Package 'poplite'

October 12, 2017

Version 0.99.19

Date 2017-10-12

Title Tools for Simplifying the Population and Querying of SQLite Databases

Depends R (>= 3.1), methods, dplyr (>= 0.7.3), DBI (>= 0.7)

Imports igraph, lazyeval, dbplyr, RSQLite (>= 2.0)

Suggests testthat, Lahman, VariantAnnotation

Description Provides objects and accompanying methods which facilitates populating and querying SQLite databases.

License GPL-3

URL https://github.com/dbottomly/poplite

Collate tableSchemaList.R Database.R external_funcs.R

NeedsCompilation no

Author Daniel Bottomly [cre, aut],
Shannon McWeeney [aut],
Beth Wilmot [aut]

Maintainer Daniel Bottomly <bottomly@ohsu.edu>

Repository CRAN

Date/Publication 2017-10-12 18:56:45 UTC

R topics documented:

clinical *Example sample tracking dataset*

Description

A synthetic sample tracking dataset consisting of a clinical table containing clinical information on a set given fictional patients (samples in this context). A samples table which contains addition information on the samples in the clinical dataset including whether they have had DNA collected from them. Finally there is a DNA table which indicates the quality of a given DNA specimen as well as the ID provided by the DNA isolation lab. See the vignette for more descriptions and an example of loading and querying the data using `poplite`.

Usage

```
data("clinical")
```

Format

The format is:

"clinical"	data.frame
"samples"	data.frame
"dna"	data.frame

Examples

```
data(clinical)
str(clinical)
str(samples)
str(dna)
```

Database-class *Class* "Database"

Description

An object representing an SQLite database containing both the location of the file as well as a `TableSchemaList` object describing the structure of the database.

Objects from the Class

Objects can be created by calls of the form `Database(tbsl, db.file)`.

Slots

`tbsl`: Object of class "TableSchemaList" representing the current or desired database schema

`db.file`: Single file path to the desired location of the database

`connection`: An object of class "SQLiteConnection"

Methods

isOpen `signature(obj = "Database")`: Return a logical value indicating if the database connection is open

columns `signature(obj = "Database")`: Returns a list of the database table columns indexed by table name

dbFile `signature(obj = "Database")`: Returns the file path associated with the database

populate `signature(obj = "Database")`, ..., use.tables = NULL, should.debug = FALSE: Populate an SQLite database using the schema and location from the `Database` object and the data to be inserted as specified in the `dta.func` element of the `TableSchemaList`. The `use.tables` argument can be used to limit the tables populated. The `should.debug` argument outputs more verbose messages regarding the SQL queries.

schema `signature(obj = "Database")`: Returns the associated `TableSchemaList` object

tables `signature(obj = "Database")`: Returns a character vector containing the table names

Author(s)

Daniel Bottomly

See Also

`TableSchemaList`

Examples

```
if (require(Lahman) && require(RSQLite))
{
 baseball.teams <- makeSchemaFromData(TeamsFranchises, name="team_franch")
 baseball.teams <- append(baseball.teams, makeSchemaFromData(Teams, name="teams"))

 relationship(baseball.teams, from="team_franch", to="teams") <- franchID ~ franchID

 baseball.db <- Database(baseball.teams, tempfile())

 tables(baseball.db)
 columns(baseball.db)
 schema(baseball.db)

 populate(baseball.db, teams=Teams, team_franch=TeamsFranchises)

 examp.con <- dbConnect(SQLite(), dbFile(baseball.db))
```

```
dbListTables(examp.con)

head(dbReadTable(examp.con, "teams"))
head(dbReadTable(examp.con, "team_franch"))

dbDisconnect(examp.con)

}
```

`External methods` *Specific methods for generics defined in external packages.*

Description

These functions provide convienient interfaces to functionality provided in external packages (currently only **dplyr**). See the vignette and below examples.

Usage

```
filter(.data, ...)
select(.data, ...,.tables=NULL)
```

Arguments

`.data`	A `Database` object.
`.tables`	A character vector indicating the table(s) the specified columns refer to.
`...`	For `filter`, a single valid R expression which would result in a logical vector upon execution. For `select`, expression indicating the columns to choose from the given table(s). See the examples in `dplyr::filter` and `dplyr::select`. In addition, the names of the tables can be prepended to each variable name similar to SQL statements (e.g. 'table.column').

Value

An object of class `tbl_sqlite`.

Author(s)

Daniel Bottomly

See Also

`filter`,`select`

Examples

```
if (require(Lahman))
{
    baseball.teams <- makeSchemaFromData(TeamsFranchises, name="team_franch")
    baseball.teams <- append(baseball.teams, makeSchemaFromData(Teams, name="teams"))

   relationship(baseball.teams, from="team_franch", to="teams") <- franchID ~ franchID

    baseball.db <- Database(baseball.teams, tempfile())

    populate(baseball.db, teams=Teams, team_franch=TeamsFranchises)

    select(baseball.db, .tables="teams")

    select(baseball.db, .tables=c("teams", "team_franch"))

    select(baseball.db, yearID:WCWin, franchName)

    filter(baseball.db, active == "Y")

  select(filter(baseball.db, active == "Y" & W > 50 & teamID == "CAL"), active, W, teamID)
}
```

Helper Functions *Functions to facilitate the creation of* poplite*'s data structures.*

Description

These functions facilitate the creation of TableSchemaList objects from existing or supplementary R data structures such as the data.frame.

Usage

```
makeSchemaFromData(tab.df, name=NULL,dta.func=NULL)
makeSchemaFromFunction(dta.func, name,...)
correct.df.names(tab.df)
```

Arguments

tab.df	A data.frame representing a database table
name	Desired name of the database table
dta.func	An optional function which will take a specified object and turn it into a SQLite table. The function parameters should have the same names as the objects supplied to populate.
...	Arbitary objects provided to the function specified in dta.func. They should be of the same type and be named like the objects to be passed to populate.

Value

`makeSchemaFromData`
: A `TableSchemaList` object

`makeSchemaFromData`
: A `TableSchemaList` object

`correct.df.names`
: A `data.frame` with valid names for SQLite

Author(s)

Daniel Bottomly

See Also

`TableSchemaList`

Examples

```
if (require(Lahman))
{
  franches <- makeSchemaFromData(TeamsFranchises, name="team_franch")
  show(franches)

  makeSchemaFromFunction(function(x) head(x),  name="team_franch", x=TeamsFranchises)

  test.df <- TeamsFranchises
  names(test.df)[1] <- "franch.ID"

  names(test.df)

  names(correct.df.names(test.df))
}
```

TableSchemaList-class *Class* "TableSchemaList"

Description

A list-based representation of a SQLite database which provides a simple approach to loading data into a database as well as merging with the existing data. See the vignette for more complex examples.

Objects from the Class

Objects can be created by calls of the form `new("TableSchemaList", tab.list, search.cols)`.

Slots

`tab.list`**:** Object of class `"list"` A list of lists with each list representing a table and each element containing information on the definition of columns. There should be 6 elements to the list: `db.cols` a character vector containing the names of the columns `db.schema` a character vector of the same length as `db.cols` which contains the columns types (e.g. TEXT, INTEGER) `db.constr` a character string containing the statement at the end of a query indicating constraints `dta.func` a function which when applied to the input (usually a list) provides a `data.frame` to be inserted into the database. `should.ignore` a boolean value indicating whether duplicates implied by the constraints should be ignored upon insertion `foreign.keys` a list (or NULL) containing several elements named by each table to be joined. The two elements are `local.keys` which are the columns that should be kept from joining of the two tables and `ext.keys` which are the columns used in the joining.

Methods

length `signature(obj = "TableSchemaList")` Return the number of tables in the object

append `signature(obj = "TableSchemaList")`, x, values, after=length(x): Return a new `TableSchemaList` object consisting of `x`, the object to be modified, `values` the object(s) to be added and `after` the element of `x` to place them after.

columns `signature(obj = "TableSchemaList")` Returns a list of length equal to the number of tables where each element contains columns for the given table.

tables `signature(obj = "TableSchemaList")` Returns a vector of the table names in the object.

createTable `signature(obj = "TableSchemaList")`, table.name, mode=c("normal", "merge"): Produces a create table statement based on the table specified in `table.name` and whether the table should be temporary for merging purposes or normal permanent table

insertStatement `signature(obj = "TableSchemaList")`, table.name, mode=c("normal", "merge"): Produces an insert statement based on the table specified in `table.name` and whether the table should be temporary for merging purposes or normal permanent table. This insert statement will be used in conjunction with `dbGetPreparedQuery` in the RSQLite package and the `data.frame` resulting from the `dta.func` function to populate the initial database table.

mergeStatement `signature(obj = "TableSchemaList")`, table.name: Produces a statement joining an existing table and a temporary one and inserting into a new (non-temporary) tables

'relationship<-' `signature(obj = "TableSchemaList")`, from, to, value: Provides a mechanism to specify how two tables are connected to each other in a database. The arguments `from` and `to` should refer to tables in the specified `TableSchemaList`. The value should be a formula describing how the column(s) correspond to each other. The special value '.' refers to the autoincremented integer column if applicable. The simplest use would be to specify that two tables should be joined on the same column (e.g. `column1~column1`). Another typical use would be two say that the combination of one or more columns in one table should uniquely identify a row in another table (e.g. `.~column1+column2`).

'constraint<-' `signature(obj = "TableSchemaList")`, obj, table.name, should.ignore=T, constr.name=NULL, value: Allows the specification of uniqueness constraints for a given table (`table.name`) using the specified columns provided as a single sided formula (e.g. ~ column). `should.ignore` specifies whether a row of the input dataset should be ultimately ignored if determined to be duplicate in terms of the specified columns, by default it is set to TRUE. By default, `constr.name` sets the constraint name as 'table.name_idx', this can be changed by specifying `constr.name`. Setting this to NULL removes the constraint.

Author(s)

Daniel Bottomly

Examples

```
if (require(Lahman))
{
  baseball.teams <- new("TableSchemaList")

  franches <- makeSchemaFromData(TeamsFranchises, name="team_franch")

  baseball.teams <- append(baseball.teams, franches)

  teams <- makeSchemaFromData(Teams, name="teams")

  baseball.teams <- append(baseball.teams, teams)

  salaries <- makeSchemaFromData(Salaries, name="salaries")

  baseball.teams <- append(baseball.teams, salaries)

  relationship(baseball.teams, from="team_franch", to="teams") <- franchID ~ franchID

  relationship(baseball.teams, from="teams", to="salaries") <- teamID ~ teamID

  constraint(baseball.teams, "team_franch") <- ~franchID

  tables(baseball.teams)

  columns(baseball.teams)

}
```

Index