan Kennedy [ctb] (for the
included SQLite sources), Joe Mistachkin [ctb] (for the

included SQLite sources), SQLite Authors [ctb] (for the
included SQLite sources), Liam Healy [ctb] (for the included
SQLite sources), R Consortium [fnd], RStudio [cph]

Maintainer Kirill Müller <kirill@cynkra.com>

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

Date/Publication 2026-07-01 01:35:31 UTC

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

RemoteRef HEAD

RemoteSha 846a49ec677c9ef396fcce23c7c48e8001ceeafd

Contents

datasetsDb . . . 3
dbBegin_SQLiteConnection . . . 4
dbReadTable_SQLiteConnection_character . . . 5
dbWriteTable_SQLiteConnection_character_character . . . 6
initExtension . . . 9
rsqliteVersion . . . 11
SQLite . . . 11
sqliteCopyDatabase . . . 14
sqliteHasHttpVFS . . . 15
sqliteHttpConfig . . . 15
sqliteHttpStats . . . 16
sqliteHttpVfsCompiled . . . 17
sqliteRemote . . . 18
sqliteSetBusyHandler . . . 18

Index **20**

`datasetsDb` *A sample sqlite database*

Description

This database is bundled with the package, and contains all data frames in the datasets package.

Usage

```
datasetsDb()
```

Examples

```
library(DBI)
db <- RSQLite::datasetsDb()
dbListTables(db)

dbReadTable(db, "CO2")
dbGetQuery(db, "SELECT * FROM CO2 WHERE conc < 100")

dbDisconnect(db)
```

---

dbBegin_SQLiteConnection

SQLite transaction management

Description

By default, SQLite is in auto-commit mode. `dbBegin()` starts a SQLite transaction and turns auto-commit off. `dbCommit()` and `dbRollback()` commit and rollback the transaction, respectively and turn auto-commit on. `DBI::dbWithTransaction()` is a convenient wrapper that makes sure that `dbCommit()` or `dbRollback()` is called. A helper function `sqliteIsTransacting()` is available to check the current transaction status of the connection.

Usage

```
## S4 method for signature 'SQLiteConnection'
dbBegin(conn, .name = NULL, ..., name = NULL)

## S4 method for signature 'SQLiteConnection'
dbCommit(conn, .name = NULL, ..., name = NULL)

## S4 method for signature 'SQLiteConnection'
dbRollback(conn, .name = NULL, ..., name = NULL)

sqliteIsTransacting(conn)
```

Arguments

`conn`	a `SQLiteConnection` object, produced by `DBI::dbConnect()`
`.name`	For backward compatibility, do not use.
`...`	Needed for compatibility with generic. Otherwise ignored.
`name`	Supply a name to use a named savepoint. This allows you to nest multiple transaction

See Also

The corresponding generic functions `DBI::dbBegin()`, `DBI::dbCommit()`, and `DBI::dbRollback()`.

Examples

```
library(DBI)
con <- dbConnect(SQLite(), ":memory:")
dbWriteTable(con, "arrests", datasets::USArrests)
dbGetQuery(con, "select count(*) from arrests")

dbBegin(con)
rs <- dbSendStatement(con, "DELETE from arrests WHERE Murder > 1")
dbGetRowsAffected(rs)
dbClearResult(rs)

dbGetQuery(con, "select count(*) from arrests")

dbRollback(con)
dbGetQuery(con, "select count(*) from arrests")[1, ]

dbBegin(con)
rs <- dbSendStatement(con, "DELETE FROM arrests WHERE Murder > 5")
dbClearResult(rs)
dbCommit(con)
dbGetQuery(con, "SELECT count(*) FROM arrests")[1, ]

# Named savepoints can be nested ------------------------------------------
dbBegin(con, name = "a")
dbBegin(con, name = "b")
sqliteIsTransacting(con)
dbRollback(con, name = "b")
dbCommit(con, name = "a")

dbDisconnect(con)
```

---

dbReadTable_SQLiteConnection_character

Read a database table

Description

Returns the contents of a database table given by name as a data frame.

Usage

```
## S4 method for signature 'SQLiteConnection,character'
dbReadTable(
  conn,
  name,
  ...,
  row.names = pkgconfig::get_config("RSQLite::row.names.table", FALSE),
  check.names = TRUE,
  select.cols = NULL
)
```

Arguments

conn	a SQLiteConnection object, produced by DBI::dbConnect()
name	a character string specifying a table name. SQLite table names are *not* case sensitive, e.g., table names ABC and abc are considered equal.
...	Needed for compatibility with generic. Otherwise ignored.
row.names	Either TRUE, FALSE, NA or a string. If TRUE, always translate row names to a column called "row_names". If FALSE, never translate row names. If NA, translate rownames only if they're a character vector. A string is equivalent to TRUE, but allows you to override the default name. For backward compatibility, NULL is equivalent to FALSE.
check.names	If TRUE, the default, column names will be converted to valid R identifiers.
select.cols	Deprecated, do not use.

Details

Note that the data frame returned by dbReadTable() only has primitive data, e.g., it does not coerce character data to factors.

Value

A data frame.

See Also

The corresponding generic function DBI::dbReadTable().

Examples

```
library(DBI)
db <- RSQLite::datasetsDb()
dbReadTable(db, "mtcars")
dbReadTable(db, "mtcars", row.names = FALSE)
dbDisconnect(db)
```

---

dbWriteTable_SQLiteConnection_character_character

Write a local data frame or file to the database

Description

Functions for writing data frames or delimiter-separated files to database tables.

Usage

```
## S4 method for signature 'SQLiteConnection,character,character'
dbWriteTable(
  conn,
  name,
  value,
  ...,
  field.types = NULL,
  overwrite = FALSE,
  append = FALSE,
  header = TRUE,
  colClasses = NA,
  row.names = FALSE,
  nrows = 50,
  sep = ",",
  eol = "\n",
  skip = 0,
  temporary = FALSE
)

## S4 method for signature 'SQLiteConnection,character,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  ...,
  row.names = pkgconfig::get_config("RSQLite::row.names.table", FALSE),
  overwrite = FALSE,
  append = FALSE,
  field.types = NULL,
  temporary = FALSE
)
```

Arguments

conn	a SQLiteConnection object, produced by DBI::dbConnect()
name	a character string specifying a table name. SQLite table names are *not* case sensitive, e.g., table names ABC and abc are considered equal.
value	a data.frame (or coercible to data.frame) object or a file name (character). In the first case, the data.frame is written to a temporary file and then imported to SQLite; when value is a character, it is interpreted as a file name and its contents imported to SQLite.
...	Needed for compatibility with generic. Otherwise ignored.
field.types	character vector of named SQL field types where the names are the names of new table's columns. If missing, types inferred with DBI::dbDataType()).
overwrite	a logical specifying whether to overwrite an existing table or not. Its default is FALSE.

append	a logical specifying whether to append to an existing table in the DBMS. Its default is FALSE.
header	is a logical indicating whether the first data line (but see skip) has a header or not. If missing, it value is determined following read.table() convention, namely, it is set to TRUE if and only if the first row has one fewer field that the number of columns.
colClasses	Character vector of R type names, used to override defaults when imputing classes from on-disk file.
row.names	A logical specifying whether the row.names should be output to the output DBMS table; if TRUE, an extra field whose name will be whatever the R identifier "row.names" maps to the DBMS (see DBI::make.db.names()). If NA will add rows names if they are characters, otherwise will ignore.
nrows	Number of rows to read to determine types.
sep	The field separator, defaults to ','.
eol	The end-of-line delimiter, defaults to '\n'.
skip	number of lines to skip before reading the data. Defaults to 0.
temporary	a logical specifying whether the new table should be temporary. Its default is FALSE.

Details

In a primary key column qualified with AUTOINCREMENT, missing values will be assigned the next largest positive integer, while nonmissing elements/cells retain their value. If the autoincrement column exists in the data frame passed to the value argument, the NA elements are overwritten. Similarly, if the key column is not present in the data frame, all elements are automatically assigned a value.

See Also

The corresponding generic function DBI::dbWriteTable().

Examples

```
con <- dbConnect(SQLite())
dbWriteTable(con, "mtcars", mtcars)
dbReadTable(con, "mtcars")

# A zero row data frame just creates a table definition.
dbWriteTable(con, "mtcars2", mtcars[0, ])
dbReadTable(con, "mtcars2")

dbDisconnect(con)
```

`initExtension` *Add useful extension functions*

Description

Several extension functions are included in the **RSQLite** package. When enabled via `initExtension()`, these extension functions can be used in SQL queries. Extensions must be enabled separately for each connection.

Usage

```
initExtension(
  db,
  extension = c("math", "regexp", "series", "csv", "uuid", "http")
)
```

Arguments

`db`	A `SQLiteConnection` object to load these extensions into.
`extension`	The extension to load.

Details

The `"math"` extension functions are written by Liam Healy and made available through the SQLite website (`https://www.sqlite.org/src/ext/contrib`). This package contains a slightly modified version of the original code. See the section "Available functions in the math extension" for details.

The `"regexp"` extension provides a regular-expression matcher for POSIX extended regular expressions, as available through the SQLite source code repository (`https://sqlite.org/src/file?filename=ext/misc/regexp.c`). SQLite will then implement the `A regexp B` operator, where `A` is the string to be matched and `B` is the regular expression.

The `"series"` extension loads the table-valued function `generate_series()`, as available through the SQLite source code repository (`https://sqlite.org/src/file?filename=ext/misc/series.c`).

The `"csv"` extension loads the function `csv()` that can be used to create virtual tables, as available through the SQLite source code repository (`https://sqlite.org/src/file?filename=ext/misc/csv.c`).

The `"http"` extension registers an HTTP/HTTPS virtual file system (VFS) that allows opening remote databases via URI filenames, e.g., "file:https://host/path/db.sqlite?vfs=http&immutable=1". This implementation is experimental and not an official SQLite extension; it fetches pages on demand using HTTP Range requests and serves reads from an in-memory page cache, with an optional full-download fallback depending on server support and configuration. It is primarily intended for read-only access to small, immutable databases; see `sqliteHttpConfig()` and `sqliteRemote()` for configuration options and usage examples. Building this extension may require libcurl and is optional in RSQLite.

The "uuid" extension loads the functions `uuid()`, `uuid_str(X)` and `uuid_blob(X)` that can be used to create universally unique identifiers, as available through the SQLite source code repository (https://sqlite.org/src/file?filename=ext/misc/uuid.c).

Available functions in the math extension

Math functions acos, acosh, asin, asinh, atan, atan2, atanh, atn2, ceil, cos, cosh, cot, coth, degrees, difference, exp, floor, log, log10, pi, power, radians, sign, sin, sinh, sqrt, square, tan, tanh

String functions charindex, leftstr, ltrim, padc, padl, padr, proper, replace, replicate, reverse, rightstr, rtrim, strfilter, trim

Aggregate functions stdev, variance, mode, median, lower_quartile, upper_quartile

Examples

```
library(DBI)
db <- RSQLite::datasetsDb()

# math
RSQLite::initExtension(db)
dbGetQuery(db, "SELECT stdev(mpg) FROM mtcars")
sd(mtcars$mpg)

# regexp
RSQLite::initExtension(db, "regexp")
dbGetQuery(db, "SELECT * FROM mtcars WHERE carb REGEXP '[12]'")

# series
RSQLite::initExtension(db, "series")
dbGetQuery(db, "SELECT value FROM generate_series(0, 20, 5);")

dbDisconnect(db)

# csv
db <- dbConnect(RSQLite::SQLite())
RSQLite::initExtension(db, "csv")
# use the filename argument to mount CSV files from disk
sql <- paste0(
  "CREATE VIRTUAL TABLE tbl USING ",
  "csv(data='1,2', schema='CREATE TABLE x(a INT, b INT)')"
)
dbExecute(db, sql)
dbGetQuery(db, "SELECT * FROM tbl")

# uuid
db <- dbConnect(RSQLite::SQLite())
RSQLite::initExtension(db, "uuid")
dbGetQuery(db, "SELECT uuid();")
dbDisconnect(db)
```

rsqliteVersion *RSQLite version*

Description

Return the version of RSQLite.

Usage

```
rsqliteVersion()
```

Value

A character vector containing header and library versions of RSQLite.

Examples

```
RSQLite::rsqliteVersion()
```

SQLite *Connect to an SQLite database*

Description

Together, `SQLite()` and `dbConnect()` allow you to connect to a SQLite database file. See `DBI::dbSendQuery()` for how to issue queries and receive results.

Usage

```
SQLite(...)

## S4 method for signature 'SQLiteConnection'
dbConnect(drv, ...)

## S4 method for signature 'SQLiteDriver'
dbConnect(
  drv,
  dbname = "",
  ...,
  loadable.extensions = TRUE,
  default.extensions = loadable.extensions,
  cache_size = NULL,
  synchronous = "off",
  flags = SQLITE_RWC,
  vfs = NULL,
  bigint = c("integer64", "integer", "numeric", "character"),
```

```
  extended_types = FALSE
)

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

Arguments

`...`	In previous versions, `SQLite()` took arguments. These have now all been moved to `dbConnect()`, and any arguments here will be ignored with a warning.
`drv, conn`	An objected generated by `SQLite()`, or an existing `SQLiteConnection`. If an connection, the connection will be cloned.
`dbname`	The path to the database file. SQLite keeps each database instance in one single file. The name of the database *is* the file name, thus database names should be legal file names in the running platform. There are two exceptions: • `""` will create a temporary on-disk database. The file will be deleted when the connection is closed. • `":memory:"` or `"file::memory:"` will create a temporary in-memory database.
`loadable.extensions`	When `TRUE` (default) SQLite3 loadable extensions are enabled. Setting this value to `FALSE` prevents extensions from being loaded.
`default.extensions`	When `TRUE` (default) the `initExtension()` function will be called on the new connection.Setting this value to `FALSE` requires calling `initExtension()` manually.
`cache_size`	Advanced option. A positive integer to change the maximum number of disk pages that SQLite holds in memory (SQLite's default is 2000 pages). See https://www.sqlite.org/pragma.html#pragma_cache_size for details.
`synchronous`	Advanced options. Possible values for `synchronous` are "off" (the default), "normal", or "full". Users have reported significant speed ups using `sychronous = "off"`, and the SQLite documentation itself implies considerable improved performance at the very modest risk of database corruption in the unlikely case of the operating system (*not* the R application) crashing. See https://www.sqlite.org/pragma.html#pragma_synchronous for details.
`flags`	`SQLITE_RWC`: open the database in read/write mode and create the database file if it does not already exist; `SQLITE_RW`: open the database in read/write mode. Raise an error if the file does not already exist; `SQLITE_RO`: open the database in read only mode. Raise an error if the file does not already exist
`vfs`	Select the SQLite3 OS interface. See https://www.sqlite.org/vfs.html for details. Allowed values are `"unix-posix"`, `"unix-unix-afp"`, `"unix-unix-flock"`, `"unix-dotfile"`, and `"unix-none"`.
`bigint`	The R type that 64-bit integer types should be mapped to, default is bit64::integer64, which allows the full range of 64 bit integers.
`extended_types`	When `TRUE` columns of type `DATE`, `DATETIME` / `TIMESTAMP`, and `TIME` are mapped to corresponding R-classes, c.f. below for details. Defaults to `FALSE`.

Details

Connections are automatically cleaned-up after they're deleted and reclaimed by the GC. You can use `DBI::dbDisconnect()` to terminate the connection early, but it will not actually close until all open result sets have been closed (and you'll get a warning message to this effect).

Value

`SQLite()` returns an object of class SQLiteDriver.

`dbConnect()` returns an object of class SQLiteConnection.

Extended Types

When parameter `extended_types = TRUE` date and time columns are directly mapped to corresponding R-types. How exactly depends on whether the actual value is a number or a string:

Column type	*Value is numeric*	*Value is Text*
DATE	Count of days since 1970-01-01	YMD formatted string (e.g. 2
TIME	Count of (fractional) seconds	HMS formatted string (e.g. 1
DATETIME / TIMESTAMP	Count of (fractional) seconds since midnight 1970-01-01 UTC	DATE and TIME as above se

If a value cannot be mapped an `NA` is returned in its place with a warning.

See Also

The corresponding generic functions `DBI::dbConnect()` and `DBI::dbDisconnect()`.

Examples

```
library(DBI)
# Initialize a temporary in memory database and copy a data.frame into it
con <- dbConnect(RSQLite::SQLite(), ":memory:")
data(USArrests)
dbWriteTable(con, "USArrests", USArrests)
dbListTables(con)

# Fetch all query results into a data frame:
dbGetQuery(con, "SELECT * FROM USArrests")

# Or do it in batches
rs <- dbSendQuery(con, "SELECT * FROM USArrests")
d1 <- dbFetch(rs, n = 10) # extract data in chunks of 10 rows
dbHasCompleted(rs)
d2 <- dbFetch(rs, n = -1) # extract all remaining data
dbHasCompleted(rs)
dbClearResult(rs)

# clean up
dbDisconnect(con)
```

`sqliteCopyDatabase` *Copy a SQLite database*

Description

Copies a database connection to a file or to another database connection. It can be used to save an in-memory database (created using `dbname = ":memory:"` or `dbname = "file::memory:"`) to a file or to create an in-memory database a copy of another database.

Usage

```
sqliteCopyDatabase(from, to)
```

Arguments

`from`	A `SQLiteConnection` object. The main database in `from` will be copied to `to`.
`to`	A `SQLiteConnection` object pointing to an empty database.

Author(s)

Seth Falcon

References

https://www.sqlite.org/backup.html

Examples

```
library(DBI)
# Copy the built in databaseDb() to an in-memory database
con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbListTables(con)

db <- RSQLite::datasetsDb()
RSQLite::sqliteCopyDatabase(db, con)
dbDisconnect(db)
dbListTables(con)

dbDisconnect(con)
```

sqliteHasHttpVFS *Check for HTTP VFS support*

Description

Capability query for optional HTTP VFS support.

Usage

```
sqliteHasHttpVFS()
```

sqliteHttpConfig *Configure experimental HTTP VFS behavior (optional)*

Description

lifecycle::badge("experimental")

Set conservative, process-wide options for the experimental HTTP/HTTPS VFS. These values are read on first open of a remote database in the current process.

Usage

```
sqliteHttpConfig(
  cache_size_mb = NULL,
  prefetch_pages = NULL,
  fallback_full_download = NULL
)
```

Arguments

`cache_size_mb`	Integer megabytes for in-memory page cache. `NULL` leaves unchanged.
`prefetch_pages`	Integer pages to prefetch ahead. `NULL` leaves unchanged.
`fallback_full_download`	Logical; if `TRUE`, allow full-download fallback when Range is not available. `NULL` leaves unchanged.

Details

Options are stored in environment variables so they also affect C-level code:

- `RSQLITE_HTTP_CACHE_MB`: In-memory page cache size in megabytes (default 4).
- `RSQLITE_HTTP_PREFETCH_PAGES`: Prefetch this many pages ahead (default 0).
- `RSQLITE_HTTP_FALLBACK_FULLDL`: If `TRUE` (default), fall back to full download when the server does not support HTTP Range; if `FALSE`, open will fail.

Value

A named list of previous values (in R types).

Examples

```
old <- sqliteHttpConfig(cache_size_mb = 8, prefetch_pages = 1, fallback_full_download = TRUE)
on.exit(do.call(sqliteHttpConfig, old), add = TRUE)
# ... connect with sqliteRemote() ...
```

---

sqliteHttpStats *Experimental HTTP VFS statistics*

Description

Returns counters collected for HTTP VFS operations in the current process. These values are best-effort and may currently report zeros until global aggregation is implemented.

Usage

```
sqliteHttpStats()
```

Details

Statistics are maintained per R process and reset when the process terminates. Values:

- `bytes_fetched`: Total bytes transferred via HTTP GET/Range requests.
- `range_requests`: Count of HTTP Range requests performed.
- `full_download`: Logical flag; `TRUE` if a fallback full download occurred for any open in this process.

If the HTTP VFS was not compiled in, all zeros (and `FALSE`) are returned.

Value

A list with elements `bytes_fetched`, `range_requests`, `full_download`.

See Also

`sqliteHttpConfig()`, `sqliteRemote()`, `sqliteHasHttpVFS()`

Examples

```
if (sqliteHasHttpVFS()) {
  # Hypothetical remote DB (replace with a real URL for actual use):
  # old <- sqliteHttpConfig(cache_size_mb = 8, prefetch_pages = 1)
  # on.exit(do.call(sqliteHttpConfig, old), add = TRUE)
  # con <- sqliteRemote("https://example.org/db.sqlite")
  # dbGetQuery(con, "SELECT name FROM sqlite_master WHERE type='table'")
  stats <- sqliteHttpStats()
  str(stats)
}
```

---

sqliteHttpVfsCompiled *Was HTTP VFS compiled into this build?*

Description

Returns TRUE if the experimental HTTP/HTTPS virtual file system was compiled in (libcurl detected at build time), FALSE otherwise. This is a compile-time indicator; even if TRUE the VFS might still fail to register at runtime on unusual platforms, in which case `sqliteHasHttpVFS()` is the definitive runtime capability probe.

Usage

```
sqliteHttpVfsCompiled()
```

Details

`sqliteHasHttpVFS()` performs a lazy registration attempt when the VFS is compiled in but not yet registered. If the HTTP VFS was not compiled in, calls to `sqliteRemote()` will error.

Value

A logical scalar.

Examples

```
sqliteHttpVfsCompiled()
```

sqliteRemote *Convenience helper for opening remote SQLite databases over HTTP/HTTPS*

Description

lifecycle::badge("experimental")

Constructs a URI filename and sets the appropriate VFS and immutable flags. Requires the optional http extension to be available; see `sqliteHasHttpVFS()`.

Usage

```
sqliteRemote(url, immutable = TRUE, ...)
```

Arguments

`url`	Remote HTTP/HTTPS URL to a SQLite database file.
`immutable`	Logical, append `immutable=1` to the URI.
`...`	Passed through to `DBI::dbConnect()`.

Value

A SQLiteConnection object.

Examples

```
if (sqliteHasHttpVFS()) {
  con <- sqliteRemote("https://example.org/db.sqlite")
  dbGetQuery(con, "SELECT name FROM sqlite_master WHERE type='table'")
  dbDisconnect(con)
}
```

sqliteSetBusyHandler *Configure what SQLite should do when the database is locked*

Description

When a transaction cannot lock the database, because it is already locked by another one, SQLite by default throws an error: `database is locked`. This behavior is usually not appropriate when concurrent access is needed, typically when multiple processes write to the same database.

`sqliteSetBusyHandler()` lets you set a timeout or a handler for these events. When setting a timeout, SQLite will try the transaction multiple times within this timeout. To set a timeout, pass an integer scalar to `sqliteSetBusyHandler()`.

Another way to set a timeout is to use a `PRAGMA`, e.g. the SQL query

```
PRAGMA busy_timeout=3000
```

sets the busy timeout to three seconds.

Usage

```
sqliteSetBusyHandler(dbObj, handler)
```

Arguments

dbObj	A SQLiteConnection object.
handler	Specifies what to do when the database is locked by another transaction. It can be: • NULL: fail immediately, • an integer scalar: this is a timeout in milliseconds that corresponds to PRAGMA busy_timeout, • an R function: this function is called with one argument, see details below.

Details

Note that SQLite currently does *not* schedule concurrent transactions fairly. If multiple transactions are waiting on the same database, any one of them can be granted access next. Moreover, SQLite does not currently ensure that access is granted as soon as the database is available. Make sure that you set the busy timeout to a high enough value for applications with high concurrency and many writes.

If the handler argument is a function, then it is used as a callback function. When the database is locked, this will be called with a single integer, which is the number of calls for same locking event. The callback function must return an integer scalar. If it returns 0L, then no additional attempts are made to access the database, and an error is thrown. Otherwise another attempt is made to access the database and the cycle repeats.

Handler callbacks are useful for debugging concurrent behavior, or to implement a more sophisticated busy algorithm. The latter is currently considered experimental in RSQLite. If the callback function fails, then RSQLite will print a warning, and the transaction is aborted with a "database is locked" error.

Note that every database connection has its own busy timeout or handler function.

Calling sqliteSetBusyHandler() on a connection that is not connected is an error.

Value

Invisible NULL.

See Also

https://www.sqlite.org/c3ref/busy_handler.html

Index

bit64::integer64, *12*

datasetsDb, 3
dbBegin,SQLiteConnection-method
(dbBegin_SQLiteConnection), 4
dbBegin_SQLiteConnection, 4
dbCommit,SQLiteConnection-method
(dbBegin_SQLiteConnection), 4
dbCommit_SQLiteConnection
(dbBegin_SQLiteConnection), 4
dbConnect(), *12*
dbConnect,SQLiteConnection-method
(SQLite), 11
dbConnect,SQLiteDriver-method (SQLite),
11
dbConnect_SQLiteConnection (SQLite), 11
dbConnect_SQLiteDriver (SQLite), 11
dbDisconnect,SQLiteConnection-method
(SQLite), 11
dbDisconnect_SQLiteConnection (SQLite),
11
DBI::dbBegin(), *4*
DBI::dbCommit(), *4*
DBI::dbConnect(), *4*, *6*, *7*, *13*, *18*
DBI::dbDataType(), *7*
DBI::dbDisconnect(), *13*
DBI::dbReadTable(), *6*
DBI::dbRollback(), *4*
DBI::dbSendQuery(), *11*
DBI::dbWithTransaction(), *4*
DBI::dbWriteTable(), *8*
DBI::make.db.names(), *8*
dbReadTable,SQLiteConnection,character-method
(dbReadTable_SQLiteConnection_character),
5
dbReadTable_SQLiteConnection_character,
5
dbRollback,SQLiteConnection-method
(dbBegin_SQLiteConnection), 4
dbRollback_SQLiteConnection
(dbBegin_SQLiteConnection), 4
dbWriteTable,SQLiteConnection,character,character-method
(dbWriteTable_SQLiteConnection_character_character),
6
dbWriteTable,SQLiteConnection,character,data.frame-method
(dbWriteTable_SQLiteConnection_character_character),
6
dbWriteTable_SQLiteConnection_character_character,
6
dbWriteTable_SQLiteConnection_character_data.frame
(dbWriteTable_SQLiteConnection_character_character),
6

initExtension, 9
initExtension(), *12*

read.table(), *8*
RSQLite (SQLite), 11
RSQLite-package (SQLite), 11
rsqliteVersion, 11

SQLite, 11
SQLite(), *12*
sqlite-transaction
(dbBegin_SQLiteConnection), 4
SQLITE_RO (SQLite), 11
SQLITE_RW (SQLite), 11
SQLITE_RWC (SQLite), 11
SQLiteConnection, *4*, *6*, *7*, *9*, *12*, *13*, *18*, *19*
sqliteCopyDatabase, 14
SQLiteDriver, *13*
sqliteHasHttpVFS, 15
sqliteHasHttpVFS(), *16*, *18*
sqliteHttpConfig, 15
sqliteHttpConfig(), *9*, *16*
sqliteHttpStats, 16
sqliteHttpVfsCompiled, 17
sqliteIsTransacting
(dbBegin_SQLiteConnection), 4

sqliteRemote, 18
sqliteRemote(), *9*, *16*, *17*
sqliteSetBusyHandler, 18