Package 'RSQLite'

February 19, 2015

Version 1.0.0

Title SQLite Interface for R

Description This package embeds the SQLite database engine in R and provides an interface compliant with the DBI package. The source for the SQLite engine (version 3.8.6) is included.

Depends R (>= 2.10.0), DBI (>= 0.3.1), methods

Suggests testthat

License LGPL (>= 2)

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

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

Collate 'ConnectionExtensions.R' 'Connection.R' 'Driver.R' 'Connect.R' 'ConnectionRead.R' 'ConnectionTransactions.R' 'ConnectionWrite.R' 'Constants.R' 'Escaping.R' 'Result.R' 'Object.R' 'Summary.R' 'Utils.R' 'datasetsDb.R' 'dbGetInfo.R' 'extensions.R' 'zzz.R'

Author Hadley Wickham [aut, cre], David A. James [aut], Seth Falcon [aut], SQLite Authors [ctb] (for the included SQLite sources), Liam Healy [ctb] (for the include SQLite extensions), RStudio [cph]

Maintainer Hadley Wickham <hadley@rstudio.com>

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-10-25 01:58:48

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

```
db <- datasetsDb()
dbListTables(db)

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

dbDisconnect(db)
```

dbConnect,SQLiteDriver-method

Connect to/disconnect from a SQLite database.

Description

Connect to/disconnect from a SQLite database.

Usage

```
## 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

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. There are two special values: • "": creates a temporary on-disk database The file will be deleted when the connection is closed. • ":memory:": 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".

Examples

```
# Create temporary in-memory db
tmp <- dbConnect(SQLite(), ":memory:")
summary(tmp)
dbDisconnect(tmp)

# Create temporary on-disk db with bigger cache and safer synchronisation
tmp <- dbConnect(SQLite(), "", cache_size = 5000, synchronous = "full")
summary(tmp)
dbDisconnect(tmp)
```

---

dbDataType,SQLiteConnection-method

Determine the SQL Data Type of an R object.

Description

This method is a straight-forward implementation of the corresponding generic function.

Usage

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

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

Arguments

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

Examples

```
data(quakes)
drv <- SQLite()

sapply(quakes, function(x) dbDataType(drv, x))

dbDataType(drv, 1)
dbDataType(drv, as.integer(1))
dbDataType(drv, "1")
dbDataType(drv, charToRaw("1"))
```

dbExistsTable,SQLiteConnection,character-method *Does the table exist?*

Description

Does the table exist?

Usage

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

Arguments

conn	An existing SQLiteConnection
name	String, name of table. Match is case insensitive.

dbGetInfo *Get metadata about a database object.*

Description

Get metadata about a database object.

Usage

```
## S4 method for signature 'SQLiteDriver'
dbGetInfo(dbObj)

## S4 method for signature 'SQLiteConnection'
dbGetInfo(dbObj)

## S4 method for signature 'SQLiteResult'
dbGetInfo(dbObj)
```

Arguments

dbObj	An object of class SQLiteDriver, SQLiteConnection or SQLiteResult

Examples

```
dbGetInfo(SQLite())

con <- dbConnect(SQLite())
dbGetInfo(con)

dbWriteTable(con, "mtcars", mtcars)
rs <- dbSendQuery(con, "SELECT * FROM mtcars")
dbGetInfo(rs)
dbFetch(rs, 1)
dbGetInfo(rs)

dbClearResult(rs)
dbDisconnect(con)
```

---

dbIsValid *Check whether an SQLite object is valid or not.*

Description

Support function that verifies that the holding a reference to a foreign object is still valid for communicating with the RDBMS

Usage

```
## S4 method for signature 'SQLiteDriver'
dbIsValid(dbObj)

## S4 method for signature 'SQLiteConnection'
dbIsValid(dbObj)

## S4 method for signature 'SQLiteResult'
dbIsValid(dbObj)

isIdCurrent(obj)
```

Arguments

dbObj,obj	A driver, connection or result.

Value

A logical scalar.

Examples

```
dbIsValid(SQLite())

con <- dbConnect(SQLite())
dbIsValid(con)

dbDisconnect(con)
dbIsValid(con)
```

---

dbListFields,SQLiteConnection,character-method

List fields in specified table.

Description

List fields in specified table.

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.

Examples

```
con <- dbConnect(SQLite())
dbWriteTable(con, "iris", iris)
dbListFields(con, "iris")
dbDisconnect(con)
```

---

dbListTables,SQLiteConnection-method

List available SQLite tables.

Description

List available SQLite tables.

Usage

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

Arguments

conn	An existing SQLiteConnection

---

dbReadTable,SQLiteConnection,character-method
Convenience functions for importing/exporting DBMS tables

Description

These functions mimic their R/S-Plus counterpart get, assign, exists, remove, and objects, except that they generate code that gets remotely executed in a database engine.

Usage

```
## S4 method for signature 'SQLiteConnection,character'
dbReadTable(conn, name, row.names,
  check.names = TRUE, select.cols = "*")
```

Arguments

conn	a SQLiteConnection object, produced by 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.
row.names	A string or an index specifying the column in the DBMS table to use as row.names in the output data.frame. Defaults to using the row_names column if present. Set to NULL to never use row names.
check.names	If TRUE, the default, column names will be converted to valid R identifiers.
select.cols	A SQL statement (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.

Value

A data.frame in the case of dbReadTable; otherwise a logical indicating whether the operation was successful.

Note

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

Examples

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

# Supress row names
dbReadTable(con, "mtcars", row.names = FALSE)

dbDisconnect(con)
```

`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

---

`dbSendPreparedQuery` *Generics for getting and sending prepared queries.*

Description

Generics for getting and sending prepared queries.

Usage

```
dbSendPreparedQuery(conn, statement, bind.data, ...)

dbGetPreparedQuery(conn, statement, bind.data, ...)
```

Arguments

`conn`	An `DBIConnection` object.
`statement`	A SQL string
`bind.data`	A data frame
`...`	Other arguments used by methods

dbUnloadDriver,SQLiteDriver-method
Unload SQLite driver.

Description

Unload SQLite driver.

Usage

```
## S4 method for signature 'SQLiteDriver'
dbUnloadDriver(drv, ...)
```

Arguments

drv	Object created by SQLite
...	Ignored. Needed for compatibility with generic.

Value

A logical indicating whether the operation succeeded or not.

Examples

```
## Not run:
db <- SQLite()
dbUnloadDriver(db)

## End(Not run)
```

dbWriteTable,SQLiteConnection,character,data.frame-method
Write a local data frame or file to the database.

Description

Write a local data frame or file to the database.

Usage

```
## S4 method for signature 'SQLiteConnection,character,data.frame'
dbWriteTable(conn, name,
  value, row.names = NA, overwrite = FALSE, append = FALSE,
  field.types = NULL)

## 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)
```

Arguments

conn	a SQLiteConnection object, produced by 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.
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 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. (See the BUGS section below)
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 dbDataType).
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.

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 database to load these extensions.

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

```
db <- dbConnect(SQLite())
initExtension(db)

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

query *Execute a SQL statement on a database connection*

Description

To retrieve results a chunk at a time, use dbSendQuery, dbFetch, then ClearResult. Alternatively, if you want all the results (and they'll fit in memory) use dbGetQuery which sends, fetches and clears for you.

Usage

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

## S4 method for signature 'SQLiteConnection,character,data.frame'
dbSendPreparedQuery(conn,
  statement, bind.data)

## S4 method for signature 'SQLiteResult'
dbFetch(res, n = 0)

## S4 method for signature 'SQLiteResult'
fetch(res, n = 0)

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

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

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

## S4 method for signature 'SQLiteConnection,character'
dbGetQuery(conn, statement)

## S4 method for signature 'SQLiteConnection,character,data.frame'
dbGetPreparedQuery(conn,
  statement, bind.data)
```

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.
bind.data	A data frame of data to be bound.

res	an SQLiteResult object.
n	maximum number of records to retrieve per fetch. Use -1 to retrieve all pending records; use 0 for to fetch the default number of rows as defined in SQLite
...	Unused. Needed for compatibility with generic.

Examples

```
con <- dbConnect(SQLite(), ":memory:")
dbWriteTable(con, "arrests", datasets::USArrests)

# Run query to get results as dataframe
dbGetQuery(con, "SELECT * FROM arrests limit 3")

# Send query to pull requests in batches
res <- dbSendQuery(con, "SELECT * FROM arrests")
data <- fetch(res, n = 2)
data
dbHasCompleted(res)

dbListResults(con)
dbClearResult(res)

# Use dbSendPreparedQuery/dbGetPreparedQuery for "prepared" queries
dbGetPreparedQuery(con, "SELECT * FROM arrests WHERE Murder < ?",
   data.frame(x = 3))
dbGetPreparedQuery(con, "SELECT * FROM arrests WHERE Murder < (:x)",
   data.frame(x = 3))

dbDisconnect(con)
```

---

sqlite-meta *Database interface meta-data.*

Description

See documentation of generics for more details.

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

Examples

```
data(USArrests)
con <- dbConnect(SQLite(), dbname=":memory:")
dbWriteTable(con, "t1", USArrests)
dbWriteTable(con, "t2", USArrests)

dbListTables(con)

rs <- dbSendQuery(con, "select * from t1 where UrbanPop >= 80")
dbGetStatement(rs)
dbHasCompleted(rs)

info <- dbGetInfo(rs)
names(info)
info$fields

fetch(rs, n=2)
dbHasCompleted(rs)
info <- dbGetInfo(rs)
info$fields
dbClearResult(rs)

# DBIConnection info
names(dbGetInfo(con))

dbDisconnect(con)
```

SQLiteConnection-class

Class SQLiteConnection.

Description

SQLiteConnection objects are usually created by dbConnect

Examples

```
con <- dbConnect(SQLite(), dbname = tempfile())
dbDisconnect(con)
```

`sqliteCopyDatabase` *Copy a SQLite database*

Description

This function 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:"`) to a file or to create an in-memory database as 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`	Either a string specifying the file name where the copy will be written or a `SQLiteConnection` object pointing to an empty database. If `to` specifies an already existing file, it will be overwritten without a warning. When `to` is a database connection, it is assumed to point to an empty and unused database; the behavior is undefined otherwise.

Details

This function uses SQLite's experimental online backup API to make the copy.

Value

Returns `NULL`.

Author(s)

Seth Falcon

References

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

Examples

```
## Create an in memory database
db <- dbConnect(SQLite(), dbname = ":memory:")
df <- data.frame(letters=letters[1:4], numbers=1:4, stringsAsFactors = FALSE)
ok <- dbWriteTable(db, "table1", df, row.names = FALSE)
stopifnot(ok)

## Copy the contents of the in memory database to
## the specified file
backupDbFile <- tempfile()
sqliteCopyDatabase(db, backupDbFile)
diskdb <- dbConnect(SQLite(), dbname = backupDbFile)
stopifnot(identical(df, dbReadTable(diskdb, "table1")))

## Copy from one connection to another
db2 <- dbConnect(SQLite(), dbname = ":memory:")
sqliteCopyDatabase(db, db2)
stopifnot(identical(df, dbReadTable(db2, "table1")))

## cleanup
dbDisconnect(db)
dbDisconnect(diskdb)
dbDisconnect(db2)
unlink(backupDbFile)
```

`SQLiteDriver-class` *Class SQLiteDriver with constructor SQLite.*

Description

An SQLite driver implementing the R/S-Plus database (DBI) API. This class should always be initializes with the `SQLite()` function. It returns a singleton object that allows you to connect to the SQLite engine embedded in R.

Usage

```
SQLite(max.con = 200L, fetch.default.rec = 500, force.reload = FALSE,
  shared.cache = FALSE)
```

Arguments

`max.con,force.reload`	Ignored and deprecated.
`fetch.default.rec`	default number of records to fetch at one time from the database. The `fetch` method will use this number as a default, but individual calls can override it.
`shared.cache`	logical describing whether shared-cache mode should be enabled on the SQLite driver. The default is `FALSE`.

Details

This implementation allows the R embedded SQLite engine to work with multiple database instances through multiple connections simultaneously.

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.

Value

An object of class `SQLiteDriver` which extends `dbDriver` and `dbObjectId`. This object is needed to create connections to the embedded SQLite database. There can be many SQLite database instances running simultaneously.

Examples

```
# initialize a new database to a tempfile and copy some data.frame
# from the base package into it
con <- dbConnect(SQLite(), ":memory:")
data(USArrests)
dbWriteTable(con, "USArrests", USArrests)

# query
rs <- dbSendQuery(con, "select * from USArrests")
d1 <- fetch(rs, n = 10)      # extract data in chunks of 10 rows
dbHasCompleted(rs)
d2 <- fetch(rs, n = -1)      # extract all remaining data
dbHasCompleted(rs)
dbClearResult(rs)
dbListTables(con)

# clean up
dbDisconnect(con)
```

`sqliteQuickColumn` *Return an entire column from a SQLite database*

Description

Return an entire column from a table in a SQLite database as an R vector of the appropriate type. This function is experimental and subject to change.

Usage

```
sqliteQuickColumn(con, table, column)
```

Arguments

`con`	a `SQLiteConnection` object as produced by `sqliteNewConnection`.
`table`	a string specifying the name of the table
`column`	a string specifying the name of the column in the specified table to retrieve.

Details

This function relies upon the SQLite internal `ROWID` column to determine the number of rows in the table. This may not work depending on the table schema definition and pattern of update.

Value

an R vector of the appropriate type (based on the type of the column in the database).

Author(s)

Seth Falcon

SQLiteResult-class *Class SQLiteResult*

Description

SQLite's query results class. This classes encapsulates the result of an SQL statement (either `select` or not).

---

transactions *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.

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 `dbConnect`
`name`	Supply a name to use a named savepoint. This allows you to nest multiple transaction

Value

A boolean, indicating success or failure.

Examples

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

dbBegin(con)
rs <- dbSendQuery(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 <- dbSendQuery(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)
```

Index