Package 'RSQLite'

April 22, 2021

Title 'SQLite' Interface for R

Version 2.2.7

Date 2021-04-22

Description Embeds the 'SQLite' database engine in R and
provides an interface compliant with the 'DBI' package. The source for
the 'SQLite' engine is included.

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 (>= 1.1.0), memoise, methods,
pkgconfig, Rcpp (>= 0.12.7)

Suggests DBItest (>= 1.7.0), gert, gh, knitr, rmarkdown, hms, rvest,
testthat, xml2

LinkingTo plogr (>= 0.2.0), Rcpp

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.1.1

Collate 'RcppExports.R' 'SQLiteConnection.R' 'SQLiteDriver.R'
'SQLiteResult.R' 'coerce.R' 'connect.R' 'copy.R' 'datasetsDb.R'
'deprecated.R' 'export.R' 'extensions.R' 'names.R'
'pkgconfig.R' 'query.R' 'regularExpressions.R' 'rownames.R'
'table.R' 'transactions.R' 'utils.R' 'zzz.R'

NeedsCompilation yes

Author Kirill Müller [aut, cre] (<https://orcid.org/0000-0002-1416-3412>),
Hadley Wickham [aut],
David A. James [aut],
Seth Falcon [aut],
SQLite Authors [ctb] (for the included SQLite sources),
Liam Healy [ctb] (for the included SQLite sources),
R Consortium [fnd],
RStudio [cph]

Maintainer Kirill Müller <krlmlr+r@mailbox.org>

Repository CRAN

Date/Publication 2021-04-22 15:30:06 UTC

R topics documented:

`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)
```

dbReadTable,SQLiteConnection,character-method
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,data.frame-method

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,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
)

## 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
)
```

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.
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.
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.
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()).
temporary	a logical specifying whether the new table should be temporary. 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.
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.

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

These extension functions are written by Liam Healy and made available through the SQLite website (https://www.sqlite.org/contrib).

Usage

```
initExtension(db)
```

Arguments

db	A SQLiteConnection object to load these extensions into.

Available extension functions

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()
RSQLite::initExtension(db)

dbGetQuery(db, "SELECT stdev(mpg) FROM mtcars")
sd(mtcars$mpg)
dbDisconnect(db)
```

initRegExp *Add regular expression operator*

Description

This loads a regular-expression matcher for posix extended regular expressions, as available through the SQLite source code repository (https://sqlite.org/src/raw?filename=ext/misc/regexp.c).

Usage

```
initRegExp(db)
```

Arguments

db	A SQLiteConnection object to add the regular expression operator into the connection.

Details

SQLite will then implement the "A regexp B" operator, where A is the string to be matched and B is the regular expression.

Note this only affects the specified connection.

Value

Always TRUE, invisibly.

Examples

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

dbGetQuery(db, "SELECT * FROM mtcars WHERE carb REGEXP '[12]'")
dbDisconnect(db)
```

rsqliteVersion *RSQLite version*

Description

RSQLite version

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 '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'
dbConnect(drv, ...)

## 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)
```

`sqlite-transaction` *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.

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)
```

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")
dbRollback(con, name = "b")
dbCommit(con, name = "a")
```

```
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)
```

`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