Package 'RSQLite'

January 8, 2017

Version 1.1-2

Date 2017-01-07

Title 'SQLite' Interface for R

Description Embeds the 'SQLite' database engine in R and provides an interface compliant with the 'DBI' package. The source for the 'SQLite' engine (version 3.8.8.2) is included.

Depends R (>= 3.1.0)

Suggests DBItest, knitr, rmarkdown, testthat

Imports DBI (>= 0.4-9), memoise, methods, Rcpp (>= 0.12.7)

LinkingTo Rcpp, BH, plogr

Encoding UTF-8

License LGPL (>= 2)

URL https://github.com/rstats-db/RSQLite

BugReports https://github.com/rstats-db/RSQLite/issues

Collate 'RcppExports.R' 'SQLiteConnection.R' 'SQLiteDriver.R' 'SQLiteResult.R' 'connect.R' 'copy.R' 'datasetsDb.R' 'deprecated.R' 'dummy.R' 'extensions.R' 'query.R' 'rownames.R' 'table.R' 'transactions.R' 'utils.R' 'zzz.R'

VignetteBuilder knitr

RoxygenNote 5.0.1.9000

NeedsCompilation yes

Author Kirill Müller [aut, cre],
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 [cph],
RStudio [cph]

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

Repository CRAN

Date/Publication 2017-01-08 16:57:09

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)
```

dbDataType,SQLiteDriver-method

Determine the SQL Data Type of an R object

Description

Given an object, return its SQL data type as a SQL database identifier.

Usage

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

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

Arguments

dbObj	a SQLiteConnection or SQLiteDriver object
obj	an R object whose SQL type we want to determine.
...	Needed for compatibility with generic. Otherwise ignored.

See Also

The corresponding generic function DBI::dbDataType().

Examples

```
library(DBI)
dbDataType(RSQLite::SQLite(), 1)
dbDataType(RSQLite::SQLite(), 1L)
dbDataType(RSQLite::SQLite(), "1")
dbDataType(RSQLite::SQLite(), TRUE)
dbDataType(RSQLite::SQLite(), list(raw(1)))

sapply(datasets::quakes, dbDataType, dbObj = RSQLite::SQLite())
```

dbExistsTable,SQLiteConnection,character-method

Tables in a database

Description

dbExistsTable() returns a logical that indicates if a table exists, dbListTables() lists all tables as a character vector.

Usage

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

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

Arguments

conn	An existing SQLiteConnection
name	String, name of table. Match is case insensitive.
...	Needed for compatibility with generics, otherwise ignored.

See Also

The corresponding generic functions DBI::dbExistsTable() and DBI::dbListTables().

Examples

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

dbExistsTable(db, "mtcars")
dbExistsTable(db, "nonexistingtable")
dbListTables(db)

dbDisconnect(db)
```

---

dbListFields,SQLiteConnection,character-method

List fields in a table

Description

Returns the fields of a given table as a character vector.

Usage

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

Arguments

conn	An existing SQLiteConnection
name	a length 1 character vector giving the name of a table.
...	Needed for compatibility with generic. Otherwise ignored.

See Also

The corresponding generic function DBI::dbListFields().

Examples

```
library(DBI)
db <- RSQLite::datasetsDb()
dbListFields(db, "iris")
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 = NA, check.names = TRUE, select.cols = "*")
```

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	A SQL expression (in the form of a character vector of length 1) giving the columns to select. E.g. "*" selects all columns, "x, y, z" selects three columns named as listed.

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)
dbReadTable(db, "mtcars", select.cols = "cyl, gear")
dbReadTable(db, "mtcars", select.cols = "row_names, cyl, gear")
dbDisconnect(db)
```

---

dbRemoveTable,SQLiteConnection,character-method

Remove a table from the database

Description

Executes the SQL `DROP TABLE`.

Usage

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

Arguments

conn	An existing SQLiteConnection
name	character vector of length 1 giving name of table to remove
...	Needed for compatibility with generic. Otherwise ignored.

See Also

The corresponding generic function DBI::dbRemoveTable().

Examples

```
library(DBI)
con <- dbConnect(RSQLite::SQLite())
dbWriteTable(con, "test", data.frame(a = 1))
dbListTables(con)
dbRemoveTable(con, "test")
dbListTables(con)
dbDisconnect(con)
```

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. `sqlData()` is mostly useful to backend implementers, but must be documented here.

Usage

```
## S4 method for signature 'SQLiteConnection,character,data.frame'
dbWriteTable(conn, name,
  value, ..., row.names = NA, 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)

## S4 method for signature 'SQLiteConnection'
sqlData(con, value, row.names = NA, ...)
```

Arguments

`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.
con, conn	a SQLiteConnection object, produced by DBI::dbConnect()

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 functions DBI::dbWriteTable() and DBI::sqlData().

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 (http://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)
```

---

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 sqlite-query for how to issue queries and receive results.

Usage

```
SQLite(...)

## S4 method for signature 'SQLiteDriver'
dbConnect(drv, dbname = "", ...,
  loadable.extensions = TRUE, cache_size = NULL, synchronous = "off",
  flags = SQLITE_RWC, vfs = NULL)

## 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.
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 http://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 http://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 http://www.sqlite.org/vfs.html for details. Allowed values are "unix-posix", "unix-unix-afp", "unix-unix-flock", "unix-dotfile", and "unix-none".

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).

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-meta *Result information*

Description

For a result object, returns information about the SQL statement used, the available columns and number of already fetched rows for a query, the number of affected rows for a statement, and the completion status.

Usage

```
## S4 method for signature 'SQLiteResult'
dbColumnInfo(res, ...)

## S4 method for signature 'SQLiteResult'
dbGetRowsAffected(res, ...)

## S4 method for signature 'SQLiteResult'
dbGetRowCount(res, ...)

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

## S4 method for signature 'SQLiteResult'
dbGetStatement(res, ...)
```

Arguments

res	An object of class SQLiteResult
...	Ignored. Needed for compatibility with generic

See Also

The corresponding generic functions DBI::dbColumnInfo(), DBI::dbGetRowsAffected(), DBI::dbGetRowCount(), DBI::dbHasCompleted(), and DBI::dbGetStatement().

Examples

```
library(DBI)
db <- RSQLite::datasetsDb()
rs <- dbSendQuery(db, "SELECT * FROM USArrests WHERE UrbanPop >= 80")
dbGetStatement(rs)
dbColumnInfo(rs)
dbHasCompleted(rs)
dbGetRowCount(rs)

dbFetch(rs, n = 2)
dbHasCompleted(rs)
```

```
dbGetRowCount(rs)

invisible(dbFetch(rs))
dbHasCompleted(rs)
dbGetRowCount(rs)
dbClearResult(rs)

dbDisconnect(db)

con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbExecute(con, "CREATE TABLE test (a INTEGER)")
rs <- dbSendStatement(con, "INSERT INTO test VALUES (:a)", list(a = 1:3))
dbGetRowsAffected(rs)
dbClearResult(rs)
dbDisconnect(con)
```

`sqlite-query` *Execute a SQL statement on a database connection*

Description

To retrieve results a chunk at a time, use `dbSendQuery()`, `dbFetch()`, then `dbClearResult()`. Alternatively, if you want all the results (and they'll fit in memory) use `dbGetQuery()` which sends, fetches and clears for you. To run the same prepared query with multiple inputs, use `dbBind()`. For statements that do not return a table, use `dbSendStatement()` and `dbExecute()` instead of `dbSendQuery()` and `dbGetQuery()`. See sqlite-meta for how to extract other metadata from the result set.

Usage

```
## S4 method for signature 'SQLiteConnection,character'
dbSendQuery(conn, statement,
  params = NULL, ...)

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

## S4 method for signature 'SQLiteResult'
dbFetch(res, n = -1, ..., row.names = NA)

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

Arguments

`conn`	an `SQLiteConnection` object.
`statement`	a character vector of length one specifying the SQL statement that should be executed. Only a single SQL statment should be provided.

params	A named list of query parameters to be substituted into a parameterised query. The elements of the list can be vectors which all must be of the same length.
...	Unused. Needed for compatibility with generic.
res	an SQLiteResult object.
n	maximum number of records to retrieve per fetch. Use -1 to retrieve all pending records; 0 retrieves only the table definition.
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.

See Also

The corresponding generic functions DBI::dbSendQuery(), DBI::dbFetch(), DBI::dbClearResult(), DBI::dbGetQuery(), DBI::dbBind(), DBI::dbSendStatement(), and DBI::dbExecute().

Examples

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

# Run query to get results as dataframe
dbGetQuery(db, "SELECT * FROM USArrests LIMIT 3")

# Send query to pull requests in batches
rs <- dbSendQuery(db, "SELECT * FROM USArrests")
dbFetch(rs, n = 2)
dbFetch(rs, n = 2)
dbHasCompleted(rs)
dbClearResult(rs)

# Parameterised queries are safest when you accept user input
dbGetQuery(db, "SELECT * FROM USArrests WHERE Murder < ?", list(3))

# Or create and then bind
rs <- dbSendQuery(db, "SELECT * FROM USArrests WHERE Murder < ?")
dbBind(rs, list(3))
dbFetch(rs)
dbClearResult(rs)

# Named parameters are a little more convenient
rs <- dbSendQuery(db, "SELECT * FROM USArrests WHERE Murder < :x")
dbBind(rs, list(x = 3))
dbFetch(rs)
dbClearResult(rs)
dbDisconnect(db)

# Passing multiple values is especially useful for statements
```

```
con <- dbConnect(RSQLite::SQLite())

dbWriteTable(con, "test", data.frame(a = 1L, b = 2L))
dbReadTable(con, "test")

dbExecute(con, "INSERT INTO test VALUES (:a, :b)",
          params = list(a = 2:4, b = 3:5))
dbReadTable(con, "test")

rs <- dbSendStatement(con, "DELETE FROM test WHERE a = :a AND b = :b")
dbBind(rs, list(a = 3:1, b = 2:4))
dbBind(rs, list(a = 4L, b = 5L))
dbClearResult(rs)
dbReadTable(con, "test")

# Multiple values passed to queries are executed one after another,
# the result appears as one data frame
dbGetQuery(con, "SELECT * FROM TEST WHERE a >= :a", list(a = 0:3))

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, ...)

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

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

Arguments

conn	a `SQLiteConnection` object, produced by `DBI::dbConnect()`
name	Supply a name to use a named savepoint. This allows you to nest multiple transaction
...	Needed for compatibility with generic. Otherwise ignored.

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, "a")
dbBegin(con, "b")
dbRollback(con, "b")
dbCommit(con, "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 anothe 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

http://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)
```

Index