Package ‘Andromeda’

May 7, 2020

Type Package

Title Asynchronous Disk-Based Representation of Massive Data

Version 0.1.3

Date 2020-05-07

Maintainer Martijn Schuemie <schuemie@ohdsi.org>

Description Storing very large data objects on a local drive, while still making it possible to manipulate the data in an efficient manner.

License Apache License 2.0

VignetteBuilder knitr

URL https://github.com/OHDSI/Andromeda

BugReports https://github.com/OHDSI/Andromeda/issues

Depends dplyr

Imports RSQLite,
DBI,
zip,
methods,
dbplyr,
cli,
pillar,
Rcpp

Suggests testthat,
knitr,
rmarkdown,
rlang,
tibble

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.0

Encoding UTF-8

LinkingTo Rcpp

R topics documented:

andromeda — *Create an Andromeda object*

Description

By default the `Andromeda` object is created in the systems temporary file location. You can override this by specifying a folder using `options(andromedaTempFolder = "c:/andromedaTemp")`, where `"c:/andromedaTemp"` is the folder to create the Andromeda objects in.

Usage

```
andromeda(...)
```

Arguments

`...`	Named objects. See details for what objects are valid. If no objects are provided, an empty Andromeda is returned.

Details

Valid objects are data frames, `Andromeda` tables, or any other `dplyr` table.

Value

Returns an `Andromeda` object.

Examples

```
andr <- andromeda(cars = cars, iris = iris)

names(andr)
# [1] 'cars' 'iris'

andr$cars %>% filter(speed > 10) %>% collect()
# # A tibble: 41 x 2
# speed dist
# <dbl> <dbl>
# 1 11 17
# ...

close(andr)
```

Andromeda-class *The Andromeda class*

Description

The Andromeda class is an S4 object.

This class provides the ability to work with data objects in R that are too large to fit in memory. Instead, these objects are stored on disk. This is slower than working from memory, but may be the only viable option.

Show the names of the tables in an Andromeda object.

Usage

```
## S4 method for signature 'Andromeda'
show(object)

## S4 method for signature 'Andromeda'
x$name

## S4 replacement method for signature 'Andromeda'
x$name <- value

## S4 replacement method for signature 'Andromeda'
x[[i]] <- value

## S4 method for signature 'Andromeda'
x[[i]]

## S4 method for signature 'Andromeda'
names(x)

## S4 method for signature 'Andromeda'
length(x)

## S4 method for signature 'Andromeda'
close(con, ...)
```

Arguments

object	An Andromeda object.
x	An Andromeda object.
name	The name of a table in the Andromeda object.
value	A data frame, Andromeda table, or other 'DBI' table.
i	The name of a table in the Andromeda object.

`con`	An `Andromeda` object.
`...`	Included for compatibility with generic `close()` method.

Value

A vector of names.

Tables

An `Andromeda` object has zero, one or more tables. The list of table names can be retrieved using the `names()` method. Tables can be accessed using the dollar sign syntax, e.g. `andromeda$myTable`, or double-square-bracket syntax, e.g. `andromeda[["myTable"]]`

Permanence

To mimic the behavior of in-memory objects, when working with data in `Andromeda` the data is stored in a temporary location on the disk. You can modify the data as you can see fit, and when needed can save the data to a permanent location. Later this data can be loaded to a temporary location again and be read and modified, while keeping the saved data as is.

Inheritance

The `Andromeda` inherits directly from `SQLiteConnection.` As such, it can be used as if it is a `SQLiteConnection.` `RSQLite` is an R wrapper around 'SQLite', a low-weight but very powerful single-user SQL database that can run from a single file on the local file system.

See Also

`andromeda()`

Examples

```
andr <- andromeda(cars = cars, iris = iris)

names(andr)
# [1] 'cars' 'iris'

close(andr)
```

`appendToTable` *Append to an Andromeda table*

Description

Append a data frame, Andromeda table, or result of a query on an `Andromeda` table to an existing `Andromeda` table. If data from another `Andromeda` is appended, a batch-wise copy process is used, which will be slower than when appending data from within the same `Andromeda` object.

Usage

```
appendToTable(tbl, data)
```

Arguments

`tbl`	An `Andromeda` table. This must be a base table (i.e. it cannot be a query result).
`data`	The data to append. This can be either a data.frame or another Andromeda table.

Value

Returns no value. Executed for the side-effect of appending the data to the table.

Examples

```
andr <- andromeda(cars = cars)
nrow(andr$cars)
# [1] 50

appendToTable(andr$cars, cars)
nrow(andr$cars)
# [1] 100

appendToTable(andr$cars, andr$cars %>% filter(speed > 10))
nrow(andr$cars)
# [1] 182

close(andr)
```

batchApply *Apply a function to batches of data in an Andromeda table*

Description

Apply a function to batches of data in an Andromeda table

Usage

```
batchApply(tbl, fun, ..., batchSize = 1e+05, safe = FALSE)
```

Arguments

`tbl`	An `Andromeda` table (or any other 'DBI' table).
`fun`	A function where the first argument is a data frame.
`...`	Additional parameters passed to fun.
`batchSize`	Number of rows to fetch at a time.
`safe`	Create a copy of tbl first? Allows writing to the same Andromeda as being read from.

Details

This function is similar to the `lapply()` function, in that it applies a function to sets of data. In this case, the data is batches of data from an `Andromeda` table. Each batch will be presented to the function as a data frame.

Value

Invisibly returns a list of objects, where each object is the output of the user-supplied function applied to a batch

Examples

```
andr <- andromeda(cars = cars)

fun <- function(x) {
  return(nrow(x))
}

result <- batchApply(andr$cars, fun, batchSize = 25)

result
# [[1]]
# [1] 25
#
# [[2]]
# [1] 25

close(andr)
```

---

batchTest — *Apply a boolean test to batches of data in an Andromeda table and terminate early*

Description

Apply a boolean test to batches of data in an Andromeda table and terminate early

Usage

```
batchTest(tbl, fun, ..., batchSize = 1e+05)
```

Arguments

`tbl`	An `Andromeda` table (or any other 'DBI' table).
`fun`	A function where the first argument is a data frame and returns a logical value.
`...`	Additional parameters passed to `fun`.
`batchSize`	Number of rows to fetch at a time.

Details

This function applies a boolean test function to sets of data and terminates at the first `FALSE`.

Value

Returns `FALSE` if any of the calls to the user-supplied function returned `FALSE`, else returns `TRUE`.

Examples

```
andr <- andromeda(cars = cars)

fun <- function(x) {
  is.unsorted(x %>% select(speed) %>% collect())
}

result <- batchTest(andr$cars, fun, batchSize = 25)

result
# [1] FALSE

close(andr)
```

`copyAndromeda` *Copy Andromeda*

Description

Creates a complete copy of an `Andromeda` object. Object attributes are not copied.

Usage

```
copyAndromeda(andromeda)
```

Arguments

`andromeda`	The `Andromeda` object to copy.

Value

The copied `Andromeda` object.

Examples

```
andr <- andromeda(cars = cars, iris = iris)

andr2 <- copyAndromeda(andr)

names(andr2)
# [1] 'cars' 'iris'

close(andr)
close(andr2)
```

groupApply *Apply a function to groups of data in an Andromeda table*

Description

Apply a function to groups of data in an Andromeda table

Usage

```
groupApply(tbl, groupVariable, fun, ..., batchSize = 1e+05, safe = FALSE)
```

Arguments

tbl	An Andromeda table (or any other 'DBI' table).
groupVariable	The variable to group by
fun	A function where the first argument is a data frame.
...	Additional parameters passed to fun.
batchSize	Number of rows fetched from the table at a time. This is not the number of rows to which the function will be applied. Included mostly for testing purposes.
safe	Create a copy of tbl first? Allows writing to the same Andromeda as being read from.

Details

This function applies a function to groups of data. The groups are identified by unique values of the `groupVariable`, which must be a variable in the table.

Value

Invisibly returns a list of objects, where each object is the output of the user-supplied function applied to a group.

Examples

```
andr <- andromeda(cars = cars)

fun <- function(x) {
  return(tibble::tibble(speed = x$speed[1], meanDist = mean(x$dist)))
}

result <- groupApply(andr$cars, "speed", fun)
result <- bind_rows(result)
result
# # A tibble: 19 x 2
# speed meanDist
# <dbl> <dbl>
# 1 4 6
# 2 7 13
# 3 8 16
# ...

close(andr)
```

isAndomeda — *Check whether an object is an Andromeda object*

Description

Check whether an object is an Andromeda object

Usage

```
isAndomeda(x)
```

Arguments

x	The object to check.

Details

Checks whether an object is an Andromeda object.

Value

A logical value.

isSorted — *Check if data are sorted by one or more columns*

Description

Checks whether data are sorted by one or more specified columns.

Usage

```
isSorted(data, columnNames, ascending = rep(TRUE, length(columnNames)))

## S3 method for class 'data.frame'
isSorted(data, columnNames, ascending = rep(TRUE, length(columnNames)))

## S3 method for class 'tbl_dbi'
isSorted(data, columnNames, ascending = rep(TRUE, length(columnNames)))
```

Arguments

data	Either a data frame or Andromeda table.
columnNames	Vector of one or more column names.
ascending	Logical vector indicating the data should be sorted ascending or descending according the specified columns.

Details

This function currently only supports checking for sorting on numeric values.

Value

TRUE or FALSE.

Methods (by class)

- data.frame: Check if a data.frame is sorted by one or more columns
- tbl_dbi: Check if an Andromeda table is sorted by one or more columns

Examples

```
x <- data.frame(a = runif(1000), b = runif(1000))
x <- round(x, digits=2)
isSorted(x, c("a", "b"))

x <- x[order(x$a, x$b),]
isSorted(x, c("a", "b"))

x <- x[order(x$a,-x$b),]
isSorted(x, c("a", "b"), c(TRUE, FALSE))
```

isValidAndromeda *Check whether an Andromeda object is still valid*

Description

Check whether an Andromeda object is still valid

Usage

```
isValidAndromeda(x)
```

Arguments

x	The Andromeda object to check.

Details

Checks whether an Andromeda object is still valid, or whether it has been closed.

Value

A logical value.

Examples

```
andr <- andromeda(cars = cars, iris = iris)

isValidAndromeda(andr)
# TRUE

close(andr)

isValidAndromeda(andr)
# FALSE
```

loadAndromeda *Load Andromeda from file*

Description

Load Andromeda from file

Usage

```
loadAndromeda(fileName)
```

Arguments

fileName	The path where the object was saved using saveAndromeda().

Value

An Andromeda object.

See Also

saveAndromeda()

Examples

```
# For this example we create an Andromeda object and save it to
# a temporary file locationL
fileName <- tempfile()
andr <- andromeda(cars = cars)
saveAndromeda(andr, fileName)

# Using loadAndromeda to load the object back:
andr <- loadAndromeda(fileName)

# Don't forget to close Andromeda when you are done:
close(andr)

# Cleaning up the file used in this example:
unlink(fileName)
```

saveAndromeda *Save Andromeda to file*

Description

Saves the `Andromeda` object in a zipped file. Note that by default the `Andromeda` object is automatically closed by saving it to disk. This is due to a limitation of the underlying technology ('RSQLite'). To keep the connection open, use `maintainConnection = TRUE`. This will first create a temporary copy of the `Andromeda` object. Note that this can be substantially slower.

Usage

```
saveAndromeda(
  andromeda,
  fileName,
  maintainConnection = FALSE,
  overwrite = TRUE
)
```

Arguments

`andromeda` An object of class `Andromeda`.

`fileName` The path where the object will be written.

`maintainConnection` Should the connection be maintained after saving? If `FALSE`, the Andromeda object will be invalid after this operation, but saving will be faster.

`overwrite` If the file exists, should it be overwritten? If `FALSE` and the file exists, an error will be thrown.

Value

Returns no value. Executed for the side-effect of saving the object to disk.

See Also

`loadAndromeda`

`loadAndromeda()`

Examples

```
andr <- andromeda(cars = cars)

# For this example we'll use a temporary file location:
fileName <- tempfile()

saveAndromeda(andr, fileName)

# Cleaning up the file used in this example:
unlink(fileName)
```

Index