

# Package ‘RAthena’

September 23, 2021

**Type** Package

**Title** Connect to 'AWS Athena' using 'Boto3' ('DBI' Interface)

**Version** 2.2.0

**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** 7.1.2

**URL** <https://github.com/DyfanJones/RAthena>

**BugReports** <https://github.com/DyfanJones/RAthena/issues>

**Collate** 'RAthena.R' 'Driver.R' 'Connection.R' 'DataTypes.R' 'File\_Parser.R' 'Options.R' 'Result.R' 'View.R' 'athena\_low\_api.R' 'column\_parser.R' 'dplyr\_integration.R' 'install.R' 'sql\_translate\_env.R' 'table.R' 'util.R' 'zzz.R'

**NeedsCompilation** no

**Author** Dyfan Jones [aut, cre]

**Maintainer** Dyfan Jones <[dyfan.r.jones@gmail.com](mailto:dyfan.r.jones@gmail.com)>

**Repository** CRAN

**Date/Publication** 2021-09-23 11:20:07 UTC

**R topics documented:**

RAthena-package	3
assume_role	4
athena	5
AthenaWriteTables	6
backend_dbplyr	9
dbClearResult	10
dbColumnInfo	11
dbConnect,AthenaDriver-method	12
dbConvertTable	15
dbDataType,AthenaDriver,ANY-method	17
dbDisconnect	18
dbExistsTable	19
dbFetch	20
dbGetInfo	21
dbGetPartition	23
dbGetQuery	24
dbGetStatement	25
dbGetTables	26
dbHasCompleted	27
dbIsValid	28
dbListFields	29
dbListTables	30
dbQuote	31
dbRemoveTable	32
dbShow	33
dbStatistics	34
db_compute	35
db_copy_to	36
db_desc	38
install_boto	39
Query	40
RAthena_options	41
session_token	42
sqlCreateTable	44
sqlData	45
sql_translate_env	46
work_group	47

## 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](mailto:dyfan.r.jones@gmail.com)>

## See Also

Useful links:

- <https://github.com/DyfanJones/RAthena>
- Report bugs at <https://github.com/DyfanJones/RAthena/issues>

---

assume_role	<i>Assume AWS ARN Role</i>
-------------	----------------------------

---

### 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

```
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 <a href="#">AWS Command Line Interface (AWS CLI)</a> will need to be configured. To configure AWS CLI please refer to: <a href="#">Configuring the AWS CLI</a> .
region_name	Default region when creating new connections. Please refer to <a href="#">link</a> 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:assum
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)
```

---

athena

*Athena Driver*

---

**Description**

Driver for an Athena Boto3 connection.

**Usage**

```
athena()
```

**Value**

athena() returns a `s4` class. This class is used active Athena method for [dbConnect](#)

**See Also**[dbConnect](#)**Examples**

```
RAthena::athena()
```

---

AthenaWriteTables      *Convenience functions for reading/writing DBMS tables*

---

## Description

Convenience functions for reading/writing DBMS tables

## Usage

```
## 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 <a href="#">AthenaConnection</a> 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. <ul style="list-style-type: none"> <li>• <b>csv/tsv:</b> <a href="#">LazySimpleSerDe</a></li> <li>• <b>parquet:</b> <a href="#">Parquet SerDe</a></li> <li>• <b>json:</b> <a href="#">JSON SerDe Libraries</a></li> </ul>
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 = "2019-20-13")
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 <a href="#">AthenaConnection</a> object. <b>Note:</b> 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}" 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.

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 ( $\leq 1.6.0$ ) 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. <b>Note:</b> "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 <a href="#">link</a>
...	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

*Athena S3 implementation of dbplyr backend functions*

---

## 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

```
db_save_query.AthenaConnection(
  con,
  sql,
  name,
  file_type = c("NULL", "csv", "tsv", "parquet", "json", "orc"),
  s3_location = NULL,
  partition = NULL,
  compress = TRUE,
  ...
)

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

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

```
sql_escape_date.AthenaConnection(con, x)
```

```
sql_escape_datetime.AthenaConnection(con, x)
```

### Arguments

con	A <a href="#">dbConnect</a> object, as returned by <code>dbConnect()</code>
sql	SQL code to be sent to AWS Athena
name	Table name if left default RAthena will use default from 'dplyr's compute function.
file_type	What file type to store data.frame on s3, RAthena currently supports ["NULL", "csv", "tsv", "parquet", "json", "orc"]. "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.
compress	Compress Athena table, currently can only compress ["parquet", "orc"] <a href="#">AWS Athena CTAS</a>
...	other parameters, currently not implemented
x	R object to be transformed into athena equivalent

### Value

**db\_save\_query** Returns table name

**db\_explain** Raises an error as AWS Athena does not support EXPLAIN queries [Athena Limitations](#)

**db\_query\_fields** Returns sql query column names

---

dbClearResult	<i>Clear Results</i>
---------------	----------------------

---

### 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

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

### Arguments

res	An object inheriting from <a href="#">DBIResult</a> .
...	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)
```

---

dbConnect,AthenaDriver-method

*Connect to Athena using python's sdk boto3*

---

**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

```
## 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,
  ...
)
```

### Arguments

drv	an object that inherits from <a href="#">DBIDriver</a> , or an existing <a href="#">DBIConnection</a> 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 <b>work group</b> 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 <a href="#">link</a> . 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	<a href="#">AWS Key Management Service</a> , please refer to <a href="#">link</a> 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 <a href="#">AWS Command Line Interface</a> (AWS CLI) will need to be configured. To configure AWS CLI please refer to: <a href="#">Configuring the AWS CLI</a> .
role_arn	The Amazon Resource Name (ARN) of the role to assume (such as arn:aws:sts::123456789012:assum
role_session_name	An identifier for the assumed role session. By default 'RAthena' creates a session name <code>sprintf("RAthena-session-%s", as.integer(Sys.time()))</code>
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 <code>s3://path/to/query/bucket/</code>
region_name	Default region when creating new connections. Please refer to <a href="#">link</a> for AWS region codes (region code example: Region = EU (Ireland) <code>region_name = "eu-west-1"</code> )
botocore_session	Use this Botocore session instead of creating a new default one.
bigint	The R type that 64-bit integer types should be mapped to, default is <code>[bit64::integer64]</code> , which allows the full range of 64 bit integers.
binary	The R type that <code>[binary/varbinary]</code> types should be mapped to, default is <code>[raw]</code> . If the mapping fails R will resort to <code>[character]</code> type. To ignore data type conversion set to <code>["character"]</code> .
json	Attempt to converts AWS Athena data types <code>[arrays, json]</code> using <code>jsonlite::parse_json</code> . If the mapping fails R will resort to <code>[character]</code> type. Custom Json parsers can be provide by using a function with data frame parameter. To ignore data type conversion set to <code>["character"]</code> .
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: <a href="https://docs.aws.amazon.com/athena/latest/ug/athena-supported-time-zones.html">https://docs.aws.amazon.com/athena/latest/ug/athena-supported-time-zones.html</a> .
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 and display it in RStudio's Connections Tab. Default set to TRUE.
...	Any other parameter for Boto3 session: <a href="#">Boto3 session documentation</a>

**Value**

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

**See Also**

[dbConnect](#)

**Examples**

```
## 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

*Simple wrapper to convert Athena backend file types*

---

**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**

```
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 <a href="#">AthenaConnection</a> 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 <a href="#">link</a>
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"] ( <a href="#">AWS Athena CTAS</a> )
data	If name should be created with data or not.

### Value

dbConvertTable() returns TRUE but invisible.

### Examples

```

## 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)

```

---

dbDataType,AthenaDriver,ANY-method

*Determine SQL data type of object*

---

## Description

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

## Usage

```

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

## 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 <a href="#">DBIDriver</a> or <a href="#">DBIConnection</a>
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**

```
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 conenction
dbDisconnect(con)

## End(Not run)
```

---

dbDisconnect

*Disconnect (close) an Athena connection*

---

**Description**

This closes the connection to Athena.

**Usage**

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

**Arguments**

```
conn      A DBIConnection object, as returned by dbConnect\(\).
...       Other parameters passed on to methods.
```

**Value**

dbDisconnect() returns TRUE, invisibly.

**See Also**

[dbDisconnect](#)

**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())

# Disconnect conenction
dbDisconnect(con)

## End(Not run)
```

---

dbExistsTable	<i>Does Athena table exist?</i>
---------------	---------------------------------

---

**Description**

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

**Usage**

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

**Arguments**

conn            A [DBIConnection](#) object, as returned by `dbConnect()`.  
 name           A character string specifying a DBMS table name.  
 ...            Other parameters passed on to methods.

**Value**

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

**See Also**

[dbExistsTable](#)

**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/")

# Check if table exists from Athena
dbExistsTable(con, "mtcars")

# Disconnect conenction
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**

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

**Arguments**

res	An object inheriting from <a href="#">DBIResult</a> , created by <a href="#">dbSendQuery()</a> .
n	maximum number of records to retrieve per fetch. Use <code>n = -1</code> or <code>n = Inf</code> 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 <code>n = -1</code> or <code>n = Inf</code> .
...	Other arguments passed on to methods.

**Value**

`dbFetch()` returns a data frame.

**See Also**

[dbFetch](#)

**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())

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**

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

**Arguments**

dbObj	An object inheriting from <a href="#">DBIObject</a> , i.e. <a href="#">DBIDriver</a> , <a href="#">DBIConnection</a> , or a <a href="#">DBIResult</a>
...	Other arguments to methods.

**Value**

a named list

**See Also**

[dbGetInfo](#)

**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())  
  
# 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	<i>Athena table partitions</i>
----------------	--------------------------------

---

### 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 <a href="#">DBIConnection</a> object, as returned by <code>dbConnect()</code> .
name	A character string specifying a DBMS 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	<i>Send query, retrieve results and then clear result set</i>
------------	---

---

### Description

Send query, retrieve results and then clear result set

### Usage

```
## S4 method for signature 'AthenaConnection,character'
dbGetQuery(conn, statement = NULL, statistics = FALSE, ...)
```

### Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
statement	a character string containing SQL.
statistics	If set to TRUE will print out AWS Athena statistics of query.
...	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. It is better use query caching [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 conenction
dbDisconnect(con)

## End(Not run)
```

---

dbGetStatement	<i>Get the statement associated with a result set</i>
----------------	---

---

## Description

Returns the statement that was passed to [dbSendQuery()] or [dbSendStatement()].

## Usage

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

## Arguments

res	An object inheriting from <a href="#">DBIResult</a> .
...	Other arguments passed on to methods.

## Value

dbGetStatement() returns a character.

## See Also

[dbGetStatement](#)

## 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())

rs <- dbSendQuery(con, "SHOW TABLES in default")
```

```
dbGetStatement(rs)

## 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

```
dbGetTables(conn, ...)

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

## Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
...	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.

## Value

dbGetTables() returns a data.frame.

## Examples

```
## Not run:
# Note:
# - Require AWS Account to run below example.
# - Different connection methods can be used please see `noctua::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 conenction
dbDisconnect(con)

## End(Not run)
```

---

dbHasCompleted	<i>Completion status</i>
----------------	--------------------------

---

### Description

This method returns if the query has completed.

### Usage

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

### Arguments

res	An object inheriting from <a href="#">DBIResult</a> .
...	Other arguments passed on to methods.

### Value

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

### 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())  
  
# Check if query has completed  
res <- dbSendQuery(con, "show databases")  
dbHasCompleted(res)  
  
dbClearResult(res)  
  
# Disconnect from Athena  
dbDisconnect(con)  
  
## End(Not run)
```

---

dbIsValid	<i>Is this DBMS object still valid?</i>
-----------	---

---

### 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 <a href="#">DBIObject</a> , i.e. <a href="#">DBIDriver</a> , <a href="#">DBIConnection</a> , or a <a href="#">DBIResult</a>
...	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 <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	a character string with the name of the remote table.
...	Other parameters passed on to methods.

## Value

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

## See Also

[dbListFields](#)

## 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/")

# 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

```

## S4 method for signature 'AthenaConnection'
dbListTables(conn, schema = NULL, ...)

```

### Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
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

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

### See Also

[dbListTables](#)

### 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())

```

```
# Return list of tables in Athena
dbListTables(con)

# Disconnect connection
dbDisconnect(con)

## End(Not run)
```

---

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

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

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

### Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
x	A character vector to quote as string.
...	Other arguments passed on to methods.

### Value

Returns a character object, for more information please check out: [dbQuoteString](#), [dbQuoteIdentifier](#)

### See Also

[dbQuoteString](#), [dbQuoteIdentifier](#)

---

dbRemoveTable	<i>Remove table from Athena</i>
---------------	---------------------------------

---

### Description

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

### Usage

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

### Arguments

conn	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	A character string specifying a DBMS 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 conenction
dbDisconnect(con)

## End(Not run)

```

---

dbShow	<i>Show Athena table's DDL</i>
--------	--------------------------------

---

## 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 <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
name	A character string specifying a DBMS table name.
...	Other parameters passed on to methods.

## Value

dbShow() returns [SQL](#) characters of the Athena table DDL.

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

# disconnect from Athena
dbDisconnect(con)

## End(Not run)

```

---

dbStatistics

*Show AWS Athena Statistics*


---

## Description

Returns AWS Athena Statistics from execute queries [dbSendQuery](#)

## Usage

```

dbStatistics(res, ...)

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

```

## Arguments

```

res           An object inheriting from DBIResult.
...           Other arguments passed on to methods.

```

## Value

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

## Examples

```

## 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)

## End(Not run)
```

---

db\_compute

*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

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

## Arguments

con	A <a href="#">dbConnect</a> object, as returned by <code>dbConnect()</code>
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: <code>[file_type,s3_location,partition]</code> <ul style="list-style-type: none"><li>• <code>file_type</code>: What file type to store data.frame on s3, RAthena currently supports ["NULL","csv", "parquet", "json"]. "NULL" will let Athena set the <code>file_type</code> for you.</li><li>• <code>s3_location</code>: s3 bucket to store Athena table, must be set as a s3 uri for example ("s3://mybucket/data/")</li><li>• <code>partition</code>: Partition Athena table, requires to be a partitioned variable from previous table.</li></ul>

## Value

db\_compute returns table name

## See Also

[backend\\_dbplyr](#)

## Examples

```
## 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_filer %>%
  compute()

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

# Disconnect from Athena
dbDisconnect(con)

## End(Not run)
```

---

db\_copy\_to

*S3 implementation of db\_copy\_to for Athena*

---

## Description

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

## Usage

```
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 <a href="#">dbConnect</a> object, as returned by <code>dbConnect()</code>
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.
values	A <code>data.frame</code> to write to the database.
overwrite	Allow overwriting the destination table. Cannot be TRUE if <code>append</code> is also TRUE.
append	Allow appending to the destination table. Cannot be TRUE if <code>overwrite</code> is also TRUE. Existing Athena DDL file type will be retained and used when uploading data to AWS Athena. If parameter <code>file.type</code> 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: <code>c(var1 = "2019-20-13")</code>
s3_location	s3 bucket to store Athena table, must be set as a s3 uri for example ( <code>"s3://mybucket/data/"</code> )
file_type	What file type to store <code>data.frame</code> on s3, RAthena currently supports ["tsv", "csv", "parquet"]. Default delimited file type is "tsv", in previous versions of RAthena ( $\leq 1.6.0$ ) 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. <b>Note:</b> "parquet" format is supported by the arrow package and it will need to be installed to utilise the "parquet" format.
compress	FALSE   TRUE To determine if to compress <code>file.type</code> . If file type is ["csv", "tsv"] then "gzip" compression is used, for 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 TRUE and <code>file.type</code> is "csv" or "tsv" <code>max.batch</code> will split <code>data.frame</code> into 20 batches. This is to help the performance of AWS Athena when working with files compressed in "gzip" format. <code>max.batch</code> will not split the <code>data.frame</code> when loading file in parquet format. For more information please go to <a href="#">link</a>
...	other parameters currently not supported in RAthena

### Value

`db_copy_to` returns table name

**See Also**[AthenaWriteTables](#)**Examples**

```
## 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())

# 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*

---

**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()`

**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 = "RAthena",
  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 <b>Finding Conda</b> 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 <code>[reticulate::conda_install()]</code> or <code>[reticulate::virtualenv_install()]</code> .

**Value**

Returns NULL after installing Python Boto3.

**Note**

[reticulate::use\_python] or [reticulate::use\_condaenv] might be required before connecting to Athena.

---

Query

*Execute a query on 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**

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

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

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

**Arguments**

<code>conn</code>	A <a href="#">DBIConnection</a> object, as returned by <a href="#">dbConnect()</a> .
<code>statement</code>	a character string containing SQL.
<code>...</code>	Other parameters passed on to methods.

**Value**

Returns `AthenaResult` s4 class.

**See Also**

[dbSendQuery](#), [dbSendStatement](#), [dbExecute](#)



**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
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 = c("data.table", "vroom"),
  bigint = NULL,
  binary = NULL,
  json = NULL,
  cache_size = 0,
  clear_cache = FALSE,
  retry = 5,
  retry_quiet = FALSE
)
```

**Arguments**

**file\_parser** Method to read and write tables to Athena, currently defaults to `data.table`. The `file_parser` also determines the data format returned for example `data.table` will return `data.table` and `vroom` will return `tibble`.

bigint	The R type that 64-bit integer types should be mapped to. Default NULL won't make any changes that dbConnect has set. Inbuilt bigint conversion types ["integer64", "integer", "numeric", "character"].
binary	The R type that [binary/varbinary] types should be mapped to. Default NULL won't make any changes that dbConnect has set. Inbuilt binary conversion types ["raw", "character"].
json	Attempt to convert AWS Athena data types [arrays, json] using <code>jsonlite::parse_json</code> . Default NULL won't make any changes that dbConnect has set. Inbuilt json conversion types ["auto", "character"]. Custom Json parsers can be provided 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.
clear_cache	Clears all previous cached query metadata
retry	Maximum number of requests to attempt.
retry_quiet	If FALSE, will print a message from retry displaying how long until the next request.

**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 <a href="#">AWS Command Line Interface</a> (AWS CLI) will need to be configured. To configure AWS CLI please refer to: <a href="#">Configuring the AWS CLI</a> .
region_name	Default region when creating new connections. Please refer to <a href="#">link</a> 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**

```
## 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	<i>Creates query to create a simple Athena table</i>
----------------	--

---

## Description

Creates an interface to compose CREATE EXTERNAL TABLE.

## Usage

```
## 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	Name of the table. Escaped with <code>dbQuoteIdentifier()</code> .
fields	Either a character vector or a data frame. A named character vector: Names are column names, values are types. Names are escaped with <code>dbQuoteIdentifier()</code> . Field types are unescaped. A data frame: field types are generated using <code>dbDataType()</code> .
field.types	Additional field types used to override derived types.
partition	Partition Athena table (needs to be a named list or vector) for example: <code>c(var1 = "2019-20-13")</code>
s3.location	s3 bucket to store Athena table, must be set as a s3 uri for example (" <code>s3://mybucket/data/</code> "). By default s3.location is set s3 staging directory from <code>AthenaConnection</code> object.
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 ( <code>=&lt; 1.6.0</code> ) 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. <b>Note:</b> "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 <code>jsonlite</code> 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.

... Other arguments used by individual methods.

### 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 `RAthena::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**

```
## 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"]. <b>Note:</b> 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](#)

---

sql\_translate\_env      *AWS Athena backend dbplyr*

---

**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_translate_env.AthenaConnection(con)

sql_escape_string.AthenaConnection(con, x)
```

**Arguments**

con	An <a href="#">AthenaConnection</a> 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.

work\_group

*Athena Work Groups***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 work-group 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](#)

**Usage**

```
create_work_group(
  conn,
  work_group = NULL,
  enforce_work_group_config = FALSE,
  publish_cloud_watch_metrics = FALSE,
  bytes_scanned_cut_off = 10000000L,
  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 = 10000000L,
  requester_pays = FALSE,
  description = NULL,
  state = c("ENABLED", "DISABLED")
)

```

### Arguments

conn	A <a href="#">dbConnect</a> object, as returned by <code>dbConnect()</code>
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 <a href="#">Workgroup Settings Override Client-Side Settings</a> .
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 <a href="#">Requester Pays Buckets</a> 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 <code>tag_options()</code> 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 <a href="#">Workgroup Settings Override Client-Side Settings</a> .
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

```
## 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](#), [16](#), [44](#), [47](#)
- AthenaWriteTables, [6](#), [38](#)
- backend\_dbplyr, [9](#), [35](#)
- create\_work\_group (work\_group), [47](#)
- db\_compute, [35](#)
- db\_copy\_to, [36](#)
- db\_desc, [38](#)
- db\_explain.AthenaConnection  
(backend\_dbplyr), [9](#)
- db\_query\_fields.AthenaConnection  
(backend\_dbplyr), [9](#)
- db\_save\_query.AthenaConnection  
(backend\_dbplyr), [9](#)
- dbClearResult, [10](#)
- dbClearResult, AthenaResult-method  
(dbClearResult), [10](#)
- dbColumnInfo, [11](#)
- dbColumnInfo, AthenaResult-method  
(dbColumnInfo), [11](#)
- dbConnect, [5](#), [10](#), [15](#), [35](#), [37](#), [39](#), [47](#), [48](#)
- dbConnect  
(dbConnect, AthenaDriver-method),  
[12](#)
- dbConnect(), [19](#), [20](#), [23](#), [24](#), [26](#), [29–33](#), [40](#)
- dbConnect, AthenaDriver-method, [12](#)
- dbConvertTable, [15](#)
- dbConvertTable, AthenaConnection-method  
(dbConvertTable), [15](#)
- dbDataType, [18](#)
- dbDataType  
(dbDataType, AthenaDriver, ANY-method),  
[17](#)
- dbDataType(), [44](#)
- dbDataType, AthenaConnection, ANY-method  
(dbDataType, AthenaDriver, ANY-method),  
[17](#)
- dbDataType, AthenaConnection, data.frame-method  
(dbDataType, AthenaDriver, ANY-method),  
[17](#)
- dbDataType, AthenaDriver, ANY-method, [17](#)
- dbDataType, AthenaDriver, list-method  
(dbDataType, AthenaDriver, ANY-method),  
[17](#)
- dbDisconnect, [18](#), [19](#)
- dbDisconnect, AthenaConnection-method  
(dbDisconnect), [18](#)
- dbExecute, [40](#)
- dbExecute, AthenaConnection, character-method  
(Query), [40](#)
- dbExistsTable, [19](#), [20](#)
- dbExistsTable, AthenaConnection, character-method  
(dbExistsTable), [19](#)
- dbFetch, [20](#), [21](#)
- dbFetch, AthenaResult-method (dbFetch),  
[20](#)
- dbGetInfo, [21](#), [22](#)
- dbGetInfo, AthenaConnection-method  
(dbGetInfo), [21](#)
- dbGetInfo, AthenaResult-method  
(dbGetInfo), [21](#)
- dbGetPartition, [23](#)
- dbGetPartition, AthenaConnection-method  
(dbGetPartition), [23](#)
- dbGetQuery, [24](#), [24](#)
- dbGetQuery, AthenaConnection, character-method  
(dbGetQuery), [24](#)
- dbGetStatement, [25](#), [25](#)
- dbGetStatement, AthenaResult-method  
(dbGetStatement), [25](#)
- dbGetTables, [26](#)
- dbGetTables, AthenaConnection-method  
(dbGetTables), [26](#)
- dbHasCompleted, [12](#), [27](#), [27](#), [40](#)
- dbHasCompleted, AthenaResult-method  
(dbHasCompleted), [27](#)

- DBIConnection, [13](#), [17](#), [19](#), [20](#), [22–24](#), [26](#), [28–33](#), [40](#)
- DBIDriver, [13](#), [17](#), [22](#), [28](#)
- DBIObject, [22](#), [28](#)
- DBIResult, [10](#), [11](#), [21](#), [22](#), [25](#), [27](#), [28](#), [34](#)
- dbIsValid, [11](#), [28](#), [28](#)
- dbIsValid, AthenaConnection-method  
(dbIsValid), [28](#)
- dbIsValid, AthenaResult-method  
(dbIsValid), [28](#)
- dbListFields, [29](#), [29](#)
- dbListFields, AthenaConnection, character-method  
(dbListFields), [29](#)
- dbListTables, [30](#), [30](#)
- dbListTables, AthenaConnection-method  
(dbListTables), [30](#)
- dbQuote, [31](#)
- dbQuoteIdentifier, [31](#)
- dbQuoteIdentifier(), [44](#)
- dbQuoteIdentifier, AthenaConnection, SQL-method  
(dbQuote), [31](#)
- dbQuoteString, [31](#)
- dbQuoteString, AthenaConnection, character-method  
(dbQuote), [31](#)
- dbRemoveTable, [32](#), [32](#)
- dbRemoveTable, AthenaConnection, character-method  
(dbRemoveTable), [32](#)
- dbSendQuery, [34](#), [40](#)
- dbSendQuery(), [21](#)
- dbSendQuery, AthenaConnection, character-method  
(Query), [40](#)
- dbSendStatement, [40](#)
- dbSendStatement, AthenaConnection, character-method  
(Query), [40](#)
- dbShow, [33](#)
- dbShow, AthenaConnection-method  
(dbShow), [33](#)
- dbStatistics, [34](#)
- dbStatistics, AthenaResult-method  
(dbStatistics), [34](#)
- 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), [47](#)
- get\_session\_token (session\_token), [42](#)
- get\_work\_group (work\_group), [47](#)
- install\_boto, [39](#)
- list\_work\_groups (work\_group), [47](#)
- Query, [40](#)
- RAthena (RAthena-package), [3](#)
- RAthena-package, [3](#)
- RAthena\_options, [11](#), [24](#), [41](#)
- session\_token, [42](#)
- SQL, [33](#), [45](#)
- sql\_escape\_date.AthenaConnection  
(backend\_dbplyr), [9](#)
- sql\_escape\_datetime.AthenaConnection  
(backend\_dbplyr), [9](#)
- sql\_escape\_string.AthenaConnection  
(sql\_translate\_env), [46](#)
- sql\_translate\_env, [46](#)
- sqlCreateTable, [44](#), [45](#)
- sqlCreateTable, AthenaConnection-method  
(sqlCreateTable), [44](#)
- sqlData, [45](#), [46](#)
- sqlData, AthenaConnection-method  
(sqlData), [45](#)
- tag\_options (work\_group), [47](#)
- update\_work\_group (work\_group), [47](#)
- work\_group, [47](#)