

# Package ‘SQLiteDF’

May 26, 2009

**Type** Package

**Title** Stores data frames & matrices in SQLite tables

**Version** 0.1.34

**Date** 2007-07-10

**Author** Miguel A. R. Manese

**Maintainer** Miguel A. R. Manese <jjonphl@gmail.com>

**Description** Transparently stores data frames & matrices into SQLite tables.

**Depends** R (>= 2.4.0), DBI (>= 0.1-10), utils

**Suggests** biglm (>= 0.7)

**License** GPL-2

**Repository** CRAN

**Date/Publication** 2009-05-26 06:38:29

## R topics documented:

SQLiteDF-package . . . . .	2
attachSdf . . . . .	3
detachSdf . . . . .	4
dupSdf . . . . .	4
getSdf . . . . .	5
inameSdf . . . . .	6
lsSdf . . . . .	7
rbindSdf . . . . .	8
renameSdf . . . . .	9
sdfImportDBI . . . . .	9
sdfImportSQLite . . . . .	10
sdfm . . . . .	11
sdfm2 . . . . .	12

sdfSelect . . . . .	13
sqlite.data.frame . . . . .	14
sqlite.matrix . . . . .	17
sqlite.vector . . . . .	18
typeSvec . . . . .	19

<b>Index</b>	<b>20</b>
--------------	-----------

---

SQLiteDF-package    *SQLite data frames*

---

## Description

S3 and utility methods to implement SQLite data frames.

## Details

This package aims to transparently operate on data frames stored in SQLite databases. The following functions are known to be working with SQLiteDF types:\

### sqlite.vector

Math, Summary, Ops groups except for those with 2 args in Math like `round`, and `%%` and `%/%` in Ops

summary out of the box works for numeric, integer, character. Also works for logical, factor using "group by", however factor's do not handle NA's specially.

length Ok

\ [ not yet \ [<-

sort sorts to a new sqlite.vector

### sqlite.data.frame

length returns no. of variables

dimnames, names, row.names returns names of variables and rows

dim, nrow, ncol dimension of variables

\$/, [ [ returns columns as sqlite.vector. no assignments yet

[ works for numeric & integer indexing (ranges and vectors), logical indexes with recycling. does not support negative and character (row/column name) indexing.

is.list returns false, so that other functions can

rbind works only when rbind()-ing data frames with the same names()

with Ok

as.data.frame just returns itself

as.list returns a list of sqlite.vector for each variable

rbind currently works only with data.frame's

lapply, sapply out of the box

summary out of the box, after implementing sqlite.vector

**Author(s)**

Miguel A. R. Manese Maintainer: Miguel A. R. Manese <jjonphl@gmail.com>

---

`attachSdf`*Attach SDF Database*

---

**Description**

Attaches a SDF to the workspace by adding to the SDF list in `workspace.db` and then attaching the SDF file in the SQLite engine.

**Usage**

```
attachSdf(sdf_filename, sdf_iname = NULL)
```

**Arguments**

`sdf_filename` A string containing a path (recommended is relative path) to the SDF database file.

`sdf_iname` A string containing an internal name with which the SDF to be attached will be renamed. By default, the internal name stored in the SDF's attribute table will be used as the internal name. Duplicates will be resolved by appending numbers.

**Details**

Attaching a SDF into the SQLiteDF workspace involves adding an entry in `workspace.db` and then attaching to the SQLite engine. Checks are made to avoid attaching the same file twice. This is done by storing the full path name of the SDF file in `workspace.db`. When loading SQLiteDF, the relative path is used to locate the SDF files and then the full path are recalculated. When a SDF file is to be attached with `attachSdf`, its full path is calculated and then it is compared with those in the workspace. These restrictions are to avoid possibly subtle problems. (In retrospect, I can't imagine any serious problem yet but since it's there already... To think that I cringed for a week on this problem, I *am* an idiot.)

**Value**

Returns an `sqlite.data.frame` object of the attached SDF if successful. `NULL` with an error message displayed otherwise.

**Author(s)**

Miguel A. R. Manese

**See Also**

`sqlite.data.frame.attachSdf lsSdf`

---

`detachSdf`*Detach SDF*

---

**Description**

Detaches a SDF from the workspace by detaching from the SQLite engine and then deleting from the SDF list in `workspace.db`.

**Usage**

```
detachSdf (iname)
```

**Arguments**

`iname`            A string containing the internal name of the desired SDF.

**Value**

TRUE if the operation was successful. FALSE with an error message displayed otherwise.

**Author(s)**

Miguel A. R. Manese

**See Also**

[sqlite.data.frame attachSdf lsSdf](#)

---

`dupSdf`*Duplicate SDF*

---

**Description**

Duplicates an SDF.

**Usage**

```
dupSdf (sdf)
```

**Arguments**

`sdf`            The SDF object to be duplicated.

**Value**

An `sqlite.data.frame` that is a duplicate of the argument. The internal name will be autogenerated, i.e. the default `data<n>`.

**Author(s)**

Miguel A. R. Manese

**See Also**

[renameSdf](#)

---

getSdf

*Get an SDF*

---

**Description**

Gets a handle to an SDF registered in the workspace. The returned handle is a `sqlite.data.frame` object which can be used with the most common operators for data frames.

**Usage**

```
getSdf(name)
```

**Arguments**

`name` A string containing the internal name of the desired SDF. This SDF is attached to the SQLiteDF workspace if it is not yet attached.

**Details**

The SDF must be *registered* with the SQLiteDF workspace. To get a list of registered SDF, use `lsSdf`.

**Value**

Returns an `sqlite.data.frame` object for the SDF with the specified internal name.

**Author(s)**

Miguel A. R. Manese

**See Also**

[sqlite.data.frame lsSdf](#)

**Examples**

```
iris.sdf <- sqlite.data.frame(iris)
iris.sdf.iname <- inameSdf(iris.sdf)
iris.sdf.too <- getSdf(iris.sdf.iname)
```

---

inameSdf	<i>SDF Internal Name</i>
----------	--------------------------

---

### Description

Get the internal name of an SDF.

### Usage

```
inameSdf(sdf)
```

### Arguments

sdf                    a `sqlite.data.frame`

### Details

Internal names are those that are listed in `lsSdf`, not the names of the R variables holding a *reference* to SDFs. There could be many R variable that refers to the same SDF (when `inameSdf(sdf1) == inameSdf(sdf2)`).

### Value

A 2 element string vector with the internal name of the SDF and the SQLite database file name where it is stored.

### Author(s)

Miguel A. R. Manese

### See Also

[sqlite.data.frame lsSdf](#)

### Examples

```
iris.sdf1 <- sqlite.data.frame(iris)
inameSdf(iris.sdf1) # c("data1", "data1.db"), or generally data<n>
iris.sdf2 <- sqlite.data.frame(iris, "iris")
inameSdf(iris.sdf2) # c("iris", "iris.db"), or if it already exists, iris<n>
```

---

`lsSdf`*List SDF's*

---

**Description**

Lists the SDF's available in the workspace.

**Usage**

```
lsSdf(pattern = NULL)
```

**Arguments**

`pattern`          Pattern of the internal name to be searched. Currently not implemented.

**Details**

`lsSdf` works much like `ls` for examining the contents of R's workspace. It does this by querying the list of SDF's stored in `workspace.db` file. See [sqlite.data.frame](#) for more details on `workspace.db`.

**Value**

Returns in a character vector the internal names of SDF's *registered* with the SQLiteDF workspace. The internal names are sorted according to (1) those that are attached first and then (2) the usage score.

**Author(s)**

Miguel A. R. Manese

**See Also**

[sqlite.data.frame](#) [lsSdf](#) [attachSdf](#) [detachSdf](#) [renameSdf](#)

**Examples**

```
lsSdf()
```

---

`rbindSdf`*Add Rows to a SDF*

---

**Description**

Adds rows to a SQLite data frame.

**Usage**

```
rbindSdf(sdf, df)
```

**Arguments**

<code>sdf</code>	the SQLite data frame with which to insert new data.
<code>df</code>	the data frame to attach

**Details**

This is equivalent to performing an `insert` statement. The data frame names and the column types must match, although they may be not arranged exactly the same.

**Value**

Returns the 1st argument (`sdf`).

**Note**

Currently SDFs are not supported as the 2nd arg. This could be achieved by doing batch `rbindSdf` to chunks of another SDF.

**Author(s)**

Miguel A. R. Manese

**Examples**

```
b1 <- sqlite.data.frame(iris)
rbindSdf(b1, iris)
nrow(b1) # nrow(iris) * 2
```

---

renameSdf	<i>Rename SDF</i>
-----------	-------------------

---

**Description**

Changes the internal name of an SDF, both in the workspace and in the SDF's attributes table.

**Usage**

```
renameSdf(sdf, name)
```

**Arguments**

sdf	The SDF to be renamed.
name	The new internal name of the SDF.

**Details**

This simple operation have a quite complicated implementation. The intended name is checked against SDF's already registered to the workspace, the SDF is detached and removed from the SDF workspace, and then re-attached and re-added to the workspace.

**Value**

TRUE if the operation was successful. FALSE with an error message displayed otherwise.

**Author(s)**

Miguel A. R. Manese

**See Also**

[lsSdf](#)

---

sdfImportDBI	<i>Import Data to an SQLite Data Frame</i>
--------------	--

---

**Description**

Import data from a DBMS to a sqlite.data.frame using DBI.

**Usage**

```
sdfImportDBI(con, sql, batch.size = 2048, rownames = "row_names", iname = NULL)
```

**Arguments**

<code>con</code>	a DBI connection object
<code>sql</code>	a select statement to retrieve the data to be imported
<code>batch.size</code>	the number of rows to be fetched from the connection at a time
<code>rownames</code>	a string or index specifying the column in the result of the select statement to be used as the row.names of the SDF
<code>iname</code>	the internal name of the created SDF

**Details**

This function just sends the query to the connection (`DBI:::dbSendQuery`) and then loops over the result set of the select statement (`DBI:::fetch`).

**Value**

The SDF containing the imported data.

**Author(s)**

Miguel A. R. Manese

**See Also**

[dbSendQuery](#) [fetch](#) [dbReadTable](#)

**Examples**

```
## Not run:
conn <- dbConnect("MySQL", group = "vitalAnalysis")
fuel_frame.sdf <- sdfImportDBI(conn, "select * from fuel_frame", iname="fuel_frame")
## End(Not run)
```

---

`sdfImportSQLite`     *Import SQLite data to an SQLite Data Frame*

---

**Description**

Import from SQLite tables directly.

**Usage**

```
sdfImportSQLite(dbfilename, tablename, iname = tablename)
```

**Arguments**

<code>dbfilename</code>	file name of the SQLite database.
<code>tablename</code>	name of the table inside the SQLite database that will be imported
<code>iname</code>	the internal name of the created SDF

**Value**

The SDF containing the imported data.

**Note**

Row names is just the sequence from 1 to number of rows. Text columns are converted into factors. Blobs are not supported.

**Author(s)**

Miguel Angel R. Manese

**See Also**

[sdfImportDBI](#)

**Examples**

```
## Not run:
data.sdf <- sdfImportSQLite("data.db", "fuel_frame", iname="fuel_frame")
## End(Not run)
```

---

sdfm

---

*Linear Models On SQLite Data Frames*


---

**Description**

Currently, a convenience wrapper around biglm. In the future, biglm functionalities will be implemented directly inside the package.

**Usage**

```
sdfm(formula, sdf, batch.size = 1024)
```

**Arguments**

formula	model formula
sdf	the sqlite.data.frame
batch.size	number of rows at a time to be <i>fed</i> to biglm

**Details**

biglm can computes the linear model coefficients and statistics incrementally by *feeding* it with part of the data first at a time. sdfm just gets data from the SDF into a plain data frame, batch.size at a time, and then used with biglm. to

Currently, models using only those variables inside the SDF are supported (i.e. do not use variables that is not in the SDF, like those in the global environment).

**Value**

An object of class `biglm`.

**Author(s)**

Miguel A. R. Manese

**See Also**

[biglm](#)

**Examples**

```
library(biglm)
iris.sdf <- sqlite.data.frame(iris)
iris.lm <- sdflm(Sepal.Length ~ Species, iris.sdf)
summary(iris.lm)
```

---

sdflm2

*Linear Models on SQLite Data Frames*

---

**Description**

`biglm` specialized for SQLite Data Frames

**Usage**

```
sdflm2(x, y, intercept = TRUE)
```

**Arguments**

<code>x</code>	a SQLite data frame containing the design matrix which may not include the intercept
<code>y</code>	a SQLite vector containing the observed response
<code>intercept</code>	if TRUE, adds an intercept term when doing computation

**Details**

Algorithm is identical with `biglm`. The only difference is that the rows of `x` and the values of `y` are directly fed to the algorithm.

**Value**

Returns a subclass of `biglm`. `biglm` methods can be used with the output, e.g. compute coefficients, `vcov`, etc.

**Author(s)**

Miguel A. R. Manese

**References**

Algorithm AS274 Applied Statistics (1992) Vol.41, No. 2

**See Also**

[biglm](#)

**Examples**

```
library(biglm)
iris.sdf <- sqlite.data.frame(iris)
x <- iris.sdf[,1:3]
y <- iris.sdf[,4]
iris.biglm <- sdflm2(x, y)
summary(iris.biglm)
```

---

sdfSelect

*Directly Query an SQLite Data Frame*

---

**Description**

Directly query an SQLite Data Frame using a SELECT statement.

**Usage**

```
sdfSelect(sdf, select = NULL, where = NULL, limit = NULL, debug = FALSE)
```

**Arguments**

sdf	the <code>sqlite.data.frame</code>
select	content of the SELECT clause. If NULL, then "*" is assumed
where	content of the WHERE clause. If NULL, then an empty WHERE clause is assumed
limit	content of the LIMIT clause. This limits the rows returned
debug	if true, prints the SQL statement issued to SQLite

## Details

Issues a SELECT statement to the corresponding data table of the passed SDF. It forms the actual SELECT statement from the fragments supplied as arguments. This insulates the user from the underlying naming conventions and database organization used by the package. To do more sophisticated queries, please use RSQLite and open the databases in the `.SQLiteDF` directory under your working directory.

Use square brackets to quote column names that are not valid SQL object names. E.g. to select the `Petal.Length` column of a SDF copy of dataset `iris`, use `[Petal.Length]`.

The limit clause is a way to restrict the results of a query. Note that the limiting operation is done after the resultset has been determined, and in no way can effect any computation in the select clause. E.g. `select count(*) from sdf_data limit 4,10` will not return 10. It will return an empty set since the result of the SELECT statement without the LIMIT clause is a single row, and the LIMIT clause takes 10 rows starting from the 5th row in the result set.

## Value

Returns NULL if the query does not return any row, a vector of the appropriate class (for factors) if there is only one column selected, or a data frame if there are more than one columns.

## Author(s)

Miguel A. R. Manese

## References

~put references to the literature/web site here ~

## Examples

```
iris.sdf <- sqlite.data.frame(iris)
sdfSelect(iris.sdf, "[Petal.Length]", "[Petal.Length]>3")
sdfSelect(iris.sdf, where="[Petal.Length]>3", limit="9,5")
sdfSelect(iris.sdf, where="Species=3")
```

---

sqlite.data.frame *SQLite Data Frame*

---

## Description

Creates an SQLite Data Frame (SDF) from ordinary data frames.

## Usage

```
sqlite.data.frame(x, name=NULL)
```

## Arguments

<code>x</code>	The object to be coerced into a data frame which is then stored in a SQLite database. <code>as.data.frame</code> is called first on <code>x</code> before creating the SDF database.
<code>name</code>	The internal name of the SDF. If none is provided, a generic name <code>data&lt;n&gt;</code> is used (e.g. <code>data1</code> , <code>data2</code> , etc). Each SDF should have a unique internal name and also be a valid R symbol. Numbers are appended to names in case of duplicates, e.g. if name arg is <code>iris</code> , and it already exists, then the new SDF will have a name <code>iris1</code> . If it still exists, then the name will be <code>iris2</code> , and so on.

## Details

SQLite data frames (SDF's) are data frames whose data are stored in a SQLite database. SQLite is an open source, powerful (considering its size), light weight data base engine. It stores each database (composed of tables, indices, etc.) in a single file. Since a single SDF occupies a whole database, each SDF will be contained in a single file.

Each SDF file contains the following tables:

`sdf_attributes` a key-value table that contains the SDF attributes. Currently, only `name` is used representing the SDF's internal name.

`sdf_data` contains the actual data. Factors and ordered variables are stored as integers. Their levels are stored in other tables. Numeric (real) are stored as double, characters as text and integers as int's. Currently, complex numbers are not supported. Column names correspond exactly to the variable names of the ordinary data frames. E.g. `Petal.Length` will have a column name `Petal.Length` in the table. This is possible because SQLite allows almost any kind of column name as long as it is quoted by square brackets (`[ ]`). You're on your own if you try to be a smartass on this. Also, an extra column named `row name` (with the space between the words), of type text is used to store the data frame row names and is set as the table's primary key. So please don't use `row name` as a variable name.

`[factor <colname>` and `[ordered <colname>]` stores the levels and level labels for each factor variable in the SDF. One such table will be created for every factor or ordered var, even if two variables share the same level labels. Besides storing the level data, it is used to mark a column as being a factor.

SDF's are managed in a workspace separate from R's. When SQLiteDF is loaded, it searches for the file `workspace.db` inside the subdirectory `.SQLiteDF` in the current working directory. This file contains a list of SDF's created/used in the previous session (i.e. SQLiteDF sessions are automatically saved), including their full and relative path and attach information. Workspace is managed using the SQLite engine by opening `workspace.db` as the main database and then attaching (SQLite's `attach`) the SDF's. Unfortunately, the number of attached databases is limited to 31 (actually 32, but 1 is reserved for the temp db). Therefore, SDF's are *scored* according to the number of times it has been used. When the maximum allowed attachment is reached, the least used attached SDF's is detached and the needed one is attached in its place. On compiling the package, the configure script modifies the bundled SQLite source such that constant controlling the maximum attachments is modified to 31 (default is 10).

Back to when SQLiteDF is loaded, after opening `workspace.db`, the SDF's stored in the list are sorted according to their number of uses in the previous session and then the first 30 are attached. The relative path is used for finding the SQLite file. If the file cannot be found, it is deleted from the SQLiteDF workspace (with a warning message). The scores are then all reset.

A `sqlite.data.frame` object is a list a single element:

`iname` the internal name of the SDF.

and the following attributes:

```
class The S3 class vector c("sqlite.data.frame", "data.frame")
```

`row.names` A `sqlite.vector` of mode character containing the row names of the SDF

All SDF's created in the session will have their SQLite file stored in the subdirectory `.SQLiteDF` in the current working directory. SDF's created in the other session can be imported/attached to the current SDF workspace using `attachSdf`, which may reside anywhere in the file system.

### Value

A S3 object representing the SDF. The SDF database will be created in the same directory with file name derived by appending the extension `db` to the passed internal name, or the default internal name if none is provided.

### Note

The full path is used to avoid attaching the same db which may have different relative path after the user changes directory after loading `SQLiteDF` (see `attachSdf`).

### Author(s)

Miguel A. R. Manese

### See Also

[lsSdf](#) [getSdf](#) [attachSdf](#) [detachSdf](#)

### Examples

```
library(datasets)
iris.sdf <- sqlite.data.frame(iris)
names(iris.sdf)
class(iris.sdf)
iris.sdf$Petal.Length[1:10]
iris.sdf[["Petal.Length"]][1:10]
iris.sdf[1,c(TRUE,FALSE)]
#apply(iris.sdf[1:4], 2, mean)
```

---

sqlite.matrix	<i>SQLite Matrix</i>
---------------	----------------------

---

### Description

EXPERIMENTAL: Creates a SQLite Matrix (SMAT) from data frames, matrices and SQLite Data Frames.

### Usage

```
sqlite.matrix(data, name = NULL)
```

### Arguments

data	a data frame, matrix or SQLite Data Frame
name	the name of the SQLite Matrix

### Details

Creates an SDF with 1 column named V1. The *mode* of the matrix is determined from the data, much like `as.matrix` does. It is similar to `sqlite.vector` and has the additional attributes `sdf.dim` and `sdf.dimnames` comparable to `matrix`'s `dim` and `dimnames` attributes.

Internally, it has extra tables `sdf_matrix_rownames` which holds the row names (and implicitly the row count), and `sdf_matrix_colnames` which holds the column names (and implicitly the column count).

### Value

A S3 object representing the SQLite Matrix.

### Author(s)

Miguel A. R. Manese

### See Also

[sqlite.vector](#) [sqlite.data.frame](#)

### Examples

```
iris.sdf <- sqlite.data.frame(iris)
im <- sqlite.matrix(iris.sdf[,1:4])
dim(im) # c(150, 4)
```

---

sqlite.vector      *SQLite Vector*

---

### Description

Creates a SQLite Matrix (SVEC) from atomic vectors.

### Usage

```
sqlite.vector(vec, name = NULL)
```

### Arguments

<code>vec</code>	an atomic vector
<code>name</code>	name of the SQLite Data Frame created which will hold the vector

### Details

Creates an SDF with 1 column named V1 which will hold the data of the vector. The *mode* of the SQLite vector is determined from data.

### Value

A S3 object representing the SQLite Vector.

### Note

SQLite vectors are not limited to columns of the table `sdf_data`. For example, with SQLite matrices the row names are exposed as SQLite vectors on columns of a table other than `sdf_data`.

### Author(s)

Miguel A. R. Manese

### See Also

[sqlite.matrix](#) [sqlite.data.frame](#)

### Examples

```
data <- runif(30000)
summary(data)
```

---

typeSvec	<i>Get Type of sqlite.vector</i>
----------	----------------------------------

---

**Description**

Returns the *class* of an `sqlite.vector` if it were

**Usage**

```
typeSvec(x)
has.typeSvec(x, type)
```

**Arguments**

<code>x</code>	an <code>sqlite.vector</code>
<code>type</code>	a string containing the class name, like <code>numeric</code> , <code>factor</code> , <code>character</code>

**Value**

<code>typeSvec</code>	returns a string with the class name of the argument
<code>has.typeSvec</code>	returns <code>TRUE</code> if <code>x</code> is a <code>sqlite.vector</code> and has <code>type</code> as returned by <code>typeSvec</code>

**Author(s)**

Miguel A. R. Manese

**Examples**

```
iris.sdf <- sqlite.data.frame(iris)
typeSvec(iris.sdf[,1]) # numeric
has.typeSvec(iris.sdf[,1], "numeric") # TRUE
```

# Index

## \*Topic **classes**

- `sqlite.data.frame`, 14
- `sqlite.matrix`, 17
- `sqlite.vector`, 18
- `typeSvec`, 19

## \*Topic **database**

- `sdfImportDBI`, 9
- `sdfImportSQLite`, 10

## \*Topic **data**

- `attachSdf`, 3
- `detachSdf`, 4
- `dupSdf`, 4
- `getSdf`, 5
- `lsSdf`, 7
- `renameSdf`, 9
- `sdfSelect`, 13
- `sqlite.data.frame`, 14
- `sqlite.matrix`, 17
- `sqlite.vector`, 18

## \*Topic **manip**

- `rbindSdf`, 8
- `sdfSelect`, 13
- `sqlite.data.frame`, 14
- `sqlite.matrix`, 17
- `sqlite.vector`, 18

## \*Topic **methods**

- `inameSdf`, 6

## \*Topic **package**

- `SQLiteDF-package`, 1

## \*Topic **regression**

- `sdflm`, 11
- `sdflm2`, 12

`attachSdf`, 3, 3, 4, 7, 16

`biglm`, 12, 13

`dbReadTable`, 10

`dbSendQuery`, 10

`detachSdf`, 4, 7, 16

`dupSdf`, 4

`fetch`, 10

`getSdf`, 5, 16

`has.typeSvec` (`typeSvec`), 19

`inameSdf`, 6

`lsSdf`, 3–6, 7, 7, 9, 16

`rbindSdf`, 8

`renameSdf`, 5, 7, 9

`sdf` (`sqlite.data.frame`), 14

`sdfImportDBI`, 9, 11

`sdfImportSQLite`, 10

`sdflm`, 11

`sdflm2`, 12

`sdfSelect`, 13

`sqlite.data.frame`, 3–7, 14, 17, 18

`sqlite.matrix`, 17, 18

`sqlite.vector`, 17, 18

`SQLiteDF` (`SQLiteDF-package`), 1

`SQLiteDF-package`, 1

`typeSvec`, 19