Package ‘RAthena’

October 12, 2022

Type Package
Title Connect to ‘AWS Athena’ using ‘Boto3’ (‘DBI’ Interface)
Version 2.6.0
Description Designed to be compatible with the R package ‘DBI’ (Database Interface) when connecting to Amazon Web Service (‘AWS’) Athena <https://aws.amazon.com/athena/>. To do this ‘Python’ ‘Boto3’ Software Development Kit (‘SDK’) <https://boto3.amazonaws.com/v1/documentation/api/latest/index.html> is used as a driver.

Imports data.table (>= 1.12.4), DBI (>= 0.7), methods, reticulate (>= 1.13), stats, utils, uuid (>= 0.1-4)
Suggests arrow, bit64, dplyr (>= 0.8.0), dbplyr (>= 1.4.3), testthat, tibble, vroom (>= 1.2.0), covr, knitr, rmarkdown, jsonify, jsonlite

VignetteBuilder knitr
Depends R (>= 3.2.0)
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.2.0
URL https://github.com/DyfanJones/RAthena
BugReports https://github.com/DyfanJones/RAthena/issues

NeedsCompilation no
Author Dyfan Jones [aut, cre]
Maintainer Dyfan Jones <dyfan.r.jones@gmail.com>
Repository CRAN
Date/Publication 2022-05-20 13:10:02 UTC
## R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAthena-package</td>
<td>3</td>
</tr>
<tr>
<td>assume_role</td>
<td>4</td>
</tr>
<tr>
<td>athena</td>
<td>5</td>
</tr>
<tr>
<td>AthenaWriteTables</td>
<td>6</td>
</tr>
<tr>
<td>backend_dbplyr_v1</td>
<td>9</td>
</tr>
<tr>
<td>backend_dbplyr_v2</td>
<td>10</td>
</tr>
<tr>
<td>dbClearResult</td>
<td>11</td>
</tr>
<tr>
<td>dbColumnInfo</td>
<td>12</td>
</tr>
<tr>
<td>dbConnect,AthenaDriver-method</td>
<td>13</td>
</tr>
<tr>
<td>dbConvertTable</td>
<td>17</td>
</tr>
<tr>
<td>dbDataType,AthenaDriver,ANY-method</td>
<td>18</td>
</tr>
<tr>
<td>dbDisconnect</td>
<td>20</td>
</tr>
<tr>
<td>dbExistsTable</td>
<td>21</td>
</tr>
<tr>
<td>dbFetch</td>
<td>22</td>
</tr>
<tr>
<td>dbGetInfo</td>
<td>23</td>
</tr>
<tr>
<td>dbGetPartition</td>
<td>24</td>
</tr>
<tr>
<td>dbGetQuery</td>
<td>25</td>
</tr>
<tr>
<td>dbGetStatement</td>
<td>26</td>
</tr>
<tr>
<td>dbGetTables</td>
<td>27</td>
</tr>
<tr>
<td>dbHasCompleted</td>
<td>28</td>
</tr>
<tr>
<td>dbIsValid</td>
<td>29</td>
</tr>
<tr>
<td>dbListFields</td>
<td>30</td>
</tr>
<tr>
<td>dbListTables</td>
<td>31</td>
</tr>
<tr>
<td>dbplyr_edition</td>
<td>32</td>
</tr>
<tr>
<td>dbQuote</td>
<td>33</td>
</tr>
<tr>
<td>dbRemoveTable</td>
<td>34</td>
</tr>
<tr>
<td>dbShow</td>
<td>35</td>
</tr>
<tr>
<td>dbStatistics</td>
<td>36</td>
</tr>
<tr>
<td>db_compute</td>
<td>37</td>
</tr>
<tr>
<td>db_connection_describe</td>
<td>39</td>
</tr>
<tr>
<td>db_copy_to</td>
<td>39</td>
</tr>
<tr>
<td>db_desc</td>
<td>41</td>
</tr>
<tr>
<td>install_boto</td>
<td>42</td>
</tr>
<tr>
<td>Query</td>
<td>43</td>
</tr>
<tr>
<td>RAthena_options</td>
<td>44</td>
</tr>
<tr>
<td>session_token</td>
<td>45</td>
</tr>
<tr>
<td>sqlCreateTable</td>
<td>47</td>
</tr>
<tr>
<td>sqlData</td>
<td>49</td>
</tr>
<tr>
<td>sql_translate_env</td>
<td>50</td>
</tr>
<tr>
<td>work_group</td>
<td>50</td>
</tr>
</tbody>
</table>

Index 54
Description

RAthena provides a seamless DBI interface into Athena using the python package Boto3.

Goal of Package

The goal of the RAthena package is to provide a DBI-compliant interface to Amazon’s Athena using Boto3 software development kit (SDK). This allows for an efficient, easy setup connection to Athena using the Boto3 SDK as a driver.

Installation

Before starting with RAthena, Python is require to be installed on the machine you are intending to run RAthena.

AWS Command Line Interface

As RAthena is using Boto3 as it’s backend, AWS Command Line Interface (AWS CLI) can be used to remove user credentials when interacting with Athena.

This allows AWS profile names to be set up so that RAthena can connect to different accounts from the same machine, without needing hard code any credentials.

Author(s)

Maintainer: Dyfan Jones <dyfan.r.jones@gmail.com>

See Also

Useful links:

- https://github.com/DyfanJones/RAthena
Assume AWS ARN Role

Description

Returns a set of temporary security credentials that you can use to access AWS resources that you might not normally have access to (link). These temporary credentials consist of an access key ID, a secret access key, and a security token. Typically, you use AssumeRole within your account or for cross-account access.

Usage

```r
assume_role(
  profile_name = NULL,
  region_name = NULL,
  role_arn = NULL,
  role_session_name = sprintf("RAthena-session-%s", as.integer(Sys.time())),
  duration_seconds = 3600L,
  set_env = FALSE
)
```

Arguments

- **profile_name**: The name of a profile to use. If not given, then the default profile is used. To set profile name, the AWS Command Line Interface (AWS CLI) will need to be configured. To configure AWS CLI please refer to: Configuring the AWS CLI.
- **region_name**: Default region when creating new connections. Please refer to link for AWS region codes (region code example: Region = EU (Ireland) `region_name = "eu-west-1"`).
- **role_arn**: The Amazon Resource Name (ARN) of the role to assume (such as `arn:aws:sts::123456789012:assumed-role/role_name/role_session_name`).
- **role_session_name**: An identifier for the assumed role session. By default 'RAthena' creates a session name `sprintf("RAthena-session-%s", as.integer(Sys.time()))`.
- **duration_seconds**: The duration, in seconds, of the role session. The value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role. This setting can have a value from 1 hour to 12 hours. By default duration is set to 3600 seconds (1 hour).
- **set_env**: If set to TRUE environmental variables `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` and `AWS_SESSION_TOKEN` will be set.

Value

`assume_role()` returns a list containing: "AccessKeyId", "SecretAccessKey", "SessionToken" and "Expiration"
See Also
dbConnect

Examples

## Not run:
# Note:
# - Require AWS Account to run below example.

library(RAthena)
library(DBI)

# Assuming demo ARN role
assume_role(profile_name = "YOUR_PROFILE_NAME",
            role_arn = "arn:aws:sts::123456789012:assumed-role/role_name/role_session_name",
            set_env = TRUE)

# Connect to Athena using ARN Role
con <- dbConnect(RAthena::athena())

## End(Not run)
AthenaWriteTables  Convenience functions for reading/writing DBMS tables

Description

Convenience functions for reading/writing DBMS tables

Usage

```r
## S4 method for signature 'AthenaConnection,character,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  row.names = NA,
  field.types = NULL,
  partition = NULL,
  s3.location = NULL,
  file.type = c("tsv", "csv", "parquet", "json"),
  compress = FALSE,
  max.batch = Inf,
  ...
)

## S4 method for signature 'AthenaConnection,Id,data.frame'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  row.names = NA,
  field.types = NULL,
  partition = NULL,
  s3.location = NULL,
  file.type = c("tsv", "csv", "parquet", "json"),
  compress = FALSE,
  max.batch = Inf,
  ...
)

## S4 method for signature 'AthenaConnection,SQL,data.frame'
dbWriteTable(
  conn,
  name,
```
value,
overwrite = FALSE,
append = FALSE,
row.names = NA,
field.types = NULL,
partition = NULL,
s3.location = NULL,
file.type = c("tsv", "csv", "parquet", "json"),
compress = FALSE,
max.batch = Inf,
...
)

Arguments

conn  An AthenaConnection object, produced by [DBI::dbConnect()]
name  A character string specifying a table name. Names will be automatically quoted so you can use any sequence of characters, not just any valid bare table name.
value  A data.frame to write to the database.
overwrite  Allows overwriting the destination table. Cannot be TRUE if append is also TRUE.
append  Allow appending to the destination table. Cannot be TRUE if overwrite is also TRUE. Existing Athena DDL file type will be retained and used when uploading data to AWS Athena. If parameter file.type doesn’t match AWS Athena DDL file type a warning message will be created notifying user and RAthena will use the file type for the Athena DDL. When appending to an Athena DDL that has been created outside of RAthena. RAthena can support the following SerDes and Data Formats.
  • csv/tsv: LazySimpleSerDe
  • parquet: Parquet SerDe
  • json: JSON SerDe Libraries
row.names  Either TRUE, FALSE, NA or a string. If TRUE, always translate row names to a column called "row_names". If FALSE, never translate row names. If NA, translate rownames only if they’re a character vector.
A string is equivalent to TRUE, but allows you to override the default name.
For backward compatibility, NULL is equivalent to FALSE.
field.types  Additional field types used to override derived types.
partition  Partition Athena table (needs to be a named list or vector) for example: c(var1 = "2019-02-13")
s3.location  s3 bucket to store Athena table, must be set as a s3 uri for example ('s3://mybucket/data/'). By default, the s3.location is set to s3 staging directory from AthenaConnection object. Note: When creating a table for the first time s3.location will be formatted from "s3://mybucket/data/" to the following syntax "s3://(mybucket/data)/{schema}/{table}/{partition}/". this is to support tables with the same name but existing in different schemas. If schema isn’t specified in name parameter then the schema from dbConnect is used instead.
AthenaWriteTables

file.type  What file type to store data.frame on s3. RAthena currently supports ["tsv", "csv", "parquet", "json"]. Default delimited file type is "tsv", in previous versions of RAthena (<= 1.6.0) file type "csv" was used as default. The reason for the change is that columns containing Array/JSON format cannot be written to Athena due to the separating value ",". This would cause issues with AWS Athena. Note: "parquet" format is supported by the arrow package and it will need to be installed to utilise the "parquet" format. "json" format is supported by jsonlite package and it will need to be installed to utilise the "json" format.

compress  FALSE | TRUE To determine if to compress file.type. If file type is ["csv", "tsv"] then "gzip" compression is used, for file type "parquet" "snappy" compression is used. Currently RAthena doesn’t support compression for "json" file type.

max.batch  Split the data frame by max number of rows i.e. 100,000 so that multiple files can be uploaded into AWS S3. By default when compression is set to TRUE and file.type is "csv" or "tsv" max.batch will split data.frame into 20 batches. This is to help the performance of AWS Athena when working with files compressed in "gzip" format. max.batch will not split the data.frame when loading file in parquet format. For more information please go to link

...  Other arguments used by individual methods.

Value
dbWriteTable() returns TRUE, invisibly. If the table exists, and both append and overwrite arguments are unset, or append = TRUE and the data frame with the new data has different column names, an error is raised; the remote table remains unchanged.

See Also
dbWriteTable

Examples

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation
library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# List existing tables in Athena
dbListTables(con)

# Write data.frame to Athena table
dbWriteTable(con, "mtcars", mtcars, 
  partition=c("TIMESTAMP" = format(Sys.Date(), "%Y%m%d"),
  s3.location = "s3://mybucket/data/")

# Read entire table from Athena
dbReadTable(con, "mtcars")

# List all tables in Athena after uploading new table to Athena
dbListTables(con)

# Checking if uploaded table exists in Athena
dbExistsTable(con, "mtcars")

# using default s3.location
dbWriteTable(con, "iris", iris)

# Read entire table from Athena
dbReadTable(con, "iris")

# List all tables in Athena after uploading new table to Athena
dbListTables(con)

# Checking if uploaded table exists in Athena
dbExistsTable(con, "iris")

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)

backend_dbplyr_v1

Athena S3 implementation of dbplyr backend functions (api version 1).

Description

These functions are used to build the different types of SQL queries. The AWS Athena implementation gives extra parameters to allow access to standard DBI Athena methods. They also utilise AWS Glue to speed up sql query execution.

Usage

db_explain.AthenaConnection(con, sql, ...)

db_query_fields.AthenaConnection(con, sql, ...)

Arguments

- **con**: A dbConnect object, as returned by dbConnect()
- **sql**: SQL code to be sent to AWS Athena
- ... other parameters, currently not implemented

Value

- **db_explain**: Returns AWS Athena explain statement
- **db_query_fields**: Returns sql query column names
**backend_dbplyr_v2**  
_Athena S3 implementation of dbplyr backend functions (api version 2)._  

**Description**

These functions are used to build the different types of SQL queries. The AWS Athena implementation give extra parameters to allow access the to standard DBI Athena methods. They also utilise AWS Glue to speed up sql query execution.

**Usage**

```r
sql_query_explain.AthenaConnection(con, sql, format = "text", type = NULL, ...)
sql_query_fields.AthenaConnection(con, sql, ...)
sql_escape_date.AthenaConnection(con, x)
sql_escape_datetime.AthenaConnection(con, x)
```

**Arguments**

- `con`: A `dbConnect` object, as returned by `dbConnect()`
- `sql`: SQL code to be sent to AWS Athena
- `format`: returning format for explain queries, default set to “text”. Other formats can be found: https://docs.aws.amazon.com/athena/latest/ug/athena-explain-statement.html
- `type`: return plan for explain queries, default set to ‘NULL’. Other type can be found: https://docs.aws.amazon.com/athena/latest/ug/athena-explain-statement.html
- `...`: other parameters, currently not implemented
- `x`: R object to be transformed into athena equivalent

**Value**

- `sql_query_explain`: Returns sql query for AWS Athena explain statement
- `sql_query_fields`: Returns sql query column names
- `sql_escape_date`: Returns sql escaping from dates
- `sql_escape_datetime`: Returns sql escaping from date times
dbClearResult

Description
Frees all resources (local and Athena) associated with result set. It does this by removing query output in AWS S3 Bucket, stopping query execution if still running and removed the connection resource locally.

Usage
```r
## S4 method for signature 'AthenaResult'
dbClearResult(res, ...)
```

Arguments
- `res`: An object inheriting from `DBIResult`.
- `...`: Other arguments passed on to methods.

Value
dbClearResult() returns `TRUE`, invisibly.

Note
If the user does not have permission to remove AWS S3 resource from AWS Athena output location, then an AWS warning will be returned. It is better use query caching or optionally prevent clear AWS S3 resource using `RAthena_options` so that the warning doesn’t repeatedly show.

See Also
dbIsValid

Examples
```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation
library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

res <- dbSendQuery(con, "show databases")
dbClearResult(res)

# Check if connection if valid after closing connection
```
dbColumnInfo

Information about result types

Description

Produces a data.frame that describes the output of a query.

Usage

## S4 method for signature 'AthenaResult'
dbColumnInfo(res, ...)

Arguments

res

An object inheriting from DBIResult.

...

Other arguments passed on to methods.

Value

dbColumnInfo() returns a data.frame with as many rows as there are output fields in the result. The data.frame has two columns (field_name, type).

See Also

dbHasCompleted

Examples

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see 'RAthena::dbConnect' documentation
library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Get Column information from query
res <- dbSendQuery(con, "select * from information_schema.tables")
dbColumnInfo(res)
dbClearResult(res)

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)
Description

It is never advised to hard-code credentials when making a connection to Athena (even though the option is there). Instead it is advised to use profile_name (set up by AWS Command Line Interface), Amazon Resource Name roles or environmental variables. Here is a list of supported environment variables:

- **AWS_ACCESS_KEY_ID**: is equivalent to the dbConnect parameter - aws_access_key_id
- **AWS_SECRET_ACCESS_KEY**: is equivalent to the dbConnect parameter - aws_secret_access_key
- **AWS_SESSION_TOKEN**: is equivalent to the dbConnect parameter - aws_session_token
- **AWS_ROLE_ARN**: is equivalent to the dbConnect parameter - role_arn
- **AWS_EXPIRATION**: is equivalent to the dbConnect parameter - duration_seconds
- **AWS_ATHENA_S3_STAGING_DIR**: is equivalent to the dbConnect parameter - s3_staging_dir
- **AWS_ATHENA_WORK_GROUP**: is equivalent to dbConnect parameter - work_group
- **AWS_REGION**: is equivalent to dbConnect parameter - region_name

**NOTE:** If you have set any environmental variables in .Renviron please restart your R in order for the changes to take affect.

Usage

```r
## S4 method for signature 'AthenaDriver'
dbConnect(
drv,
    aws_access_key_id = NULL,
    aws_secret_access_key = NULL,
    aws_session_token = NULL,
    schema_name = "default",
    work_group = NULL,
    poll_interval = NULL,
    encryption_option = c("NULL", "SSE_S3", "SSE_KMS", "CSE_KMS"),
    kms_key = NULL,
    profile_name = NULL,
    role_arn = NULL,
    role_session_name = sprintf("RAthena-session-%s", as.integer(Sys.time())),
    duration_seconds = 3600L,
    s3_staging_dir = NULL,
    region_name = NULL,
    botocore_session = NULL,
    bigint = c("integer64", "integer", "numeric", "character"),
    binary = c("raw", "character"),
)```
json = c("auto", "character"),
timezone = "UTC",
keyboard_interrupt = TRUE,
rstudio_conn_tab = TRUE,
endpoint_override = NULL,
...
)

Arguments

drv an object that inherits from DBDriver, or an existing DBIConnection object (in order to clone an existing connection).

aws_access_key_id AWS access key ID

aws_secret_access_key AWS secret access key

aws_session_token AWS temporary session token

schema_name The schema_name to which the connection belongs

work_group The name of the work group to run Athena queries. Currently defaulted to NULL.
poll_interval Amount of time took when checking query execution status. Default set to a random interval between 0.5 - 1 seconds.

encryption_option Athena encryption at rest link. Supported Amazon S3 Encryption Options ["NULL", "SSE_S3", "SSE_KMS", "CSE_KMS"]. Connection will default to NULL, usually changing this option is not required.

kms_key AWS Key Management Service, please refer to link for more information around the concept.

profile_name The name of a profile to use. If not given, then the default profile is used. To set profile name, the AWS Command Line Interface (AWS CLI) will need to be configured. To configure AWS CLI please refer to: Configuring the AWS CLI.

role_arn The Amazon Resource Name (ARN) of the role to assume (such as arn:aws:sts::123456789012:assume

role_session_name An identifier for the assumed role session. By default 'RAthena' creates a session name sprintf("RAthena-session-%s", as.integer(Sys.time())))

duration_seconds The duration, in seconds, of the role session. The value can range from 900 seconds (15 minutes) up to the maximum session duration setting for the role. This setting can have a value from 1 hour to 12 hours. By default duration is set to 3600 seconds (1 hour).

s3_staging_dir The location in Amazon S3 where your query results are stored, such as s3://path/to/query/bucket/

region_name Default region when creating new connections. Please refer to link for AWS region codes (region code example: Region = EU (Ireland) region_name = "eu-west-1")

botocore_session Use this Botocore session instead of creating a new default one.
dbConnect.AthenaDriver-method

bigint
The R type that 64-bit integer types should be mapped to, default is [bit64::integer64], which allows the full range of 64 bit integers.

binary
The R type that [binary/varbinary] types should be mapped to, default is [raw]. If the mapping fails R will resort to [character] type. To ignore data type conversion set to ["character"].

json
Attempt to converts AWS Athena data types [arrays, json] using jsonlite::parse_json. If the mapping fails R will resort to [character] type. Custom Json parsers can be provide by using a function with data frame parameter. To ignore data type conversion set to ["character"].

timezone
Sets the timezone for the connection. The default is ‘UTC’. If ‘NULL’ then no timezone is set, which defaults to the server’s time zone. ‘AWS Athena’ accepted time zones: https://docs.aws.amazon.com/athena/latest/ug/athena-supported-time-zones.html.

keyboard_interrupt
Stops AWS Athena process when R gets a keyboard interrupt, currently defaults to TRUE

rstudio_conn_tab
Optional to get AWS Athena Schema from AWS Glue Catalogue and display it in RStudio’s Connections Tab. Default set to TRUE. For large ‘AWS Glue Catalogue’ it is recommended to set ‘rstudio_conn_tab=FALSE’ to ensure a fast connection.

dagent_connection
(character/list) The complete URL to use for the constructed client. Normally, botocore will automatically construct the appropriate URL to use when communicating with a service. You can specify a complete URL (including the "http/https" scheme) to override this behaviour. If endpoint_override is a character then AWS Athena endpoint is overridden. To override AWS S3 or AWS Glue endpoints a named list needs to be provided. The list can only have the following names ['athena', 's3', 'glue'] for example list(glue = "https://glue.eu-west-1.amazonaws.com")

... Passes parameters to boto3.session.Session and client.

• boto3.session.Session
  – botocore_session (botocore.session.Session): Use this Botocore session instead of creating a new default one.

• client
  – config (botocore.client.Config) – Advanced client configuration options. If region_name is specified in the client config, its value will take precedence over environment variables and configuration values, but not over a region_name value passed explicitly to the method. See botocore config documentation for more details.
  – api_version (string) – The API version to use. By default, botocore will use the latest API version when creating a client. You only need to specify this parameter if you want to use a previous API version of the client.
  – use_ssl (boolean) – Whether or not to use SSL. By default, SSL is used. Note that not all services support non-ssl connections.
- **verify** (boolean/string) – Whether or not to verify SSL certificates. By default SSL certificates are verified. You can provide the following values:

  * False - do not validate SSL certificates. SSL will still be used (unless use_ssl is False), but SSL certificates will not be verified.
  * path/to/cert/bundle.pem - A filename of the CA cert bundle to uses. You can specify this argument if you want to use a different CA cert bundle than the one used by botocore.

**Value**

dbConnect() returns a s4 class. This object is used to communicate with AWS Athena.

**See Also**

dbConnect

**Examples**

```r
## Not run:
# Connect to Athena using your aws access keys
library(DBI)
con <- dbConnect(RAthena::athena(),
aws_access_key_id='YOUR_ACCESS_KEY_ID', #
aws_secret_access_key='YOUR_SECRET_ACCESS_KEY',
s3_staging_dir='s3://path/to/query/bucket/',
region_name='us-west-2')
dbDisconnect(con)

# Connect to Athena using your profile name
# Profile name can be created by using AWS CLI
con <- dbConnect(RAthena::athena(),
profile_name = "YOUR_PROFILE_NAME",
s3_staging_dir = 's3://path/to/query/bucket/')
dbDisconnect(con)

# Connect to Athena using ARN role
con <- dbConnect(RAthena::athena(),
profile_name = "YOUR_PROFILE_NAME",
role_arn = "arn:aws:sts::123456789012:assumed-role/role_name/role_session_name",
s3_staging_dir = 's3://path/to/query/bucket/')

dbDisconnect(con)

## End(Not run)
```
**dbConvertTable**

**Description**

Utilises AWS Athena to convert AWS S3 backend file types. It also also to create more efficient file types i.e. "parquet" and "orc" from SQL queries.

**Usage**

```r
dbConvertTable(conn, obj, name, ...)
```

```
## S4 method for signature 'AthenaConnection'
dbConvertTable(
  conn,
  obj,
  name,
  partition = NULL,
  s3.location = NULL,
  file.type = c("NULL", "csv", "tsv", "parquet", "json", "orc"),
  compress = TRUE,
  data = TRUE,
  ...
)
```

**Arguments**

- `conn` An **AthenaConnection** object, produced by [DBI::dbConnect()]
- `obj` Athena table or SQL DML query to be converted. For SQL, the query need to be wrapped with [DBI::SQL()] and follow AWS Athena DML format link
- `name` Name of destination table
- `...` Extra parameters, currently not used
- `partition` Partition Athena table
- `s3.location` location to store output file, must be in s3 uri format for example ("s3://mybucket/data/").
- `file.type` File type for name, currently support ["NULL","csv", "tsv", "parquet", "json", "orc"]. "NULL" will let Athena set the file type for you.
- `compress` Compress name, currently can only compress ["parquet", "orc"] ([AWS Athena CTAS])
- `data` If name should be created with data or not.

**Value**

`dbConvertTable()` returns TRUE but invisible.
Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)
library(RAthena)

# Demo connection to Athena using profile name
con <- dbConnect(athena())

# write iris table to Athena in default delimited format
dbWriteTable(con, "iris", iris)

# convert delimited table to parquet
dbConvertTable(con,
  obj = "iris",
  name = "iris_parquet",
  file.type = "parquet")

# Create partitioned table from non-partitioned
# iris table using SQL DML query
dbConvertTable(con,
  obj = SQL("select
    iris.`,
    `date_format(current_date, '%Y%m%d') as time_stamp
    from iris"),
  name = "iris_orc_partitioned",
  file.type = "orc",
  partition = "time_stamp")

# disconnect from Athena
dbDisconnect(con)

## End(Not run)
```

---

**Description**

Returns a character string that describes the Athena SQL data type for the `obj` object.

**Usage**

```r
## S4 method for signature 'AthenaDriver,ANY'
dbDataType(dbObj, obj, ...)
```
dbDataType.AthenaDriver.ANY-method

## S4 method for signature 'AthenaDriver,list'
dbDataType(dbObj, obj, ...)

## S4 method for signature 'AthenaConnection,ANY'
dbDataType(dbObj, obj, ...)

## S4 method for signature 'AthenaConnection,data.frame'
dbDataType(dbObj, obj, ...)

Arguments

- `dbObj` A object inheriting from DBIDriver or DBIConnection
- `obj` An R object whose SQL type we want to determine.
- `...` Other arguments passed on to methods.

Value

dbDataType returns the Athena type that correspond to the obj argument as an non-empty character string.

See Also

dbDataType

Examples

```r
library(RAthena)
dbDataType(athena(), 1:5)
dbDataType(athena(), 1)
dbDataType(athena(), TRUE)
dbDataType(athena(), Sys.Date())
dbDataType(athena(), Sys.time())
dbDataType(athena(), c("x", "abc"))
dbDataType(athena(), list(raw(10), raw(20)))

vapply(iris, function(x) dbDataType(RAthena::athena(), x),
       FUN.VALUE = character(1), USE.NAMES = TRUE)

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Sending Queries to Athena
dbDataType(con, iris)
```
# Disconnect connection

dbDisconnect(con)

## End(Not run)

---

**dbDisconnect**  
*Disconnect (close) an Athena connection*

---

**Description**

This closes the connection to Athena.

**Usage**

```r
## S4 method for signature 'AthenaConnection'
dbDisconnect(conn, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>conn</code></td>
<td>A <code>DBIConnection</code> object, as returned by <code>dbConnect()</code>.</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Other parameters passed on to methods.</td>
</tr>
</tbody>
</table>

**Value**

`dbDisconnect()` returns `TRUE`, invisibly.

**See Also**

`dbDisconnect`

**Examples**

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demonstration connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Disconnect connection
dbDisconnect(con)

## End(Not run)
```
**dbExistsTable**

*Does Athena table exist?*

**Description**

Returns logical scalar if the table exists or not. TRUE if the table exists, FALSE otherwise.

**Usage**

```r
## S4 method for signature 'AthenaConnection,character'
dbExistsTable(conn, name, ...)
```

**Arguments**

- `conn`: A `DBIConnection` object, as returned by `dbConnect()`.  
  - The table name, passed on to `dbQuoteIdentifier()`. Options are:  
    - a character string with the unquoted DBMS table name, e.g. "table_name",  
    - a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`  
    - a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("my_schema."table_name")`  
  - Other parameters passed on to methods.

**Value**

`dbExistsTable()` returns logical scalar. TRUE if the table exists, FALSE otherwise.

**See Also**

`dbExistsTable`

**Examples**

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentmentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Write data.frame to Athena table
dbWriteTable(con, "mtcars", mtcars,  
  partition=c("TIMESTAMP" = format(Sys.Date(), "%Y%m%d")),  
  s3.location = "s3://mybucket/data/")
```
# Check if table exists from Athena
dbExistsTable(con, "mtcars")

# Disconnect connection
dbDisconnect(con)

## End(Not run)

---

**dbFetch**

*Fetch records from previously executed query*

---

**Description**

Currently returns the top n elements (rows) from result set or returns entire table from Athena.

**Usage**

```r
## S4 method for signature 'AthenaResult'
dbFetch(res, n = -1, ...)
```

**Arguments**

- `res`: An object inheriting from `DBIResult`, created by `dbSendQuery()`.
- `n`: Maximum number of records to retrieve per fetch. Use `n = -1` or `n = Inf` to retrieve all pending records. Some implementations may recognize other special values. Currently chunk sizes range from 0 to 999, if entire dataframe is required use `n = -1` or `n = Inf`.
- `...`: Other arguments passed on to methods.

**Value**

dbFetch() returns a data frame.

**See Also**

dbFetch

**Examples**

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())
```
```r
res <- dbSendQuery(con, "show databases")
dbFetch(res)
dbClearResult(res)

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)
```

---

**dbGetInfo**

**Get DBMS metadata**

**Description**

Get DBMS metadata

**Usage**

```r
## S4 method for signature 'AthenaConnection'
dbGetInfo(dbObj, ...)
## S4 method for signature 'AthenaResult'
dbGetInfo(dbObj, ...)
```

**Arguments**

- `dbObj` 
  An object inheriting from `DBIObjec`, i.e. `DBIDriver`, `DBIConnection`, or a `DBIResult`
- `...` 
  Other arguments to methods.

**Value**

A named list

**See Also**

`dbGetInfo`

**Examples**

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation
library(DBI)

# Demo connection to Athena using profile name
```
con <- dbConnect(RAthena::athena())

# Returns metadata from connection object
metadata <- dbGetInfo(con)

# Return metadata from Athena query object
res <- dbSendQuery(con, "show databases")
dbGetInfo(res)

# Clear result
dbClearResult(res)

# disconnect from Athena
dbDisconnect(con)

## End(Not run)

dbGetPartition

Athena table partitions

Description
This method returns all partitions from Athena table.

Usage

```
dbGetPartition(conn, name, ..., .format = FALSE)
```

```
## S4 method for signature 'AthenaConnection'
dbGetPartition(conn, name, ..., .format = FALSE)
```

Arguments

- **conn**
  A DBIConnection object, as returned by `dbConnect()`. Options are:
  - a character string with the unquoted DBMS table name, e.g. "table_name",
  - a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`
  - a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("my_schema"."table_name")`
- **name**
  The table name, passed on to `dbQuoteIdentifier()`. Options are:
  - a character string with the unquoted DBMS table name, e.g. "table_name",
  - a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`
  - a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("my_schema"."table_name")`
- **...**
  Other parameters passed on to methods.
- **.format**
  re-formats AWS Athena partitions format. So that each column represents a partition from the AWS Athena table. Default set to FALSE to prevent breaking previous package behaviour.
Value
data.frame that returns all partitions in table, if no partitions in Athena table then function will return error from Athena.

Examples
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation
library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# write iris table to Athena
dbWriteTable(con, "iris",
iris,
  partition = c("timestamp" = format(Sys.Date(), "%Y%m%d")),
  s3.location = "s3://path/to/store/athena/table/"
)

# return table partitions
RAthena::dbGetPartition(con, "iris")

# disconnect from Athena
dbDisconnect(con)

## End(Not run)

dbGetQuery

Send query, retrieve results and then clear result set

Description
Send query, retrieve results and then clear result set

Usage
## S4 method for signature 'AthenaConnection,character'
dbGetQuery(conn, statement, statistics = FALSE, unload = athena_unload(), ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conn</td>
<td>A DBIConnection object, as returned by dbConnect().</td>
</tr>
<tr>
<td>statement</td>
<td>a character string containing SQL.</td>
</tr>
<tr>
<td>statistics</td>
<td>If set to TRUE will print out AWS Athena statistics of query.</td>
</tr>
</tbody>
</table>
unload boolean input to modify 'statement' to align with AWS Athena UNLOAD, default is set to FALSE.
...
Other parameters passed on to methods.

Value

dbGetQuery() returns a dataframe.

Note

If the user does not have permission to remove AWS S3 resource from AWS Athena output location, then an AWS warning will be returned. For example AccessDenied (HTTP 403). Access Denied. It is better use query caching or optionally prevent clear AWS S3 resource using RAthena_options so that the warning doesn’t repeatedly show.

See Also

dbGetQuery

Examples

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena:::athena())

# Sending Queries to Athena
dbGetQuery(con, "show databases")

# Disconnect connection
dbDisconnect(con)

## End(Not run)
**dbGetTables**

**List Athena Schema, Tables and Table Types**

**Description**

Method to get Athena schema, tables and table types return as a data.frame

**Usage**

```r
dbGetTables(conn, ...) 
```

## S4 method for signature 'AthenaConnection'

```r
dbGetTables(conn, schema = NULL, ...)
```

**Arguments**

- **conn**
  - A DBIConnection object, as returned by `dbConnect()`.
- **...**
  - Other parameters passed on to methods.
- **schema**
  - Athena schema, default set to NULL to return all tables from all Athena schemas.
  - Note: The use of DATABASE and SCHEMA is interchangeable within Athena.

**Description**

Method to get Athena schema, tables and table types return as a data.frame

**Usage**

```r
dbGetTables(conn, ...) 
```

## S4 method for signature 'AthenaConnection'

```r
dbGetTables(conn, schema = NULL, ...)
```

**Arguments**

- **conn**
  - A DBIConnection object, as returned by `dbConnect()`.
- **...**
  - Other parameters passed on to methods.
- **schema**
  - Athena schema, default set to NULL to return all tables from all Athena schemas.
  - Note: The use of DATABASE and SCHEMA is interchangeable within Athena.
\textit{dbGetTables()} returns a \texttt{data.frame}.

\section*{Examples}

\begin{verbatim}
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)
library(RAthena)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Return hierarchy of tables in Athena
dbGetTables(con)

# Disconnect connection
dbDisconnect(con)

## End(Not run)
\end{verbatim}

\begin{tabular}{ll}
\textbf{dbHasCompleted} & \textit{Completion status} \\
\hline

\end{tabular}

\section*{Description}

This method returns if the query has completed.

\section*{Usage}

```
## S4 method for signature 'AthenaResult'
dbHasCompleted(res, ...)
```

\section*{Arguments}

\begin{itemize}
\item \texttt{res} \hspace{1cm} \texttt{An object inheriting from DBIResult.}
\item \texttt{...} \hspace{1cm} \texttt{Other arguments passed on to methods.}
\end{itemize}

\section*{Value}

\texttt{dbHasCompleted()} returns a logical scalar. \texttt{TRUE} if the query has completed, \texttt{FALSE} otherwise.

\section*{See Also}

\texttt{dbHasCompleted}
dbIsValid

Is this DBMS object still valid?

Description

This method tests whether the dbObj is still valid.

Usage

## S4 method for signature 'AthenaConnection'
dbIsValid(dbObj, ...)

## S4 method for signature 'AthenaResult'
dbIsValid(dbObj, ...)

Arguments

dbObj An object inheriting from DBIObjec, i.e. DBIDriver, DBIConnection, or a DBIResult

... Other arguments to methods.

Value

dbIsValid() returns logical scalar, TRUE if the object (dbObj) is valid, FALSE otherwise.
See Also

dbIsValid

Examples

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Check is connection is valid
dbIsValid(con)

# Check is query is valid
res <- dbSendQuery(con, "show databases")
dbIsValid(res)

# Check if query is valid after clearing result
dbClearResult(res)
dbIsValid(res)

# Check if connection if valid after closing connection
dbDisconnect(con)
dbIsValid(con)

## End(Not run)

---

dbListFields  List Field names of Athena table

Description

List Field names of Athena table

Usage

## S4 method for signature 'AthenaConnection,character'
dbListFields(conn, name, ...)

Arguments

conn  A DBIConnection object, as returned by dbConnect().
name  The table name, passed on to dbQuoteIdentifier(). Options are:
dbListTables

- a character string with the unquoted DBMS table name, e.g. "table_name",
- a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`
- a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("'my_schema'."table_name")`

... Other parameters passed on to methods.

Value

dbListFields() returns a character vector with all the fields from an Athena table.

See Also

dbListFields

Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Write data.frame to Athena table
dbWriteTable(con, "mtcars", mtcars, 
  partition=c("TIMESTAMP" = format(Sys.Date(), "%Y%m%d"), 
              s3.location = "s3://mybucket/data/"))

# Return list of fields in table
dbListFields(con, "mtcars")

# Disconnect connection
dbDisconnect(con)

## End(Not run)
```

---

dbListTables

List Athena Tables

Description

Returns the unquoted names of Athena tables accessible through this connection.
Usage

```r
## S4 method for signature 'AthenaConnection'
dblListTables(conn, schema = NULL, ...)
```

Arguments

- `conn`: A `DBIConnection` object, as returned by `dbConnect()`.
- `schema`: Athena schema, default set to NULL to return all tables from all Athena schemas. Note: The use of DATABASE and SCHEMA is interchangeable within Athena.
- `...`: Other parameters passed on to methods.

Value

dblListTables() returns a character vector with all the tables from Athena.

See Also

dblListTables

Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation
library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Return list of tables in Athena
dblListTables(con)

# Disconnect conenction
dbDisconnect(con)

## End(Not run)
```

---

**dbplyr**

Declare which version of dbplyr API is being called.

Description

Declare which version of dbplyr API is being called.

Usage

```r
dbplyr_edition.AthenaConnection(con)
```
**dbQuote**

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>con</code></td>
<td>A <code>dbConnect</code> object, as returned by <code>dbConnect()</code></td>
</tr>
</tbody>
</table>

### Value

Integer for which version of `dbplyr` is going to be used.

---

**dbQuote**

### Quote Identifiers

**Description**

Call this method to generate string that is suitable for use in a query as a column or table name.

**Usage**

```r
## S4 method for signature 'AthenaConnection,character'
dbQuoteString(conn, x, ...)

## S4 method for signature 'AthenaConnection,POSIXct'
dbQuoteString(conn, x, ...)

## S4 method for signature 'AthenaConnection,Date'
dbQuoteString(conn, x, ...)

## S4 method for signature 'AthenaConnection,SQL'
dbQuoteIdentifier(conn, x, ...)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>conn</code></td>
<td>A <code>DBIConnection</code> object, as returned by <code>dbConnect()</code></td>
</tr>
<tr>
<td><code>x</code></td>
<td>A character vector to quote as string.</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Other arguments passed on to methods.</td>
</tr>
</tbody>
</table>

### Value

Returns a character object, for more information please check out: `dbQuoteString`, `dbQuoteIdentifier`

### See Also

`dbQuoteString`, `dbQuoteIdentifier`
dbRemoveTable  

Remove table from Athena

Description

Removes Athena table but does not remove the data from Amazon S3 bucket.

Usage

```r
## S4 method for signature 'AthenaConnection,character'
dbRemoveTable(conn, name, delete_data = TRUE, confirm = FALSE, ...)
```

Arguments

- **conn**: A DBIConnection object, as returned by `dbConnect()`.  
- **name**: The table name, passed on to `dbQuoteIdentifier()`. Options are:
  - a character string with the unquoted DBMS table name, e.g. "table_name",  
  - a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`  
  - a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("'my_schema"."table_name")`  
- **delete_data**: Deletes S3 files linking to AWS Athena table  
- **confirm**: Allows for S3 files to be deleted without the prompt check. It is recommend to leave this set to `FALSE` to avoid deleting other S3 files when the table's definition points to the root of S3 bucket.  
- **...**: Other parameters passed on to methods.

Value

`dbRemoveTable()` returns `TRUE`, invisibly.

Note

If you are having difficulty removing AWS S3 files please check if the AWS S3 location following AWS best practises: Table Location in Amazon S3

See Also

`dbRemoveTable`
Examples

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Write data.frame to Athena table
dbWriteTable(con, "mtcars", mtcars,
  partition=c("TIMESTAMP" = format(Sys.Date(), "%Y%m%d")),
  s3.location = "s3://mybucket/data/")

# Remove Table from Athena
dbRemoveTable(con, "mtcars")

# Disconnect connection
dbDisconnect(con)

## End(Not run)

---

**dbShow**  
*Show Athena table’s DDL*

**Description**

Executes a statement to return the data description language (DDL) of the Athena table.

**Usage**

`dbShow(conn, name, ...)`

## S4 method for signature 'AthenaConnection'
`dbShow(conn, name, ...)`

**Arguments**

- `conn`  
  A `DBIConnection` object, as returned by `dbConnect()`.

- `name`  
  The table name, passed on to `dbQuoteIdentifier()`. Options are:
  - a character string with the unquoted DBMS table name, e.g. "table_name",
  - a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`
  - a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("my_schema"."table_name")`

- `...`  
  Other parameters passed on to methods.
Value

\texttt{dbShow()} returns SQL characters of the Athena table DDL.

Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# write iris table to Athena
dbWriteTable(con, "iris",
iris,
  partition = c("timestamp" = format(Sys.Date(), "%Y%m%d")),
  s3.location = "s3://path/to/store/athena/table/"
)

# return table ddl
RAthena::dbShow(con, "iris")

# disconnect from Athena
dbDisconnect(con)

## End(Not run)
```

\textbf{dbStatistics} \textit{Show AWS Athena Statistics}

\textbf{Description}

Returns AWS Athena Statistics from execute queries \texttt{dbSendQuery}

\textbf{Usage}

\texttt{dbStatistics(res, ...)}

\texttt{## S4 method for signature 'AthenaResult'
  dbStatistics(res, ...)}

\textbf{Arguments}

- \texttt{res} \hspace{1cm} An object inheriting from \texttt{DBIResult}.
- \texttt{...} \hspace{1cm} Other arguments passed on to methods.
Value

`dbStatistics()` returns list containing Athena Statistics return from boto3.

Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)
library(RAthena)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

res <- dbSendQuery(con, "show databases")
dbStatistics(res)

# Clean up
dbClearResult(res)
```

### S3 implementation of `db_compute` for Athena

**Description**

This is a backend function for dplyr’s `compute` function. Users won't be required to access and run this function.

**Usage**

```r
db_compute.AthenaConnection(con, table, sql, ...)
```

**Arguments**

- `con` A `dbConnect` object, as returned by `dbConnect()`
- `table` Table name, if left default RAthena will use the default from dplyr’s `compute` function.
- `sql` SQL code to be sent to the data
- `...` passes RAthena table creation parameters: `file_type`, `s3_location`, `partition`
  - `file_type`: What file type to store data.frame on s3, RAthena currently supports ['"NULL","csv", "parquet", "json"]). "NULL" will let Athena set the file_type for you.
• `s3_location`: s3 bucket to store Athena table, must be set as a s3 uri for example ("s3://mybucket/data")
• `partition`: Partition Athena table, requires to be a partitioned variable from previous table.

**Value**

db_compute returns table name

**See Also**

AthenaWriteTables backend_dbplyr_v2 backend_dbplyr_v1

**Examples**

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)
library(dplyr)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Write data.frame to Athena table
copy_to(con, mtcars, 
    s3_location = "s3://mybucket/data/"
)

# Write Athena table from tbl_sql
athena_mtcars <- tbl(con, "mtcars")
mtcars_filter <- athena_mtcars %>% filter(gear >= 4)

# create athena with unique table name
mtcars_filter %>%
    compute()

# create athena with specified name and s3 location
mtcars_filter %>%
    compute("mtcars_filter", 
        s3_location = "s3://mybucket/mtcars_filer/"
)

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)
```
db_connection_describe

S3 implementation of `db_connection_describe` for Athena (api version 2).

Description

This is a backend function for `dplyr` to retrieve metadata about Athena queries. Users won’t be required to access and run this function.

Usage

```r
db_connection_describe.AthenaConnection(con)
```

Arguments

- `con` A `dbConnect` object, as returned by `dbConnect()`

Value

Character variable containing metadata about query sent to Athena. The metadata is returned in the following format:

```
"Athena <boto3 version> [<profile_name>@region/database]"
```

db_copy_to

S3 implementation of `db_copy_to` for Athena

Description

This is an Athena method for dbplyr function `db_copy_to` to create an Athena table from a data.frame.

Usage

```r
db_copy_to.AthenaConnection(
  con,
  table,
  values,
  overwrite = FALSE,
  append = FALSE,
  types = NULL,
  partition = NULL,
  s3_location = NULL,
  file_type = c("csv", "tsv", "parquet"),
  compress = FALSE,
  max_batch = Inf,
  ...
)
```
Arguments

con A \texttt{dbConnect} object, as returned by \texttt{dbConnect()}

table A character string specifying a table name. Names will be automatically quoted so you can use any sequence of characters, not just any valid bare table name.

dataframe A data.frame to write to the database.

overwrite Allows overwriting the destination table. Cannot be \texttt{TRUE} if \texttt{append} is also \texttt{TRUE}.

append Allow appending to the destination table. Cannot be \texttt{TRUE} if \texttt{overwrite} is also \texttt{TRUE}. Existing Athena DDL file type will be retained and used when uploading data to AWS Athena. If parameter \texttt{file.type} doesn’t match AWS Athena DDL file type a warning message will be created notifying user and RAthena will use the file type for the Athena DDL.

types Additional field types used to override derived types.

partition Partition Athena table (needs to be a named list or vector) for example: \texttt{c(var1 = "2019-02-13")}

s3_location s3 bucket to store Athena table, must be set as a s3 uri for example \texttt{"s3://mybucket/data/"}

file_type What file type to store data.frame on s3, RAthena currently supports \{"tsv", "csv", "parquet"\}. Default delimited file type is "tsv", in previous versions of RAthena (<= 1.4.0) file type "csv" was used as default. The reason for the change is that columns containing \texttt{Array/JSON} format cannot be written to Athena due to the separating value ".". This would cause issues with AWS Athena. \textbf{Note}: "parquet" format is supported by the \texttt{arrow} package and it will need to be installed to utilise the "parquet" format.

compress \texttt{FALSE} | \texttt{TRUE} To determine if to compress \texttt{file.type}. If \texttt{file.type} is \{"csv", "tsv"\} then "gzip" compression is used, for \texttt{file.type} "parquet" "snappy" compression is used.

max_batch Split the data frame by max number of rows i.e. 100,000 so that multiple files can be uploaded into AWS S3. By default when compression is set to \texttt{TRUE} and \texttt{file.type} is "csv" or "tsv" \texttt{max.batch} will split data.frame into 20 batches. This is to help the performance of AWS Athena when working with files compressed in "gzip" format. \texttt{max.batch} will not split the data.frame when loading file in parquet format. For more information please go to \texttt{link}.

Value

db_copy_to returns table name

See Also

\texttt{AthenaWriteTables}

Examples

\texttt{## Not run:}
\texttt{# Note:}
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)
library(dplyr)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# List existing tables in Athena
dbListTables(con)

# Write data.frame to Athena table
copy_to(con, mtcars, 
    s3_location = "s3://mybucket/data/"
)

# Checking if uploaded table exists in Athena
dbExistsTable(con, "mtcars")

# Write Athena table from tbl_sql
athena_mtcars <- tbl(con, "mtcars")
mtcars_filter <- athena_mtcars %>% filter(gear >=4)
copy_to(con, mtcars_filter)

# Checking if uploaded table exists in Athena
dbExistsTable(con, "mtcars_filter")

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)

---

### db_desc

*S3 implementation of db_desc for Athena (api version 1).*

**Description**

This is a backend function for dplyr to retrieve meta data about Athena queries. Users won’t be required to access and run this function.

**Usage**

`db_desc.AthenaConnection(x)`

**Arguments**

- `x` A `dbConnect` object, as returned by `dbConnect()`
install_boto

**Value**

Character variable containing Meta Data about query sent to Athena. The Meta Data is returned in the following format:

"Athena <boto3 version> [<profile_name>@region/database]"

---

**install_boto** *Install Amazon SDK boto3 for Athena connection*

**Description**

Install Amazon SDK boto3 for Athena connection

**Usage**

```
install_boto(
  method = c("auto", "virtualenv", "conda"),
  conda = "auto",
  envname = "R Athena",
  conda_python_version = "3.7",
  ...
)
```

**Arguments**

- **method**: Installation method. By default, "auto" automatically finds a method that will work in the local environment. Change the default to force a specific installation method. Note that the "virtualenv" method is not available on Windows. Note also that since this command runs without privilege the "system" method is available only on Windows.
- **conda**: The path to a conda executable. Use "auto" to allow reticulate to automatically find an appropriate conda binary. See **Finding Conda** and `conda_binary()` for more details.
- **envname**: Name of Python environment to install within, by default environment name RAthena.
- **conda_python_version**: the python version installed in the created conda environment. Python 3.7 is installed by default.
- ... other arguments passed to `reticulate::conda_install()` or `reticulate::virtualenv_install()`.

**Value**

Returns NULL after installing Python Boto3.

**Note**

[reticulate::use_python] or [reticulate::use_condaenv] might be required before connecting to Athena.
**Description**

The `dbSendQuery()` and `dbSendStatement()` method submits a query to Athena but does not wait for query to execute. `dbHasCompleted` method will need to ran to check if query has been completed or not. The `dbExecute()` method submits a query to Athena and waits for the query to be executed.

**Usage**

```r
## S4 method for signature 'AthenaConnection,character'
dbSendQuery(conn, statement, unload = athena_unload(), ...)

## S4 method for signature 'AthenaConnection,character'
dbSendStatement(conn, statement, unload = athena_unload(), ...)

## S4 method for signature 'AthenaConnection,character'
dbExecute(conn, statement, unload = athena_unload(), ...)
```

**Arguments**

- `conn` A `DBIConnection` object, as returned by `dbConnect()`.
- `statement` a character string containing SQL.
- `unload` boolean input to modify `statement` to align with AWS Athena UNLOAD, default is set to `FALSE`.
- `...` Other parameters passed on to methods.

**Value**

Returns `AthenaResult` s4 class.

**See Also**

`dbSendQuery`, `dbSendStatement`, `dbExecute`

**Examples**

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(DBI)

# Demo connection to Athena using profile name
```
RAthena_options

con <- dbConnect(RAthena::athena())

# Sending Queries to Athena
res1 <- dbSendQuery(con, "show databases")
res2 <- dbSendStatement(con, "show databases")
res3 <- dbExecute(con, "show databases")

# Disconnect connection
dbDisconnect(con)

## End(Not run)

RAthena_options A method to configure RAthena backend options.

Description

RAthena_options() provides a method to change the backend. This includes changing the file parser, whether RAthena should cache query ids locally and number of retries on a failed api call.

Usage

RAthena_options(
  file_parser,
  bigint,
  binary,
  json,
  cache_size,
  clear_cache,
  retry,
  retry_quiet,
  unload,
  clear_s3_resource,
  verbose
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_parser</td>
<td>Method to read and write tables to Athena, currently default to &quot;data.table&quot;. The file_parser also determines the data format returned for example &quot;data.table&quot; will return data.table and &quot;vroom&quot; will return tibble.</td>
</tr>
<tr>
<td>bigint</td>
<td>The R type that 64-bit integer types should be mapped to (default: &quot;integer64&quot;). Inbuilt bigint conversion types [&quot;integer64&quot;, &quot;integer&quot;, &quot;numeric&quot;, &quot;character&quot;].</td>
</tr>
<tr>
<td>binary</td>
<td>The R type that [binary/varbinary] types should be mapped to (default &quot;raw&quot;). Inbuilt binary conversion types [&quot;raw&quot;, &quot;character&quot;].</td>
</tr>
</tbody>
</table>
json

Attempt to converts AWS Athena data types [arrays, json] using `jsonlite::parse_json` (default: "auto"). Inbuilt json conversion types ["auto", "character"]. Custom Json parsers can be provide by using a function with data frame parameter.

cache_size

Number of queries to be cached. Currently only support caching up to 100 distinct queries (default: 0).

clear_cache

Clears all previous cached query metadata

retry

Maximum number of requests to attempt (default: 5).

retry_quiet

This method is deprecated please use verbose instead.

unload

set AWS Athena unload functionality globally (default: FALSE)

clear_s3_resource

Clear down ‘AWS Athena‘ ‘AWS S3‘ resource (s3_staging_dir location). This is useful for users that don’t have the ‘AWS IAM role’ permissions delete from ‘s3_staging_dir‘ (default: TRUE)

verbose

print package info messages (default: TRUE)

Value

RAthena_options() returns NULL, invisibly.

Examples

library(RAthena)

# change file parser from default data.table to vroom
RAthena_options("vroom")

# cache queries locally
RAthena_options(cache_size = 5)

session_token

Get Session Tokens for Boto3 Connection

Description

Returns a set of temporary credentials for an AWS account or IAM user (link).

Usage

get_session_token(  
  profile_name = NULL,  
  region_name = NULL,  
  serial_number = NULL,  
  token_code = NULL,  
  duration_seconds = 3600L,  
  set_env = FALSE  
)
Arguments

profile_name  The name of a profile to use. If not given, then the default profile is used. To set profile name, the AWS Command Line Interface (AWS CLI) will need to be configured. To configure AWS CLI please refer to: Configuring the AWS CLI.

region_name  Default region when creating new connections. Please refer to link for AWS region codes (region code example: Region = EU (Ireland) region_name = "eu-west-1")

serial_number  The identification number of the MFA device that is associated with the IAM user who is making the GetSessionToken call. Specify this value if the IAM user has a policy that requires MFA authentication. The value is either the serial number for a hardware device (such as ‘GAHT12345678’) or an Amazon Resource Name (ARN) for a virtual device (such as arn:aws:iam::123456789012:mfa/user).

token_code  The value provided by the MFA device, if MFA is required. If any policy requires the IAM user to submit an MFA code, specify this value. If MFA authentication is required, the user must provide a code when requesting a set of temporary security credentials. A user who fails to provide the code receives an "access denied" response when requesting resources that require MFA authentication.

duration_seconds  The duration, in seconds, that the credentials should remain valid. Acceptable duration for IAM user sessions range from 900 seconds (15 minutes) to 129,600 seconds (36 hours), with 3,600 seconds (1 hour) as the default.

set_env  If set to TRUE environmental variables AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY and AWS_SESSION_TOKEN will be set.

Value

get_session_token() returns a list containing: "AccessKeyId", "SecretAccessKey", "SessionToken" and "Expiration"

Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.

library(RAthena)
library(DBI)

# Create Temporary Credentials duration 1 hour
get_session_token("YOUR_PROFILE_NAME",
    serial_number='arn:aws:iam::123456789012:mfa/user',
    token_code = "531602",
    set_env = TRUE)

# Connect to Athena using temporary credentials
con <- dbConnect(athena())

## End(Not run)```
sqlCreateTable

Creates query to create a simple Athena table

Description

Creates an interface to compose CREATE EXTERNAL TABLE.

Usage

```r
## S4 method for signature 'AthenaConnection'
sqlCreateTable(
  con,
  table,
  fields,
  field.types = NULL,
  partition = NULL,
  s3.location = NULL,
  file.type = c("tsv", "csv", "parquet", "json"),
  compress = FALSE,
  ...
)
```

Arguments

- **con**: A database connection.
- **table**: The table name, passed on to `dbQuoteIdentifier()`. Options are:
  - a character string with the unquoted DBMS table name, e.g. "table_name".
  - a call to `Id()` with components to the fully qualified table name, e.g. `Id(schema = "my_schema", table = "table_name")`
  - a call to `SQL()` with the quoted and fully qualified table name given verbatim, e.g. `SQL("my_schema"."table_name")`
- **fields**: Either a character vector or a data frame.
  - A named character vector: Names are column names, values are types. Names are escaped with `dbQuoteIdentifier()`. Field types are unescaped.
  - A data frame: field types are generated using `dbDataType()`.
- **field.types**: Additional field types used to override derived types.
- **partition**: Partition Athena table (needs to be a named list or vector) for example: `c(var1 = "2019-20-13")`
- **s3.location**: s3 bucket to store Athena table, must be set as a s3 uri for example ("s3://mybucket/data/"). By default `s3.location` is set s3 staging directory from `AthenaConnection` object.
- **file.type**: What file type to store data.frame on s3. IRAthena currently supports ["tsv", "csv", "parquet", "json"]). Default delimited file type is "tsv", in previous versions of IRAthena (<= 1.6.0) file type "csv" was used as default. The reason
for the change is that columns containing Array/JSON format cannot be written to Athena due to the separating value ",". This would cause issues with AWS Athena. Note: "parquet" format is supported by the arrow package and it will need to be installed to utilise the "parquet" format. "json" format is supported by jsonlite package and it will need to be installed to utilise the "json" format.

Value

sqlCreateTable returns data.frame’s DDL in the SQL format.

See Also

sqlCreateTable

Examples

## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see \`R\:dbConnect\` documentation

library(DBI)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena::athena())

# Create DDL for iris data.frame
sqlCreateTable(con, "iris", iris, s3.location = "s3://path/to/athena/table")

# Create DDL for iris data.frame with partition
sqlCreateTable(con, "iris", iris, 
                 partition = "timestamp", 
                 s3.location = "s3://path/to/athena/table")

# Create DDL for iris data.frame with partition and file.type parquet
sqlCreateTable(con, "iris", iris, 
                 partition = "timestamp", 
                 s3.location = "s3://path/to/athena/table", 
                 file.type = "parquet")

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)
sqlData

Converts data frame into suitable format to be uploaded to Athena

Description

This method converts data.frame columns into the correct format so that it can be uploaded Athena.

Usage

```r
## S4 method for signature 'AthenaConnection'
sqlData(
  con,
  value,
  row.names = NA,
  file.type = c("tsv", "csv", "parquet", "json"),
  ...  
)
```

Arguments

- `con` A database connection.
- `value` A data frame
- `row.names` Either TRUE, FALSE, NA or a string. If TRUE, always translate row names to a column called "row_names". If FALSE, never translate row names. If NA, translate rownames only if they’re a character vector. A string is equivalent to TRUE, but allows you to override the default name. For backward compatibility, NULL is equivalent to FALSE.
- `file.type` What file type to store data.frame on s3. RAthena currently supports ["csv", "tsv", "parquet", "json"]. Note: This parameter is used for format any special characters that clash with file type separator.
- `...` Other arguments used by individual methods.

Value

sqlData returns a dataframe formatted for Athena. Currently converts list variable types into character split by '|', similar to how data.table writes out to files.

See Also

sqlData
Description

Create s3 implementation of sql_translate_env for AWS Athena sql translate environment based off Athena Data Types and DML Queries, Functions, and Operators

Usage

sql_translation.AthenaConnection(con)
sql_translate_env.AthenaConnection(con)
sql_escape_string.AthenaConnection(con, x)

Arguments

con An AthenaConnection object, produced by [DBI::dbConnect()]
x An object to escape. Existing sql vectors will be left as is, character vectors are escaped with single quotes, numeric vectors have trailing `.0` added if they’re whole numbers, identifiers are escaped with double quotes.

Description

Lower level API access, allows user to create and delete Athena Work Groups.

create_work_group Creates a workgroup with the specified name (link). The work group utilises parameters from the dbConnect object, to determine the encryption and output location of the work group. The s3_staging_dir, encryption_option and kms_key parameters are gotten from dbConnect
tag_options Helper function to create tag options for function create_work_group()
delete_work_group Deletes the workgroup with the specified name (link). The primary workgroup cannot be deleted.
list_work_groups Lists available workgroups for the account (link).
get_work_group Returns information about the workgroup with the specified name (link).
update_work_group Updates the workgroup with the specified name (link). The workgroup’s name cannot be changed. The work group utilises parameters from the dbConnect object, to determine the encryption and output location of the work group. The s3_staging_dir, encryption_option and kms_key parameters are gotten from dbConnect
work_group

Usage

create_work_group(
  conn,
  work_group = NULL,
  enforce_work_group_config = FALSE,
  publish_cloud_watch_metrics = FALSE,
  bytes_scanned_cut_off = 1000000L,
  requester_pays = FALSE,
  description = NULL,
  tags = tag_options(key = NULL, value = NULL)
)

tag_options(key = NULL, value = NULL)

delete_work_group(conn, work_group = NULL, recursive_delete_option = FALSE)

list_work_groups(conn)

get_work_group(conn, work_group = NULL)

update_work_group(
  conn,
  work_group = NULL,
  remove_output_location = FALSE,
  enforce_work_group_config = FALSE,
  publish_cloud_watch_metrics = FALSE,
  bytes_scanned_cut_off = 1000000L,
  requester_pays = FALSE,
  description = NULL,
  state = c("ENABLED", "DISABLED")
)

Arguments

conn          A dbConnect object, as returned by dbConnect()
work_group    The Athena workgroup name.
enforce_work_group_config
              If set to TRUE, the settings for the workgroup override client-side settings. If set
to FALSE, client-side settings are used. For more information, see Workgroup Settings Override Client-Side Settings.
publish_cloud_watch_metrics
              Indicates that the Amazon CloudWatch metrics are enabled for the workgroup.
bytes_scanned_cut_off
              The upper data usage limit (cutoff) for the amount of bytes a single query in a
workgroup is allowed to scan.
requester_pays
              If set to TRUE, allows members assigned to a workgroup to reference Amazon
S3 Requester Pays buckets in queries. If set to FALSE, workgroup members
cannot query data from Requester Pays buckets, and queries that retrieve data from Requester Pays buckets cause an error. The default is false. For more information about Requester Pays buckets, see Requester Pays Buckets in the Amazon Simple Storage Service Developer Guide.

description The workgroup description.

tags A tag that you can add to a resource. A tag is a label that you assign to an AWS Athena resource (a workgroup). Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize workgroups in Athena, for example, by purpose, owner, or environment. Use a consistent set of tag keys to make it easier to search and filter workgroups in your account. The maximum tag key length is 128 Unicode characters in UTF-8. The maximum tag value length is 256 Unicode characters in UTF-8. You can use letters and numbers representable in UTF-8, and the following characters: "+ - = . _ : / @". Tag keys and values are case-sensitive. Tag keys must be unique per resource. Please use the helper function tag_options() to create tags for work group, if no tags are required please put NULL for this parameter.

key A tag key. The tag key length is from 1 to 128 Unicode characters in UTF-8. You can use letters and numbers representable in UTF-8, and the following characters: "+ - = . _ : / @". Tag keys are case-sensitive and must be unique per resource.

value A tag value. The tag value length is from 0 to 256 Unicode characters in UTF-8. You can use letters and numbers representable in UTF-8, and the following characters: "+ - = . _ : / @". Tag values are case-sensitive.

recursive_delete_option The option to delete the workgroup and its contents even if the workgroup contains any named queries

remove_output_location If set to TRUE, indicates that the previously-specified query results location (also known as a client-side setting) for queries in this workgroup should be ignored and set to null. If set to FALSE the output location in the workgroup's result configuration will be updated with the new value. For more information, see Workgroup Settings Override Client-Side Settings.

state The workgroup state that will be updated for the given workgroup.

Value

create_work_group Returns NULL but invisible
tag_options Returns list but invisible
delete_work_group Returns NULL but invisible
list_work_groups Returns list of available work groups
get_work_group Returns list of work group meta data
update_work_group Returns NULL but invisible
Examples

```r
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `RAthena::dbConnect` documentation

library(RAthena)

# Demo connection to Athena using profile name
con <- dbConnect(RAthena:::athena())

# List current work group available
list_work_groups(con)

# Create a new work group
wg <- create_work_group(con,
                        "demo_work_group",
                        description = "This is a demo work group",
                        tags = tag_options(key = "demo_work_group", value = "demo_01"))

# List work groups to see new work group
list_work_groups(con)

# get meta data from work group
wg <- get_work_group(con, "demo_work_group")

# Update work group
wg <- update_work_group(con, "demo_work_group",
                         description = "This is a demo work group update")

# get updated meta data from work group
wg <- get_work_group(con, "demo_work_group")

# Delete work group
delete_work_group(con, "demo_work_group")

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)
```
Index

assume_role, 4
athena, 5
AthenaConnection, 7, 17, 47, 50
AthenaWriteTables, 6, 38, 40

backend_dbplyr_v1, 9, 38
backend_dbplyr_v2, 10, 38

conda_binary(), 42
create_work_group(work_group), 50

db_compute, 37
db_connection Describe, 39
db_copy_to, 39
db_desc, 41
db_explain.AthenaConnection
  (backend_dbplyr_v1), 9
db_query_fields.AthenaConnection
  (backend_dbplyr_v1), 9
dbClearResult, 11
dbClearResult, AthenaResult-method
  (dbClearResult), 11
dbColumnInfo, 12
dbColumnInfo, AthenaResult-method
  (dbColumnInfo), 12
dbConnect, 5, 9, 10, 16, 33, 37, 39–41, 50, 51
dbConnect
  (dbConnect, AthenaDriver-method), 13
  (dbConnect), 20, 21, 24, 25, 27, 30, 32–35, 43
dbConnect, AthenaDriver-method, 13
dbConvertTable, 17
dbConvertTable, AthenaConnection-method
  (dbConvertTable), 17
dbDataType, 19
dbDataType
  (dbDataType, AthenaDriver, ANY-method), 18
  (dbDataType), 47
dbDataType, AthenaConnection, ANY-method
  (dbDataType, AthenaDriver, ANY-method), 18
dbDataType, AthenaConnection, data.frame-method
  (dbDataType, AthenaDriver, ANY-method), 18
dbDataType, AthenaDriver, ANY-method, 18
dbDataType, AthenaDriver, list-method
  (dbDataType, AthenaDriver, ANY-method), 18
dbDisconnect, 20, 20
dbDisconnect, AthenaConnection-method
  (dbDisconnect), 20
dbExecute, 43
dbExecute, AthenaConnection, character-method
  (Query), 43
dbExistsTable, 21, 21
dbExistsTable, AthenaConnection, character-method
  (dbExistsTable), 21
dbFetch, 22, 22
dbFetch, AthenaResult-method (dbFetch), 22
dbGetInfo, 23, 23
dbGetInfo, AthenaConnection-method
  (dbGetInfo), 23
dbGetInfo, AthenaResult-method
  (dbGetInfo), 23
dbGetPartition, 24
dbGetPartition, AthenaConnection-method
  (dbGetPartition), 24
dbGetQuery, 25, 26
dbGetQuery, AthenaConnection, character-method
  (dbGetQuery), 25
dbGetStatement, 26, 27
dbGetStatement, AthenaConnection-method
  (dbGetStatement), 26
dbGetTables, 27
dbGetTables, AthenaConnection-method
  (dbGetTables), 27
INDEX

dbHasCompleted, 12, 28, 28, 43
dbHasCompleted, AthenaResult-method
(dbHasCompleted), 28
DBIConnection, 14, 19–21, 23–25, 27, 29, 30,
32–33, 43
DBIDriver, 14, 19, 23, 29
DBIObject, 23, 29
dbIsValid, 11, 12, 29, 30

dbIsValid, AthenaConnection-method
(dbIsValid), 29
dbIsValid, AthenaResult-method
(dbIsValid), 29
dbListFields, 30, 31
dbListFields, AthenaConnection, character-method
(dbListFields), 30
dbListTables, 31, 32
dbListTables, AthenaConnection-method
(dbListTables), 31
dbplyr_edition, 32
dbQuote, 33
dbQuoteIdentifier, 33
dbQuoteIdentifier(), 21, 24, 34, 35, 47
dbQuoteString, 33
dbQuoteString, AthenaConnection, character-method
(dbQuote), 33
dbQuoteString, AthenaConnection, Date-method
(dbQuote), 33
dbQuoteString, AthenaConnection, POSIXct-method
(dbQuote), 33
dbRemoveTable, 34, 34
dbRemoveTable, AthenaConnection, character-method
(dbRemoveTable), 34
dbSendQuery, 36, 43
dbSendQuery(), 22
dbSendQuery(), AthenaConnection, character-method
(Query), 43
dbSendStatement, 43
dbSendStatement, AthenaConnection, character-method
(Query), 43
dbShow, 35
dbShow, AthenaConnection-method
(dbShow), 35
dbStatistics, 36
dbStatistics, AthenaResult-method
(dbStatistics), 36
dbWriteTable, 8
dbWriteTable, AthenaConnection, character, data.frame-method
(AthenaWriteTables), 6
dbWriteTable, AthenaConnection, Id, data.frame-method
(AthenaWriteTables), 6
dbWriteTable, AthenaConnection, SQL, data.frame-method
(AthenaWriteTables), 6
delete_work_group (work_group), 50
get_session_token (session_token), 45
get_work_group (work_group), 50
Id(), 21, 24, 31, 34, 35, 47
install_boto, 42
RAthena, RAthena-package, 3
RAthena-package, 3
RAthena_options, 11, 26, 44
session_token, 45
SQL, 36, 48
sql_escape_date, AthenaConnection
(sql_translate_env), 10
sql_escape_string, AthenaConnection
(sql_translate_env), 50
sql_query_explain, AthenaConnection
(sql_translate_env), 10
sql_query_fields, AthenaConnection
(sql_translate_env), 50
sql_translation, AthenaConnection
(sql_translate_env), 50
sqlCreateTable, 47, 48
sqlCreateTable, AthenaConnection-method
(sqlCreateTable), 47
sqlData, 49, 49
sqlData, AthenaConnection-method
(sqlData), 49
tag_options, (work_group), 50
update_work_group, (work_group), 50
work_group, 50