

# Package ‘RSQLite’

October 4, 2009

**Version** 0.7-3

**Title** SQLite interface for R

**Author** David A. James

**Maintainer** Seth Falcon <seth@userprimary.net>

**Description** Database Interface R driver for SQLite. 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 3.6.4) is included.

**LazyLoad** yes

**Depends** R (>= 2.6.0), methods, DBI (>= 0.2-3)

**Imports** methods, DBI (>= 0.2-3)

**License** LGPL (>= 2)

**Collate** zzz.R S4R.R dbObjectId.R SQLite.R SQLiteSupport.R

**Repository** CRAN

**Date/Publication** 2009-10-04 15:43:27

## R topics documented:

dbBuildTableDefinition . . . . .	2
dbCallProc-methods . . . . .	3
dbCommit-methods . . . . .	3
dbConnect-methods . . . . .	4
dbDataType-methods . . . . .	5
dbDriver-methods . . . . .	6
dbGetInfo-methods . . . . .	7
dbListTables-methods . . . . .	8
dbObjectId-class . . . . .	9
dbReadTable-methods . . . . .	10
dbSendQuery-methods . . . . .	11

dbSetDataMappings-methods . . . . .	12
fetch-methods . . . . .	13
isIdCurrent . . . . .	14
make.db.names-methods . . . . .	15
SQLite . . . . .	16
SQLiteConnection-class . . . . .	18
SQLiteDriver-class . . . . .	20
SQLiteObject-class . . . . .	21
sqliteQuickColumn . . . . .	22
SQLiteResult-class . . . . .	22
sqliteSupport . . . . .	24
summary-methods . . . . .	27

<b>Index</b>	<b>28</b>
--------------	-----------

---

dbBuildTableDefinition

*Build the SQL CREATE TABLE definition as a string*

---

## Description

Build the SQL CREATE TABLE definition as a string for the input data.frame

## Usage

```
dbBuildTableDefinition(dbObj, name, value, field.types = NULL, row.names = TRUE, ..
```

## Arguments

dbObj	any DBI object (used only to dispatch according to the engine (e.g., MySQL, SQLite, Oracle))
name	name of the new SQL table
value	data.frame for which we want to create a table
field.types	optional named list of the types for each field in value
row.names	logical, should row.name of value be exported as a row_names field? Default is TRUE
...	reserved for future use

## Details

The output SQL statement is a simple CREATE TABLE with suitable for dbGetQuery

## Value

An SQL string

**References**

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[SQLite](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbReadTable](#).

---

dbCallProc-methods *Call an SQL stored procedure*

---

**Description**

Not yet implemented.

**Methods**

**conn** a `SQLiteConnection` object.

... additional arguments are passed to the implementing method.

**References**

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[SQLite](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbReadTable](#).

---

dbCommit-methods *DBMS Transaction Management*

---

**Description**

By default, `SQLite` is in auto-commit mode. `dbBeginTransaction` starts a `SQLite` transaction and turns auto-commit off. `dbCommit` and `dbRollback` commit and rollback the transaction, respectively and turn auto-commit on.

**Methods**

**conn** a `SQLiteConnection` object, as produced by the function `dbConnect`.

... any database-specific arguments.

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

## See Also

`SQLite`, `dbConnect`, `dbSendQuery`, `dbGetQuery`, `fetch`, `dbCommit`, `dbGetInfo`, `dbReadTable`.

## Examples

```
drv <- dbDriver("SQLite")
tfile <- tempfile()
con <- dbConnect(drv, dbname = tfile)
data(USArrests)
dbWriteTable(con, "arrests", USArrests)
dbGetQuery(con, "select count(*) from arrests")[1, ]

dbBeginTransaction(con)
rs <- dbSendQuery(con, "DELETE from arrests WHERE Murder > 1")
do_commit <- if (dbGetInfo(rs)[["rowsAffected"]] > 40) FALSE else TRUE
dbClearResult(rs)
dbGetQuery(con, "select count(*) from arrests")[1, ]
if (!do_commit)
  dbRollback(con)
dbGetQuery(con, "select count(*) from arrests")[1, ]

dbBeginTransaction(con)
rs <- dbSendQuery(con, "DELETE from arrests WHERE Murder > 5")
dbClearResult(rs)
dbCommit(con)
dbGetQuery(con, "select count(*) from arrests")[1, ]

dbDisconnect(con)
```

---

dbConnect-methods *Create a connection object to an SQLite DBMS*

---

## Description

These methods are straight-forward implementations of the corresponding generic functions.

## Methods

**drv** an object of class `SQLiteDriver`, or the character string `"SQLite"` or an `SQLiteConnection`.

**conn** an `SQLiteConnection` object as produced by `dbConnect`.

... As of RSQLite 0.4-1 you may specify values for the two PRAGMAs `cache_size` and `synchronous` when initializing a new connection (this does not apply, obviously, to cloning an existing connection).

RSQLite defaults `synchronous` to 0 (or "OFF"), although SQLite's default as of 3.2.8 is 2 (FULL). Possible values for `synchronous` are 0, 1, or 2 or the corresponding strings "OFF", "NORMAL", or "FULL". Users have reported significant speed ups using `synchronous=0`, 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 the SQLite documentation for the full details of this PRAGMA.

`cache_size` can be a positive integer to change the maximum number of disk pages that SQLite holds in memory (SQLite's default is 2000 pages).

### Side Effects

A connection between R/S-Plus and the embeddable SQLite server is established. Note that since the SQLite is embedded in R/S-Plus, connections are not too resource hungry.

SQLite connections only require the file name where the SQLite database reside. For details see [SQLite](#).

### References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

### See Also

[SQLite](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbReadTable](#).

---

dbDataType-methods *Determine the SQL Data Type of an S object*

---

### Description

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

### Methods

**dbObj** a `SQLiteDriver` object, e.g., `ODBCDriver`, `OracleDriver`.

**obj** R/S-Plus object whose SQL type we want to determine.

... any other parameters that individual methods may need.

### References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[isSQLKeyword](#) [make.db.names](#)

**Examples**

```
data(quakes)
drv <- dbDriver("SQLite")
sapply(quakes, function(x) dbDataType(drv, x))

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

---

dbDriver-methods     *SQLite implementation of the Database Interface (DBI) classes and drivers*

---

**Description**

SQLite driver initialization and closing

**Methods**

**drvName** character name of the driver to instantiate.

**drv** an object that inherits from `SQLiteDriver` as created by `dbDriver`.

... any other arguments are passed to the driver `drvName`.

**References**

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[SQLite](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbListTables](#), [dbReadTable](#).

**Examples**

```
## Not run:
# create an SQLite instance for capacity of up to 25 simultaneous
# connections.
m <- dbDriver("SQLite", max.con = 25)

con <- dbConnect(m, dbname="sqlite.db")
rs <- dbSubmitQuery(con,
```

```
      "select * from HTTP_ACCESS where IP_ADDRESS = '127.0.0.1'")
df <- fetch(rs, n = 50)
df2 <- fetch(rs, n = -1)
dbClearResult(rs)

pcon <- dbConnect(p, "user", "password", "dbname")
dbListTables(pcon)
## End(Not run)
```

---

dbGetInfo-methods *Database interface meta-data*

---

## Description

These methods are straight-forward implementations of the corresponding generic functions.

## Methods

**dbObj** any object that implements some functionality in the R/S-Plus interface to databases (a driver, a connection or a result set).

**res** an `SQLiteResult`.

... currently not being used.

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

## See Also

[SQLite](#), [dbDriver](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbListTables](#), [dbReadTable](#).

## Examples

```
data(USArrests)
drv <- dbDriver("SQLite")
con <- dbConnect(drv, 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)

names(dbGetInfo(drv))

# DBIConnection info
names(dbGetInfo(con))

dbDisconnect(con)

```

---

dbListTables-methods

*List items from an SQLite DBMS and from objects*

---

### Description

These methods are straight-forward implementations of the corresponding generic functions.

### Methods

**drv** an SQLiteDriver.  
**conn** an SQLiteConnection.  
**name** a character string with the table name.  
**...** currently not used.

### References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

### See Also

[SQLite](#), [dbGetInfo](#), [dbColumnInfo](#), [dbDriver](#), [dbConnect](#), [dbSendQuery](#)

### Examples

```

## Not run:
drv <- dbDriver("SQLite")
# after working awhile...
for(con in dbListConnections(odbc)){
  dbGetStatement(dbListResults(con))
}
## End(Not run)

```

---

dbObjectId-class    *Class dbObjectId*

---

### Description

A helper (mixin) class to provide external references in an R/S-Plus portable way.

### Objects from the Class

A virtual Class: No objects may be created from it.

### Slots

**Id:** Object of class "integer" this is an integer vector holding an opaque reference into a C struct (may or may not be a C pointer, may or may not have length one).

### Methods

**coerce** signature(from = "dbObjectId", to = "integer"): ...  
**coerce** signature(from = "dbObjectId", to = "numeric"): ...  
**coerce** signature(from = "dbObjectId", to = "character"): ...  
**format** signature(x = "dbObjectId"): ...  
**print** signature(x = "dbObjectId"): ...  
**show** signature(object = "dbObjectId"): ...

### Note

A cleaner mechanism would use external references, but historically this class has existed mainly for R/S-Plus portability.

### Examples

```
sqlite <- dbDriver("SQLite")
con <- dbConnect(sqlite, ":memory:")
is(sqlite, "dbObjectId")    ## True
is(con, "dbObjectId")    ## True
isIdCurrent(con)    ## True
dbDisconnect(con)
isIdCurrent(con)    ## False
```

---

 dbReadTable-methods

*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.

## Value

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

## Methods

**conn** an `SQLiteConnection` database connection object.

**name** a character string specifying a table name.

**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** in the case of `dbReadTable`, this argument can be a string or an index specifying the column in the DBMS table to be used as `row.names` in the output `data.frame` (a `NULL`, `" "`, or `0` specifies that no column should be used as `row.names` in the output).

In the case of `dbWriteTable`, this argument should be 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/S-Plus identifier `"row.names"` maps to the DBMS (see `make.db.names`).

**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`.

**...** optional arguments.

When `dbWriteTable` is used to import data from a file, you may optionally specify `header=`, `row.names=`, `col.names=`, `sep=`, `eol=`, `field.types=`, and `skip=`.

`header` is a logical indicating whether the first data line (but see `skip`) has a header or not. If missing, its value is determined following `read.table` convention, namely, it is set to `TRUE` if and only if the first row has one fewer field than the number of columns.

`row.names` is a logical to specify whether the first column is a set of row names. If missing its default follows the `read.table` convention.

`col.names` a character vector with column names (these names will be filtered with `make.db.names` to ensure valid SQL identifiers. (See also `field.types` below.)

The field separator `sep=` defaults to `' '`.

The end-of-line delimiter `eol` defaults to `'\n'`.

`skip` specifies number of lines to skip before reading the data and it defaults to 0.

`field.types` is a list of named field SQL types where names (`field.types`) provide the new table's column names (if missing, field types are inferred using `dbDataType`).

## BUGS

These RSQLite methods do not use transactions, thus it is dangerous to specify `overwrite=TRUE` in `dbWriteTable` (the table is first removed and in case the data exporting fails the original table is lost forever).

## Note

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

SQLite table names are *not* case sensitive, e.g., table names `ABC` and `abc` are considered equal.

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

## See Also

[sqliteImportFile](#), [SQLite](#), [dbDriver](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbListTables](#), [dbReadTable](#).

## Examples

```
## Not run:
conn <- dbConnect("SQLite", dbname = "sqlite.db")
if(dbExistsTable(conn, "fuel_frame")){
  dbRemoveTable(conn, "fuel_frame")
  dbWriteTable(conn, "fuel_frame", fuel.frame)
}
if(dbExistsTable(conn, "RESULTS")){
  dbWriteTable(conn, "RESULTS", results2000, append = T)
} else
  dbWriteTable(conn, "RESULTS", results2000)
}
## End(Not run)
```

---

dbSendQuery-methods

*Execute a SQL statement on a database connection*

---

## Description

These are the primary methods for interacting with a database via SQL queries.

**Methods**

**conn** a 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.

**res** a SQLiteResult object.

... additional parameters.

**References**

See the Database Interface definition document DBI.pdf in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[SQLite](#), [dbDriver](#), [dbConnect](#), [fetch](#), [dbCommit](#), [dbGetInfo](#), [dbReadTable](#).

**Examples**

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

res <- dbSendQuery(con, "SELECT * from arrests")
data <- fetch(res, n = 2)
data
dbClearResult(res)

dbGetQuery(con, "SELECT * from arrests limit 3")

tryCatch(dbGetQuery(con, "SELECT * FROM tableDoesNotExist"),
         error=function(e) { print("caught") })
dbGetException(con)
```

---

dbSetDataMappings-methods

*Set data mappings between SQLite and R/S-Plus*

---

**Description**

Not yet implemented

**Methods**

**res** a SQLiteResult object as returned by dbSendQuery.

**flds** a data.frame with field descriptions as returned by dbColumnInfo.

... any additional arguments are passed to the implementing method.

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

## See Also

`SQLite`, `dbSendQuery`, `fetch`, `dbColumnInfo`.

## Examples

```
## Not run:
makeImage <- function(x) {
  .C("make_Image", as.integer(x), length(x))
}

res <- dbSendQuery(con, statement)
flds <- dbColumnInfo(res)
flds[3, "Sclass"] <- makeImage

dbSetDataMappings(rs, flds)

im <- fetch(rs, n = -1)
## End(Not run)
```

---

fetch-methods

*Fetch records from a previously executed query*

---

## Description

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

## Details

The `RSQLite` implementations retrieves all records into a buffer internally managed by the `RSQLite` driver (thus this memory is not managed by R but its part of the R process), and `fetch` simply returns records from this internal buffer.

## Methods

**res** an `RSQLiteResult` object.

**n** maximum number of records to retrieve per fetch. Use `n = -1` to retrieve all pending records; use a value of `n = 0` for fetching the default number of rows `fetch.default.rec` defined in the `RSQLite` initialization invocation.

... currently not used.

## References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[SQLite](#), [dbConnect](#), [dbSendQuery](#), [dbGetQuery](#), [dbClearResult](#), [dbCommit](#), [dbGetInfo](#), [dbReadTable](#).

**Examples**

```
drv <- dbDriver("SQLite")
tfile <- tempfile()
con <- dbConnect(drv, dbname = tfile)
data(USJudgeRatings)
dbWriteTable(con, "jratings", USJudgeRatings)

res <- dbSendQuery(con, statement = paste(
  "SELECT row_names, ORAL, DILG, FAMI",
  "FROM jratings"))

# we now fetch the first 10 records from the resultSet into a data.frame
data1 <- fetch(res, n = 10)
dim(data1)

dbHasCompleted(res)

# let's get all remaining records
data2 <- fetch(res, n = -1)

dbClearResult(res)
dbDisconnect(con)
```

---

isIdCurrent

*Check whether an dbObjectId handle object is valid or not*


---

**Description**

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

**Usage**

```
isIdCurrent(obj)
```

**Arguments**

obj                    any dbObjectId (e.g., dbDriver, dbConnection, dbResult).

**Details**

dbObjectId are R/S-Plus remote references to foreign (C code) objects. This introduces differences to the object's semantics such as persistence (e.g., connections may be closed unexpectedly), thus this function provides a minimal verification to ensure that the foreign object being referenced can be contacted.

**Value**

a logical scalar.

**See Also**

[dbDriver](#) [dbConnect](#) [dbSendQuery](#) [dbGetQuery](#) [fetch](#)

**Examples**

```
## Not run:
cursor <- dbSendQuery(con, sql.statement)
isIdCurrent(cursor)
## End (Not run)
```

---

```
make.db.names-methods
```

*Make R/S-Plus identifiers into legal SQL identifiers*

---

**Description**

These methods are straight-forward implementations of the corresponding generic functions.

**Methods**

**dbObj** any SQLite object (e.g., `SQLiteDriver`).

**snames** a character vector of R/S-Plus identifiers (symbols) from which we need to make SQL identifiers.

**name** a character vector of SQL identifiers we want to check against keywords from the DBMS.

**unique** logical describing whether the resulting set of SQL names should be unique. Its default is `TRUE`. Following the SQL 92 standard, uniqueness of SQL identifiers is determined regardless of whether letters are upper or lower case.

**allow.keywords** logical describing whether SQL keywords should be allowed in the resulting set of SQL names. Its default is `TRUE`

**keywords** a character vector with SQL keywords, namely `.SQL92Keywords` defined in the DBI package.

**case** a character string specifying whether to make the comparison as lower case, upper case, or any of the two. it defaults to `any`.

... currently not used.

**References**

The set of SQL keywords is stored in the character vector `.SQL92Keywords` and reflects the SQL ANSI/ISO standard as documented in "X/Open SQL and RDA", 1994, ISBN 1-872630-68-8. Users can easily override or update this vector.

SQLite does not add keywords to the SQL 92 standard.

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

[SQLite](#), [dbReadTable](#), [dbWriteTable](#), [dbExistsTable](#), [dbRemoveTable](#), [dbListTables](#).

**Examples**

```
## Not run:
# This example shows how we could export a bunch of data.frames
# into tables on a remote database.

con <- dbConnect("SQLite", dbname = "sqlite.db")

export <- c("trantime.email", "trantime.print", "round.trip.time.email")
tabs <- make.db.names(export, unique = T, allow.keywords = T)

for(i in seq(along = export) )
  dbWriteTable(con, name = tabs[i], get(export[i]))
## End(Not run)
```

---

 SQLite

---

*Instantiate the SQLite engine from the current R session.*


---

**Description**

This function creates and initializes the SQLite engine. It returns an object that allows you to connect to the SQLite engine embedded in R.

**Usage**

```
SQLite(max.con = 16, fetch.default.rec = 500, force.reload = FALSE,
        shared.cache=FALSE)
```

**Arguments**

<code>max.con</code>	maximum number of connections that may be open at one time. This can be up to 100, a limit defined at compilation time. Note that since the SQLite engine is embedded (i.e., a set of C functions within the R process) connections consume very little resources.
<code>fetch.default.rec</code>	number of records to fetch at one time from the database. (The <code>fetch</code> method uses this number as a default.)
<code>force.reload</code>	should the package code be reloaded (reinitialized)? Setting this to <code>TRUE</code> allows you to change default settings. Notice that all connections should be closed before re-loading.
<code>shared.cache</code>	logical describing whether shared-cache mode should be enabled on the SQLite driver. The default is <code>FALSE</code> .

**Details**

This object is a singleton, that is, on subsequent invocations it returns the same initialized object.

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 required to create connections to the embedded SQLite database. There can be many SQLite database instances running simultaneously.

**Side Effects**

The R client part of the database communication is initialized, but note that connecting to database instances needs to be done through calls to `dbConnect`.

**User authentication**

SQLite is a single-user database engine, so no authentication is required.

**References**

See the Omega Project for Statistical Computing <http://stat.bell-labs.com/RS-DBI> for more details on the R database interface.

See the Adobe PDF file `DBI.pdf` under the `doc` subdirectory of the DBI package, i.e., `system.file("doc", "DBI.pdf", package = "DBI")`

See the documentation at the SQLite Web site <http://www.sqlite.org> for details.

**Author(s)**

David A. James

**See Also**

On database drivers:

`dbDriver`, `dbUnloadDriver`, `dbListConnections`.

On connections, SQL statements and resultSets:

`dbConnect`, `dbDisconnect`, `dbSendQuery`, `dbGetQuery`, `fetch`, `dbListResults`.

On transaction management:

`dbCommit`, `dbRollback`.

On meta-data:

`summary`, `dbGetInfo`, `dbListTables`, `dbListFields`, `dbColumnInfo`, `dbGetException`, `dbGetStatement`, `dbHasCompleted`, `dbGetRowCount`, `dbGetRowsAffected`.

**Examples**

```

# create a SQLite instance and create one connection.
m <- dbDriver("SQLite")

# initialize a new database to a tempfile and copy some data.frame
# from the base package into it
tfile <- tempfile()
con <- dbConnect(m, dbname = tfile)
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)
file.info(tfile)
file.remove(tfile)

```

---

SQLiteConnection-class

*Class SQLiteConnection*

---

**Description**

SQLite connection class.

**Generators**

The method [dbConnect](#) is the main generator.

**Extends**

Class "DBIConnection", directly. Class "SQLiteObject", directly. Class "DBIObject", by class "DBIConnection". Class "dbObjectId", by class "SQLiteObject".

**Methods**

**coerce** signature(from = "SQLiteConnection", to = "SQLiteResult"):...

**dbCallProc** signature(conn = "SQLiteConnection"):...

**dbCommit** signature(conn = "SQLiteConnection"):...

**dbConnect** signature(drv = "SQLiteConnection"):...

```

dbDisconnect signature(conn = "SQLiteConnection"):...
dbExistsTable signature(conn = "SQLiteConnection", name = "character"):
  ...
dbGetException signature(conn = "SQLiteConnection"):...
dbGetInfo signature(dbObj = "SQLiteConnection"):...
dbGetQuery signature(conn = "SQLiteConnection", statement = "character"):
  ...
dbListFields signature(conn = "SQLiteConnection", name = "character"):
  ...
dbListResults signature(conn = "SQLiteConnection"):...
dbListTables signature(conn = "SQLiteConnection"):...
dbReadTable signature(conn = "SQLiteConnection", name = "character"):
  ...
dbRemoveTable signature(conn = "SQLiteConnection", name = "character"):
  ...
dbRollback signature(conn = "SQLiteConnection"):...
dbSendQuery signature(conn = "SQLiteConnection", statement = "character"):
  ...
dbWriteTable signature(conn = "SQLiteConnection", name = "character",
  value = "data.frame"):...
summary signature(object = "SQLiteConnection"):...

```

**Author(s)**

R-SIG-DB

**References**

See the Database Interface definition document DBI.pdf in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

DBI classes: [SQLiteObject-class](#) [SQLiteDriver-class](#) [SQLiteConnection-class](#) [SQLiteResult-class](#)

**Examples**

```

drv <- dbDriver("SQLite")
tfile <- tempfile()
con <- dbConnect(drv, dbname = tfile)
dbDisconnect(con)
dbUnloadDriver(drv)

```

---

SQLiteDriver-class *Class SQLiteDriver*

---

### Description

An SQLite driver implementing the R/S-Plus database (DBI) API.

### Generators

The main generators are `dbDriver` and `SQLite`.

### Extends

Class "DBIDriver", directly. Class "SQLiteObject", directly. Class "DBIObject", by class "DBIDriver". Class "dbObjectId", by class "SQLiteObject".

### Methods

**coerce** signature(from = "SQLiteObject", to = "SQLiteDriver"):...

**dbConnect** signature(drv = "SQLiteDriver"):...

**dbGetInfo** signature(dbObj = "SQLiteDriver"):...

**dbListConnections** signature(drv = "SQLiteDriver"):...

**dbUnloadDriver** signature(drv = "SQLiteDriver"):...

**summary** signature(object = "SQLiteDriver"):...

### Author(s)

R-SIG-DB

### References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

### See Also

DBI classes: `SQLiteObject-class` `SQLiteDriver-class` `SQLiteConnection-class` `SQLiteResult-class`

### Examples

```
## Not run:
drv <- dbDriver("SQLite")
con <- dbConnect(drv, dbname="path/to/dbfile")
## End(Not run)
```

---

SQLiteObject-class *Class SQLiteObject*

---

### Description

Base class for all SQLite-specific DBI classes

### Objects from the Class

A virtual Class: No objects may be created from it.

### Extends

Class "DBIObject", directly. Class "dbObjectId", directly.

### Methods

**coerce** signature(from = "SQLiteObject", to = "SQLiteDriver"): ...  
**dbDataType** signature(dbObj = "SQLiteObject"): ...  
**isSQLKeyword** signature(dbObj = "SQLiteObject", name = "character"): ...  
**make.db.names** signature(dbObj = "SQLiteObject", snames = "character"): ...  
...  
**SQLKeywords** signature(dbObj = "SQLiteObject"): ...

### Author(s)

R-SIG-DB

### References

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

### See Also

DBI classes: [SQLiteObject-class](#) [SQLiteDriver-class](#) [SQLiteConnection-class](#)  
[SQLiteResult-class](#)

### Examples

```
## Not run:  
drv <- dbDriver("SQLite")  
con <- dbConnect(drv, dbname = "rsdbi.db")  
## End(Not run)
```

`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

<code>con</code>	a <code>SQLiteConnection</code> object as produced by <code>sqliteNewConnection</code> .
<code>table</code>	a string specifying the name of the table
<code>column</code>	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 class encapsulates the result of an SQL statement (either `select` or not).

### Generators

The main generator is `dbSendQuery`.

**Extends**

Class "DBIResult", directly. Class "SQLiteObject", directly. Class "DBIObject", by class "DBIResult". Class "dbObjectId", by class "SQLiteObject".

**Methods**

**coerce** signature(from = "SQLiteConnection", to = "SQLiteResult"): ...  
**dbClearResult** signature(res = "SQLiteResult"): ...  
**dbColumnInfo** signature(res = "SQLiteResult"): ...  
**dbGetException** signature(conn = "SQLiteResult"): ...  
**dbGetInfo** signature(dbObj = "SQLiteResult"): ...  
**dbGetRowCount** signature(res = "SQLiteResult"): ...  
**dbGetRowsAffected** signature(res = "SQLiteResult"): ...  
**dbGetStatement** signature(res = "SQLiteResult"): ...  
**dbHasCompleted** signature(res = "SQLiteResult"): ...  
**dbListFields** signature(conn = "SQLiteResult", name = "missing"): ...  
**fetch** signature(res = "SQLiteResult", n = "numeric"): ...  
**fetch** signature(res = "SQLiteResult", n = "missing"): ...  
**summary** signature(object = "SQLiteResult"): ...

**Author(s)**

R-SIG-DB

**References**

See the Database Interface definition document `DBI.pdf` in the base directory of this package or <http://stat.bell-labs.com/RS-DBI>.

**See Also**

DBI classes: [SQLiteObject-class](#) [SQLiteDriver-class](#) [SQLiteConnection-class](#) [SQLiteResult-class](#)

**Description**

These functions are the workhorses behind the RSQLite package, but users need not invoke these directly.

**Usage**

```
## SQLiteDriver-related
sqliteInitDriver(max.con=16, fetch.default.rec = 500, force.reload=FALSE,
                 shared.cache=FALSE)
sqliteDriverInfo(obj, what, ...)
sqliteDescribeDriver(obj, verbose = FALSE, ...)
sqliteCloseDriver(drv, ...)

## SQLiteConnection-related
sqliteNewConnection(drv, dbname, loadable.extensions=FALSE, cache_size=NULL,
                   synchronous=0)
sqliteConnectionInfo(obj, what, ...)
sqliteDescribeConnection(obj, verbose = FALSE, ...)
sqliteCloseConnection(con, ...)

## SQLiteResult-related
sqliteExecStatement(con, statement, bind.data=NULL)
sqliteTransactionStatement(con, statement)
sqliteFetch(res, n=0, ...)
sqliteQuickSQL(con, statement, bind.data=NULL, ...)
sqliteResultInfo(obj, what, ...)
sqliteDescribeResult(obj, verbose = FALSE, ...)
sqliteCloseResult(res, ...)

## data mappings and convenience functions
sqliteDataType(obj, ...)
sqliteReadTable(con, name, row.names = "row_names", check.names = TRUE, ...)
sqliteImportFile(con, name, value, field.types, overwrite=FALSE,
                append=FALSE, header, row.names, nrows=50, sep=",", eol="\n",
                skip = 0, ...)
sqliteWriteTable(con, name, value, row.names = TRUE,
                 overwrite = FALSE, append = FALSE,
                 field.types = NULL, ...)
sqliteTableFields(con, name, ...)
```

**Arguments**

`max.con` positive integer specifying maximum number of open connections. The default is 10. Note that since SQLite is embedded in R/S-Plus connections are simple,

	very efficient direct C calls.
<code>fetch.default.rec</code>	default number of rows to fetch (move to R/S-Plus). This default is used in <code>sqliteFetch</code> . The default is 500.
<code>force.reload</code>	logical indicating whether to re-initialize the driver. This may be useful if you want to change the defaults (e.g., <code>fetch.default.rec</code> ). Note that the driver is a singleton (subsequent inits just returned the previously initialized driver, thus this argument).
<code>obj</code>	any of the SQLite DBI objects (e.g., <code>SQLiteConnection</code> , <code>SQLiteResult</code> ).
<code>what</code>	character vector of metadata to extract, e.g., "version", "statement", "isSelect".
<code>verbose</code>	logical controlling how much information to display. Defaults to FALSE.
<code>drv</code>	an <code>SQLiteDriver</code> object as produced by <code>sqliteInit</code> .
<code>con</code>	an <code>SQLiteConnection</code> object as produced by <code>sqliteNewConnection</code> .
<code>res</code>	an <code>SQLiteResult</code> object as produced by <code>sqliteExecStatement</code> .
<code>dbname</code>	character string with the SQLite database file name (SQLite, like Microsoft's Access, stores an entire database in one file).
<code>loadable.extensions</code>	logical describing whether loadable extensions will be enabled for this connection. The default is FALSE.
<code>shared.cache</code>	logical describing whether shared-cache mode should be enabled on the SQLite driver. The default is FALSE.
<code>bind.data</code>	a data frame which will be used to bind variables in the statement.
<code>cache_size</code>	positive integer to pass to the <code>PRAGMA cache_size</code> ; this changes the maximum number of disk pages that SQLite will hold in memory (SQLite's default is 2000 pages).
<code>synchronous</code>	values the <code>PRAGMA synchronous</code> flag, possible values are 0, 1, or 2 or the corresponding strings "OFF", "NORMAL", or "FULL". The <code>RSQLite</code> package uses a default of 0 (OFF), although SQLite's default is 2 (FULL) as of version 3.2.8. Users have reported significant speed ups using <code>synchronous="OFF"</code> , and the SQLite documentation itself implies considerably improved performance at the very modest risk of database corruption in the unlikely case of the operating system ( <i>not</i> the R application) crashing.
<code>force</code>	logical indicating whether to close a connection that has open result sets. The default is FALSE.
<code>statement</code>	character string holding SQL statements.
<code>n</code>	number of rows to fetch from the given result set. A value of -1 indicates to retrieve all the rows. The default of 0 specifies to extract whatever the <code>fetch.default.rec</code> was specified during driver initialization <code>sqliteInit</code> .
<code>name</code>	character vector of names (table names, fields, keywords).
<code>value</code>	a <code>data.frame</code> .
<code>field.types</code>	a list specifying the mapping from R/S-Plus fields in the <code>data.frame</code> <code>value</code> to SQL data types. The default is <code>sapply(value, SQLDataType)</code> , see <code>SQLiteSQLType</code> .

<code>row.names</code>	a logical specifying whether to prepend the <code>data.frame</code> row names or not. The default is <code>TRUE</code> .
<code>check.names</code>	a logical specifying whether to convert DBMS field names into legal S names. Default is <code>TRUE</code> .
<code>overwrite</code>	logical indicating whether to replace the table name with the contents of the <code>data.frame</code> value. The defaults is <code>FALSE</code> .
<code>append</code>	logical indicating whether to append value to the existing table name.
<code>header</code>	logical, does the input file have a header line? Default is the same heuristic used by <code>read.table</code> , i.e., <code>TRUE</code> if the first line has one fewer column than the second line.
<code>nrows</code>	number of lines to rows to import using <code>read.table</code> from the input file to create the proper table definition. Default is 50.
<code>sep</code>	field separator character.
<code>eol</code>	end-of-line separator.
<code>skip</code>	number of lines to skip before reading data in the input file.
<code>...</code>	placeholder for future use.

### Value

`sqliteInitDriver` returns an `SQLiteDriver` object.

`sqliteDriverInfo` returns a list of name-value metadata pairs.

`sqliteDescribeDriver` returns `NULL` (displays the object's metadata).

`sqliteCloseDriver` returns a logical indicating whether the operation succeeded or not.

`sqliteNewConnection` returns an `SQLiteConnection` object.

`sqliteConnectionInfo` returns a list of name-value metadata pairs.

`sqliteDescribeConnection` returns `NULL` (displays the object's metadata).

`sqliteCloseConnection` returns a logical indicating whether the operation succeeded or not.

`sqliteExecStatement` returns an `SQLiteResult` object.

`sqliteFetch` returns a `data.frame`.

`sqliteQuickSQL` returns either a `data.frame` if the statement is a `select`-like or `NULL` otherwise.

`sqliteDescribeResult` returns `NULL` (displays the object's metadata).

`sqliteCloseResult` returns a logical indicating whether the operation succeeded or not.

`sqliteReadTable` returns a `data.frame` with the contents of the DBMS table.

`sqliteWriteTable` returns a logical indicating whether the operation succeeded or not.

`sqliteImportFile` returns a logical indicating whether the operation succeeded or not.

`sqliteTableFields` returns a character vector with the table name field names.

`sqliteDataType` returns a character string with the closest SQL data type. Note that SQLite is typeless, so this is mostly for creating table that are compatible across RDBMS.

`sqliteResultInfo` returns a list of name-value metadata pairs.

**Constants**

`.SQLitePkgName` (currently "RSQLite"), `.SQLitePkgVersion` (the R package version),  
`.SQLitePkgRCS` (the RCS revision), `.SQLitecle.NA.string` (character that SQLite uses  
to denote NULL on input), `.conflicts.OK`.

---

summary-methods      *Summarize an SQLite object*

---

**Description**

These methods are straight-forward implementations of the corresponding generic functions.

**Methods**

**object = "DBIObject"** Provides relevant metadata information on `object`, for instance, the SQLite  
server file, the SQL statement associated with a result set, etc.

**from** `object` to be coerced

**to** coercion class

**x** `object` to format or print or show

# Index

## \*Topic **classes**

- dbObjectId-class, 9
- SQLiteConnection-class, 18
- SQLiteDriver-class, 20
- SQLiteObject-class, 21
- SQLiteResult-class, 22

## \*Topic **database**

- dbBuildTableDefinition, 2
- dbCallProc-methods, 3
- dbCommit-methods, 3
- dbConnect-methods, 4
- dbDataType-methods, 5
- dbDriver-methods, 6
- dbGetInfo-methods, 7
- dbListTables-methods, 8
- dbReadTable-methods, 10
- dbSendQuery-methods, 11
- dbSetDataMappings-methods, 12
- fetch-methods, 13
- isIdCurrent, 14
- make.db.names-methods, 15
- SQLite, 16
- SQLiteConnection-class, 18
- SQLiteDriver-class, 20
- SQLiteObject-class, 21
- SQLiteResult-class, 22
- sqliteSupport, 24
- summary-methods, 27

## \*Topic **datasets**

- sqliteSupport, 24

## \*Topic **interface**

- dbBuildTableDefinition, 2
- dbCallProc-methods, 3
- dbCommit-methods, 3
- dbConnect-methods, 4
- dbDataType-methods, 5
- dbDriver-methods, 6
- dbGetInfo-methods, 7
- dbListTables-methods, 8

- dbReadTable-methods, 10
- dbSendQuery-methods, 11
- dbSetDataMappings-methods, 12
- fetch-methods, 13
- isIdCurrent, 14
- make.db.names-methods, 15
- SQLite, 16
- SQLiteConnection-class, 18
- SQLiteDriver-class, 20
- SQLiteObject-class, 21
- sqliteQuickColumn, 22
- SQLiteResult-class, 22
- sqliteSupport, 24
- summary-methods, 27

## \*Topic **methods**

- dbBuildTableDefinition, 2
- dbCallProc-methods, 3
- dbCommit-methods, 3
- dbConnect-methods, 4
- dbDataType-methods, 5
- dbDriver-methods, 6
- dbGetInfo-methods, 7
- dbListTables-methods, 8
- dbReadTable-methods, 10
- dbSendQuery-methods, 11
- dbSetDataMappings-methods, 12
- fetch-methods, 13
- make.db.names-methods, 15
- summary-methods, 27
- .SQLite.NA.string
  - (*sqliteSupport*), 24
- .SQLitePkgName(*sqliteSupport*), 24
- .SQLitePkgRCS(*sqliteSupport*), 24
- .SQLitePkgVersion
  - (*sqliteSupport*), 24
- .conflicts.OK(*sqliteSupport*), 24
- coerce, 9, 18, 20, 21, 23
- coerce, dbObjectId, character-method
  - (*summary-methods*), 27

- coerce, dbObjectId, integer-method  
(*summary-methods*), 27
- coerce, dbObjectId, numeric-method  
(*summary-methods*), 27
- coerce, SQLiteConnection, SQLiteDriver-method  
(*summary-methods*), 27
- coerce, SQLiteResult, SQLiteConnection-method  
(*summary-methods*), 27
- coerce-methods (*summary-methods*),  
27
  
- dbBeginTransaction  
(*dbCommit-methods*), 3
- dbBeginTransaction, SQLiteConnection-method  
(*dbCommit-methods*), 3
- dbBeginTransaction-methods  
(*dbCommit-methods*), 3
- dbBuildTableDefinition, 2
- dbCallProc, 18
- dbCallProc, SQLiteConnection-method  
(*dbCallProc-methods*), 3
- dbCallProc-methods, 3
- dbClearResult, 14, 23
- dbClearResult, SQLiteResult-method  
(*dbSendQuery-methods*), 11
- dbColumnInfo, 8, 12, 13, 17, 23
- dbColumnInfo, SQLiteResult-method  
(*dbGetInfo-methods*), 7
- dbColumnInfo-methods  
(*dbGetInfo-methods*), 7
- dbCommit, 2, 3, 5–7, 11, 12, 14, 17, 18
- dbCommit, SQLiteConnection-method  
(*dbCommit-methods*), 3
- dbCommit-methods, 3
- dbConnect, 2, 3, 5–8, 11, 12, 14, 15, 17, 18,  
20
- dbConnect, character-method  
(*dbConnect-methods*), 4
- dbConnect, SQLiteConnection-method  
(*dbConnect-methods*), 4
- dbConnect, SQLiteDriver-method  
(*dbConnect-methods*), 4
- dbConnect-methods, 4
- dbDataType, 10, 21
- dbDataType, SQLiteObject-method  
(*dbDataType-methods*), 5
- dbDataType-methods, 5
- dbDisconnect, 17, 19
- dbDisconnect, SQLiteConnection-method  
(*dbConnect-methods*), 4
- dbDisconnect-methods  
(*dbConnect-methods*), 4
- dbDriver, 7, 8, 11, 12, 15, 17, 20
- dbDriver, character-method  
(*dbDriver-methods*), 6
- dbDriver-methods, 6
- dbExistsTable, 16, 19
- dbExistsTable, SQLiteConnection, character-method  
(*dbReadTable-methods*), 10
- dbExistsTable-methods  
(*dbReadTable-methods*), 10
- dbGetDBIVersion-methods  
(*dbGetInfo-methods*), 7
- dbGetException, 17, 19, 23
- dbGetException, SQLiteConnection-method  
(*dbSendQuery-methods*), 11
- dbGetException-methods  
(*dbSendQuery-methods*), 11
- dbGetInfo, 2, 3, 5–8, 11, 12, 14, 17, 19, 20,  
23
- dbGetInfo (*dbGetInfo-methods*), 7
- dbGetInfo, SQLiteConnection-method  
(*dbGetInfo-methods*), 7
- dbGetInfo, SQLiteDriver-method  
(*dbGetInfo-methods*), 7
- dbGetInfo, SQLiteObject-method  
(*dbGetInfo-methods*), 7
- dbGetInfo, SQLiteResult-method  
(*dbGetInfo-methods*), 7
- dbGetInfo-methods, 7
- dbGetPreparedQuery  
(*dbSendQuery-methods*), 11
- dbGetPreparedQuery, SQLiteConnection, character, da  
(*dbSendQuery-methods*), 11
- dbGetPreparedQuery-methods  
(*dbSendQuery-methods*), 11
- dbGetQuery, 2, 3, 5–7, 11, 14, 15, 17, 19
- dbGetQuery, SQLiteConnection, character-method  
(*dbSendQuery-methods*), 11
- dbGetQuery-methods  
(*dbSendQuery-methods*), 11
- dbGetRowCount, 17, 23
- dbGetRowCount, SQLiteResult-method  
(*dbGetInfo-methods*), 7
- dbGetRowCount-methods  
(*dbGetInfo-methods*), 7

- dbGetRowsAffected, [17, 23](#)
- dbGetRowsAffected, SQLiteResult-method
  - ([dbGetInfo-methods](#)), [7](#)
- dbGetRowsAffected-methods
  - ([dbGetInfo-methods](#)), [7](#)
- dbGetStatement, [17, 23](#)
- dbGetStatement, SQLiteResult-method
  - ([dbGetInfo-methods](#)), [7](#)
- dbGetStatement-methods
  - ([dbGetInfo-methods](#)), [7](#)
- dbHasCompleted, [17, 23](#)
- dbHasCompleted, SQLiteResult-method
  - ([dbGetInfo-methods](#)), [7](#)
- dbHasCompleted-methods
  - ([dbGetInfo-methods](#)), [7](#)
- dbListConnections, [17, 20](#)
- dbListConnections, SQLiteDriver-method
  - ([dbListTables-methods](#)), [8](#)
- dbListConnections-methods
  - ([dbListTables-methods](#)), [8](#)
- dbListFields, [17, 19, 23](#)
- dbListFields, SQLiteConnection, character-method
  - ([dbListTables-methods](#)), [8](#)
- dbListFields-methods
  - ([dbListTables-methods](#)), [8](#)
- dbListResults, [17, 19](#)
- dbListResults, SQLiteConnection-method
  - ([dbListTables-methods](#)), [8](#)
- dbListResults-methods
  - ([dbListTables-methods](#)), [8](#)
- dbListTables, [6, 7, 11, 16, 17, 19](#)
- dbListTables, SQLiteConnection-method
  - ([dbListTables-methods](#)), [8](#)
- dbListTables-methods, [8](#)
- dbObjectId-class, [9](#)
- dbReadTable, [2, 3, 5–7, 11, 12, 14, 16, 19](#)
- dbReadTable, SQLiteConnection, character-method
  - ([dbReadTable-methods](#)), [10](#)
- dbReadTable-methods, [10](#)
- dbRemoveTable, [16, 19](#)
- dbRemoveTable, SQLiteConnection, character-method
  - ([dbReadTable-methods](#)), [10](#)
- dbRemoveTable-methods
  - ([dbReadTable-methods](#)), [10](#)
- dbRollback, [17, 19](#)
- dbRollback, SQLiteConnection-method
  - ([dbCommit-methods](#)), [3](#)
- dbRollback-methods
  - ([dbCommit-methods](#)), [3](#)
- dbSendPreparedQuery
  - ([dbSendQuery-methods](#)), [11](#)
- dbSendPreparedQuery, SQLiteConnection, character, data.frame-method
  - ([dbSendQuery-methods](#)), [11](#)
- dbSendPreparedQuery-methods
  - ([dbSendQuery-methods](#)), [11](#)
- dbSendQuery, [2, 3, 5–8, 11, 13–15, 17, 19, 22](#)
- dbSendQuery, SQLiteConnection, character-method
  - ([dbSendQuery-methods](#)), [11](#)
- dbSendQuery-methods, [11](#)
- dbSetDataMappings, SQLiteResult, data.frame-method
  - ([dbSetDataMappings-methods](#)), [12](#)
- dbSetDataMappings-methods, [12](#)
- dbUnloadDriver, [17, 20](#)
- dbUnloadDriver, SQLiteDriver-method
  - ([dbDriver-methods](#)), [6](#)
- dbUnloadDriver-methods
  - ([dbDriver-methods](#)), [6](#)
- dbWriteTable, [16, 19](#)
- dbWriteTable, SQLiteConnection, character, character-method
  - ([dbReadTable-methods](#)), [10](#)
- dbWriteTable, SQLiteConnection, character, data.frame-method
  - ([dbReadTable-methods](#)), [10](#)
- dbWriteTable-methods
  - ([dbReadTable-methods](#)), [10](#)
- fetch, [2, 3, 5–7, 11–13, 15, 17, 23](#)
- fetch, SQLiteResult, numeric-method
  - ([fetch-methods](#)), [13](#)
- fetch, SQLiteResult-method
  - ([fetch-methods](#)), [13](#)
- fetch-methods, [13](#)
- format, [9](#)
- format, dbObjectId-method
  - ([summary-methods](#)), [27](#)
- format-methods ([summary-methods](#)), [27](#)
- isIdCurrent, [14](#)
- isSQLKeyword, [5, 21](#)
- isSQLKeyword, SQLiteObject, character-method
  - ([make.db.names-methods](#)), [15](#)
- isSQLKeyword-methods
  - ([make.db.names-methods](#)), [15](#)
- last.warning([sqliteSupport](#)), [24](#)

- make.db.names, 5, 10, 21
- make.db.names, SQLiteObject, character-method (*sqliteSupport*), 24
  - (*make.db.names-methods*), 15
- make.db.names-methods, 15
- print, 9
- print, dbObjectId-method
  - (*summary-methods*), 27
- read.table, 10
- show, 9
- show, dbObjectId-method
  - (*summary-methods*), 27
- show-methods (*summary-methods*), 27
- SQLite, 2, 3, 5–8, 11–14, 16, 16, 20
- sqliteCloseConnection
  - (*sqliteSupport*), 24
- sqliteCloseDriver
  - (*sqliteSupport*), 24
- sqliteCloseResult
  - (*sqliteSupport*), 24
- SQLiteConnection-class, 19–21, 23
- SQLiteConnection-class, 18
- sqliteConnectionInfo
  - (*sqliteSupport*), 24
- sqliteDataType (*sqliteSupport*), 24
- sqliteDescribeConnection
  - (*sqliteSupport*), 24
- sqliteDescribeDriver
  - (*sqliteSupport*), 24
- sqliteDescribeResult
  - (*sqliteSupport*), 24
- SQLiteDriver (*SQLite*), 16
- SQLiteDriver-class, 19–21, 23
- SQLiteDriver-class, 20
- sqliteDriverInfo (*sqliteSupport*),
  - 24
- sqliteExecStatement
  - (*sqliteSupport*), 24
- sqliteFetch (*sqliteSupport*), 24
- sqliteFetchOneColumn
  - (*sqliteSupport*), 24
- sqliteImportFile, 11
- sqliteImportFile (*sqliteSupport*),
  - 24
- sqliteInitDriver (*sqliteSupport*),
  - 24
- sqliteNewConnection
  - character-method (*sqliteSupport*), 24
- SQLiteObject-class, 19–21, 23
- SQLiteObject-class, 21
- sqliteQuickColumn, 22
- sqliteQuickSQL (*sqliteSupport*), 24
- sqliteReadTable (*sqliteSupport*),
  - 24
- SQLiteResult-class, 19–21, 23
- SQLiteResult-class, 22
- sqliteResultInfo (*sqliteSupport*),
  - 24
- sqliteSupport, 24
- sqliteTableFields
  - (*sqliteSupport*), 24
- sqliteTransactionStatement
  - (*sqliteSupport*), 24
- sqliteWriteTable (*sqliteSupport*),
  - 24
- SQLKeywords, 21
- SQLKeywords, missing-method
  - (*make.db.names-methods*), 15
- SQLKeywords, SQLiteObject-method
  - (*make.db.names-methods*), 15
- SQLKeywords-methods
  - (*make.db.names-methods*), 15
- summary, 17, 20, 23
- summary, SQLiteConnection-method
  - (*summary-methods*), 27
- summary, SQLiteDriver-method
  - (*summary-methods*), 27
- summary, SQLiteObject-method
  - (*summary-methods*), 27
- summary, SQLiteResult-method
  - (*summary-methods*), 27
- summary-methods, 27