Package 'redshiftTools'

May 21, 2019

Type Package

Title Amazon Redshift Tools

Version 1.0.1

Depends R (>= 3.3.0)

Imports DBI, aws.s3, purrr, progress

Suggests RJDBC, RPostgres

Description Efficiently upload data into an Amazon Redshift database using the approach recommended by Amazon <https://aws.amazon.com/es/redshift/>.

URL https://github.com/sicarul/redshiftTools

BugReports https://github.com/sicarul/redshiftTools/issues

License MIT + file LICENSE

LazyData TRUE

RoxygenNote 6.1.1

Collate 'append.R' 'create.R' 'internal.R' 'replace.R' 'table_definition.R' 'upsert.R' 'upsert2.R'

NeedsCompilation no

Author Pablo Seibelt [aut, cre]

Maintainer Pablo Seibelt <pabloseibelt@sicarul.com>

Repository CRAN

Date/Publication 2019-05-21 16:20:03 UTC

R topics documented:

`rs_append_table` *Append redshift table*

Description

Upload a table to S3 and then append it to a Redshift Table. The table on redshift has to have the same structure and column ordering to work correctly.

Usage

```
rs_append_table(df, dbcon, table_name, split_files,
  bucket = Sys.getenv("AWS_BUCKET_NAME"),
  region = Sys.getenv("AWS_DEFAULT_REGION"),
  access_key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret_key = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  session_token = Sys.getenv("AWS_SESSION_TOKEN"),
  iam_role_arn = Sys.getenv("AWS_IAM_ROLE_ARN"), wlm_slots = 1,
  additional_params = "")
```

Arguments

`df`	a data frame
`dbcon`	an RPostgres/RJDBC connection to the redshift server
`table_name`	the name of the table to replace
`split_files`	optional parameter to specify amount of files to split into. If not specified will look at amount of slices in Redshift to determine an optimal amount.
`bucket`	the name of the temporary bucket to load the data. Will look for AWS_BUCKET_NAME on environment if not specified.
`region`	the region of the bucket. Will look for AWS_DEFAULT_REGION on environment if not specified.
`access_key`	the access key with permissions for the bucket. Will look for AWS_ACCESS_KEY_ID on environment if not specified.
`secret_key`	the secret key with permissions fot the bucket. Will look for AWS_SECRET_ACCESS_KEY on environment if not specified.
`session_token`	the session key with permissions for the bucket, this will be used instead of the access/secret keys if specified. Will look for AWS_SESSION_TOKEN on environment if not specified.
`iam_role_arn`	an iam role arn with permissions fot the bucket. Will look for AWS_IAM_ROLE_ARN on environment if not specified. This is ignoring access_key and secret_key if set.
`wlm_slots`	amount of WLM slots to use for this bulk load http://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html
`additional_params`	Additional params to send to the COPY statement in Redshift

Examples

```
library(DBI)

a=data.frame(a=seq(1,10000), b=seq(10000,1))

## Not run:
con <- dbConnect(RPostgres::Postgres(), dbname="dbname",
host='my-redshift-url.amazon.com', port='5439',
user='myuser', password='mypassword',sslmode='require')

rs_append_table(df=a, dbcon=con, table_name='testTable',
bucket="my-bucket", split_files=4)


## End(Not run)
```

`rs_cols_upsert_table` *Upsert Redshift table by specifying a column list*

Description

Upload a table to S3 and then load to Redshift, replacing the target value in existing rows that have the same keys, and inserting rows with new keys. New rows must match structure and column ordering of existing Redshift table.

Usage

```
rs_cols_upsert_table(dat, dbcon, table_name, values, keys, split_files,
  bucket = Sys.getenv("AWS_BUCKET_NAME"),
  region = Sys.getenv("AWS_DEFAULT_REGION"),
  access_key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret_key = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  session_token = Sys.getenv("AWS_SESSION_TOKEN"),
  iam_role_arn = Sys.getenv("AWS_IAM_ROLE_ARN"), wlm_slots = 1,
  additional_params = "")
```

Arguments

`dat`	a data frame
`dbcon`	an RPostgres/RJDBC connection to the redshift server
`table_name`	the name of the table to update/insert
`values`	the columns that will be updated
`keys`	this optional vector contains the variables by which to upsert. If not defined, the upsert becomes an append.
`split_files`	optional parameter to specify amount of files to split into. If not specified will look at amount of slices in Redshift to determine an optimal amount.

bucket	the name of the temporary bucket to load the data. Will look for AWS_BUCKET_NAME on environment if not specified.
region	the region of the bucket. Will look for AWS_DEFAULT_REGION on environment if not specified.
access_key	the access key with permissions for the bucket. Will look for AWS_ACCESS_KEY_ID on environment if not specified.
secret_key	the secret key with permissions for the bucket. Will look for AWS_SECRET_ACCESS_KEY on environment if not specified.
session_token	the session key with permissions for the bucket, this will be used instead of the access/secret keys if specified. Will look for AWS_SESSION_TOKEN on environment if not specified.
iam_role_arn	an iam role arn with permissions fot the bucket. Will look for AWS_IAM_ROLE_ARN on environment if not specified. This is ignoring access_key and secret_key if set.
wlm_slots	amount of WLM slots to use for this bulk load http://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html
additional_params	Additional params to send to the COPY statement in Redshift

See Also

https://docs.aws.amazon.com/redshift/latest/dg/merge-specify-a-column-list.html

http://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html

Examples

```
library(DBI)

a=data.frame(a=seq(1,10000), b=seq(10000,1))
n=head(a,n=5000)
n$b=n$a
nx=rbind(n, data.frame(a=seq(99999:104000), b=seq(104000:99999)))

## Not run:
con <- dbConnect(RPostgres::Postgres(), dbname="dbname",
host='my-redshift-url.amazon.com', port='5439',
user='myuser', password='mypassword',sslmode='require')

rs_cols_upsert_table(df=nx, dbcon=con, table_name='testTable',
bucket="my-bucket", split_files=4, values=c('b'), keys=c('a'))

## End(Not run)
```

`rs_create_statement` *Generate create table statement for Amazon Redshift*

Description

This lets you easily generate a table schema from a data.frame, which allows for easily uploading to redshift afterwards.

Usage

```
rs_create_statement(df, table_name = deparse(substitute(df)), sortkeys,
  sortkey_style = "compound", distkey, distkey_style = "even",
  compression = T)
```

Arguments

`df`	the data.frame you want to upload to Amazon Redshift
`table_name`	the name of the table to create, if not specified it'll use the data.frame name
`sortkeys`	Column or columns to sort the table by
`sortkey_style`	Sortkey style, can be compound or interleaved http://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_da compare-sort-styles.html
`distkey`	Distkey column, can only be one, if chosen the table is distributed among clusters according to a hash of this column's value.
`distkey_style`	Distkey style, can be even or all, for the key distribution use the distkey parameter. http://docs.aws.amazon.com/redshift/latest/dg/t_Distributing_data.html
`compression`	Add encoding for columns whose compression algorithm is easy to guess, for the rest you should upload it to Redshift and run ANALYZE COMPRESSION

Examples

```
n=1000
testdf = data.frame(
a=rep('a', n),
b=c(1:n),
c=rep(as.Date('2017-01-01'), n),
d=rep(as.POSIXct('2017-01-01 20:01:32'), n),
e=rep(as.POSIXlt('2017-01-01 20:01:32'), n),
f=rep(paste0(rep('a', 4000), collapse=''), n) )

cat(rs_create_statement(testdf, table_name='dm_great_table'))
```

rs_create_table *Create a table from scratch, guessing the table schema*

Description

Create a table from scratch, guessing the table schema

Usage

```
rs_create_table(df, dbcon, table_name = deparse(substitute(df)),
  split_files, bucket = Sys.getenv("AWS_BUCKET_NAME"),
  region = Sys.getenv("AWS_DEFAULT_REGION"),
  access_key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret_key = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  session_token = Sys.getenv("AWS_SESSION_TOKEN"),
  iam_role_arn = Sys.getenv("AWS_IAM_ROLE_ARN"), wlm_slots = 1,
  sortkeys, sortkey_style = "compound", distkey,
  distkey_style = "even", compression = T, additional_params = "")
```

Arguments

df	a data frame
dbcon	an RPostgres/RJDBC connection to the redshift server
table_name	the name of the table to create
split_files	optional parameter to specify amount of files to split into. If not specified will look at amount of slices in Redshift to determine an optimal amount.
bucket	the name of the temporary bucket to load the data. Will look for AWS_BUCKET_NAME on environment if not specified.
region	the region of the bucket. Will look for AWS_DEFAULT_REGION on environment if not specified.
access_key	the access key with permissions for the bucket. Will look for AWS_ACCESS_KEY_ID on environment if not specified.
secret_key	the secret key with permissions fot the bucket. Will look for AWS_SECRET_ACCESS_KEY on environment if not specified.
session_token	the session key with permissions for the bucket, this will be used instead of the access/secret keys if specified. Will look for AWS_SESSION_TOKEN on environment if not specified.
iam_role_arn	an iam role arn with permissions fot the bucket. Will look for AWS_IAM_ROLE_ARN on environment if not specified. This is ignoring access_key and secret_key if set.
wlm_slots	amount of WLM slots to use for this bulk load http://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html
sortkeys	Column or columns to sort the table by

sortkey_style	Sortkey style, can be compound or interleaved http://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_da compare-sort-styles.html
distkey	Distkey column, can only be one, if chosen the table is distributed among clusters according to a hash of this column's value.
distkey_style	Distkey style, can be even or all, for the key distribution use the distkey parameter. http://docs.aws.amazon.com/redshift/latest/dg/t_Distributing_data.html
compression	Add encoding for columns whose compression algorithm is easy to guess, for the rest you should upload it to Redshift and run ANALYZE COMPRESSION
additional_params	Additional params to send to the COPY statement in Redshift

Examples

```
library(DBI)

a=data.frame(a=seq(1,10000), b=seq(10000,1))

## Not run:
con <- dbConnect(RPostgres::Postgres(), dbname="dbname",
host='my-redshift-url.amazon.com', port='5439',
user='myuser', password='mypassword',sslmode='require')

rs_create_table(df=a, dbcon=con, table_name='testTable',
bucket="my-bucket", split_files=4)


## End(Not run)
```

`rs_replace_table` *Replace redshift table*

Description

Upload a table to S3 and then load it with redshift, replacing the contents of that table. The table on redshift has to have the same structure and column ordering to work correctly.

Usage

```
rs_replace_table(df, dbcon, table_name, split_files,
  bucket = Sys.getenv("AWS_BUCKET_NAME"),
  region = Sys.getenv("AWS_DEFAULT_REGION"),
  access_key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret_key = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  session_token = Sys.getenv("AWS_SESSION_TOKEN"),
  iam_role_arn = Sys.getenv("AWS_IAM_ROLE_ARN"), wlm_slots = 1,
  additional_params = "")
```

Arguments

df	a data frame
dbcon	an RPostgres/RJDBC connection to the redshift server
table_name	the name of the table to replace
split_files	optional parameter to specify amount of files to split into. If not specified will look at amount of slices in Redshift to determine an optimal amount.
bucket	the name of the temporary bucket to load the data. Will look for AWS_BUCKET_NAME on environment if not specified.
region	the region of the bucket. Will look for AWS_DEFAULT_REGION on environment if not specified.
access_key	the access key with permissions for the bucket. Will look for AWS_ACCESS_KEY_ID on environment if not specified.
secret_key	the secret key with permissions fot the bucket. Will look for AWS_SECRET_ACCESS_KEY on environment if not specified.
session_token	the session key with permissions for the bucket, this will be used instead of the access/secret keys if specified. Will look for AWS_SESSION_TOKEN on environment if not specified.
iam_role_arn	an iam role arn with permissions fot the bucket. Will look for AWS_IAM_ROLE_ARN on environment if not specified. This is ignoring access_key and secret_key if set.
wlm_slots	amount of WLM slots to use for this bulk load http://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html
additional_params	Additional params to send to the COPY statement in Redshift

Examples

```
library(DBI)

a=data.frame(a=seq(1,10000), b=seq(10000,1))

## Not run:
con <- dbConnect(RPostgres::Postgres(), dbname="dbname",
host='my-redshift-url.amazon.com', port='5439',
user='myuser', password='mypassword',sslmode='require')

rs_replace_table(df=a, dbcon=con, table_name='testTable',
bucket="my-bucket", split_files=4)


## End(Not run)
```

`rs_upsert_table` *Upsert redshift table*

Description

Upload a table to S3 and then load it with redshift, replacing rows with the same keys, and inserting rows with new keys. The table on redshift has to have the same structure and column ordering to work correctly.

Usage

```
rs_upsert_table(df, dbcon, table_name, keys, split_files,
  bucket = Sys.getenv("AWS_BUCKET_NAME"),
  region = Sys.getenv("AWS_DEFAULT_REGION"),
  access_key = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret_key = Sys.getenv("AWS_SECRET_ACCESS_KEY"),
  session_token = Sys.getenv("AWS_SESSION_TOKEN"),
  iam_role_arn = Sys.getenv("AWS_IAM_ROLE_ARN"), wlm_slots = 1,
  additional_params = "")
```

Arguments

`df`	a data frame
`dbcon`	an RPostgres/RJDBC connection to the redshift server
`table_name`	the name of the table to update/insert
`keys`	this optional vector contains the variables by which to upsert. If not defined, the upsert becomes an append.
`split_files`	optional parameter to specify amount of files to split into. If not specified will look at amount of slices in Redshift to determine an optimal amount.
`bucket`	the name of the temporary bucket to load the data. Will look for AWS_BUCKET_NAME on environment if not specified.
`region`	the region of the bucket. Will look for AWS_DEFAULT_REGION on environment if not specified.
`access_key`	the access key with permissions for the bucket. Will look for AWS_ACCESS_KEY_ID on environment if not specified.
`secret_key`	the secret key with permissions for the bucket. Will look for AWS_SECRET_ACCESS_KEY on environment if not specified.
`session_token`	the session key with permissions for the bucket, this will be used instead of the access/secret keys if specified. Will look for AWS_SESSION_TOKEN on environment if not specified.
`iam_role_arn`	an iam role arn with permissions fot the bucket. Will look for AWS_IAM_ROLE_ARN on environment if not specified. This is ignoring access_key and secret_key if set.

`wlm_slots` amount of WLM slots to use for this bulk load http://docs.aws.amazon.com/redshift/latest/dg/tutorial-configuring-workload-management.html

`additional_params` Additional params to send to the COPY statement in Redshift

Examples

```
library(DBI)

a=data.frame(a=seq(1,10000), b=seq(10000,1))
n=head(a,n=5000)
n$b=n$a
nx=rbind(n, data.frame(a=seq(99999:104000), b=seq(104000:99999)))

## Not run:
con <- dbConnect(RPostgres::Postgres(), dbname="dbname",
host='my-redshift-url.amazon.com', port='5439',
user='myuser', password='mypassword',sslmode='require')

rs_upsert_table(df=nx, dbcon=con, table_name='testTable',
bucket="my-bucket", split_files=4, keys=c('a'))


## End(Not run)
```

Index