Package 'bigQueryR'

October 12, 2022

Title Interface with Google BigQuery with Shiny Compatibility

Version 0.5.0

Description Interface with 'Google BigQuery',
see <https://cloud.google.com/bigquery/> for more information.
This package uses 'googleAuthR' so is compatible with similar packages,
including 'Google Cloud Storage' (<https://cloud.google.com/storage/>) for result extracts.

URL http://code.markedmondson.me/bigQueryR/

BugReports https://github.com/cloudyr/bigQueryR/issues

License MIT + file LICENSE

LazyData TRUE

Depends R (>= 3.3)

Imports googleAuthR (>= 1.1.1), googleCloudStorageR (>= 0.2.0),
jsonlite (>= 1.0), httr (>= 1.2.1), assertthat

Suggests shiny (>= 0.12.1), knitr, rmarkdown, testthat, data.table,
purrr

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Mark Edmondson [aut, cre],
Hadley Wickham [ctb]

Maintainer Mark Edmondson <r@sunholo.com>

Repository CRAN

Date/Publication 2019-10-09 12:40:03 UTC

R topics documented:

bigQueryR 2
bqr_auth 3
bqr_copy_dataset 4

bqr_copy_table . . . 5
bqr_create_table . . . 6
bqr_delete_table . . . 7
bqr_download_extract . . . 8
bqr_download_query . . . 8
bqr_extract_data . . . 10
bqr_get_global_dataset . . . 11
bqr_get_global_project . . . 12
bqr_get_job . . . 13
bqr_global_dataset . . . 14
bqr_global_project . . . 15
bqr_grant_extract_access . . . 16
bqr_list_datasets . . . 17
bqr_list_jobs . . . 18
bqr_list_projects . . . 19
bqr_list_tables . . . 20
bqr_partition . . . 20
bqr_patch_table . . . 22
bqr_query . . . 22
bqr_query_asynch . . . 23
bqr_table_data . . . 25
bqr_table_meta . . . 26
bqr_upload_data . . . 27
bqr_wait_for_job . . . 29
schema_fields . . . 30
Table . . . 30

Index **32**

`bigQueryR` *bigQueryR*

Description

Provides an interface with Google BigQuery

See Also

https://cloud.google.com/bigquery/docs/reference/v2/?hl=en

bqr_auth *Authenticate this session*

Description

Autheticate manually via email or service JSON file

Usage

```
bqr_auth(json_file = NULL, token = NULL,
  email = Sys.getenv("GARGLE_EMAIL"))
```

Arguments

json_file	Authentication json file you have downloaded from your Google Project
token	A preexisting token to authenticate with
email	A Google email to authenticate with
	If you have set the environment variable BQ_AUTH_FILE to a valid file location, the function will look there for authentication details. Otherwise it will trigger an authentication flow via Google login screen in your browser based on the email you provide.
	If BQ_AUTH_FILE is specified, then authentication will be called upon loading the package via library(bigQueryR), meaning that calling this function yourself at the start of the session won't be necessary.
	BQ_AUTH_FILE is a GCP service account JSON ending with file extension .json

Value

Invisibly, the token that has been saved to the session

Examples

```
## Not run:

# to use default package credentials (for testing)
library(bigQueryR)
bqr_auth("location_of_json_file.json")

# or via email
bqr_auth(email="me@work.com")

# to use your own Google Cloud Project credentials
# go to GCP console and download client credentials JSON
# ideally set this in .Renviron file, not here but just for demonstration
Sys.setenv("GAR_CLIENT_JSON" = "location/of/file.json")
library(bigQueryR)
# should now be able to log in via your own GCP project
```

```
bqr_auth()

# reauthentication
# Once you have authenticated, set email to skip the interactive message
bqr_auth(email = "my@email.com")

# or leave unset to bring up menu on which email to auth with
bqr_auth()
# The bigQueryR package is requesting access to your Google account.
# Select a pre-authorised account or enter '0' to obtain a new token.
# Press Esc/Ctrl + C to abort.
#1: my@email.com
#2: work@mybusiness.com
# you can set authentication for many emails, then switch between them e.g.
bqr_auth(email = "my@email.com")
bqr_list_projects() # lists what GCP projects you have access to
bqr_auth(email = "work@mybusiness.com")
bqr_list_projects() # lists second set of projects



## End(Not run)
```

`bqr_copy_dataset` *Copy datasets*

Description

Uses bqr_copy_table to copy all the tables in a dataset.

Usage

```
bqr_copy_dataset(source_datasetid, destination_datasetid,
  source_projectid = bqr_get_global_project(),
  destination_projectid = bqr_get_global_project(),
  createDisposition = c("CREATE_IF_NEEDED", "CREATE_NEVER"),
  writeDisposition = c("WRITE_TRUNCATE", "WRITE_APPEND", "WRITE_EMPTY"),
  destination_prefix = NULL)
```

Arguments

`source_datasetid`
: source datasetId

`destination_datasetid`
: destination datasetId

`source_projectid`
: source table's projectId

destination_projectid
: destination table's projectId

createDisposition
: Create table's behaviour

writeDisposition
: Write to an existing table's behaviour

destination_prefix
: A prefix appended to the destination tableIds

Details

You can not copy across dataset regions (e.g. EU to US), or copy BigQuery Views.

Value

A named list of jobs of the source datasets, with details of job started.

Examples

```
## Not run:

  bqr_copy_dataset("source_dataset", "destination_dataset")


## End(Not run)
```

---

bqr_copy_table *Copy BigQuery table*

Description

Copy a source table to another destination

Usage

```
bqr_copy_table(source_tableid, destination_tableid,
  source_projectid = bqr_get_global_project(),
  source_datasetid = bqr_get_global_dataset(),
  destination_projectid = bqr_get_global_project(),
  destination_datasetid = bqr_get_global_dataset(),
  createDisposition = c("CREATE_IF_NEEDED", "CREATE_NEVER"),
  writeDisposition = c("WRITE_TRUNCATE", "WRITE_APPEND", "WRITE_EMPTY"))
```

Arguments

source_tableid	source table's tableId
destination_tableid	destination table's tableId
source_projectid	source table's projectId
source_datasetid	source table's datasetId
destination_projectid	destination table's projectId
destination_datasetid	destination table's datasetId
createDisposition	Create table's behaviour
writeDisposition	Write to an existing table's behaviour

Value

A job object

See Also

Other Table meta functions: Table, bqr_create_table, bqr_delete_table, bqr_list_tables, bqr_patch_table, bqr_table_data, bqr_table_meta

---

bqr_create_table *Create a Table*

Description

Create a Table

Usage

```
bqr_create_table(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), tableId, template_data = NULL,
  timePartitioning = FALSE, expirationMs = 0L)
```

Arguments

projectId	The BigQuery project ID.
datasetId	A datasetId within projectId.
tableId	Name of table you want.
template_data	A dataframe with the correct types of data. If NULL an empty table is made.

`timePartitioning`
: Whether to create a partioned table

`expirationMs`
: If a partioned table, whether to have an expiration time on the data. The default `0` is no expiration.

Details

Creates a BigQuery table.

If setting `timePartioning` to `TRUE` then the table will be a partioned table

If you want more advanced features for the table, create it then call bqr_patch_table with advanced configuration configured from Table

Value

TRUE if created, FALSE if not.

See Also

Other Table meta functions: `Table`, `bqr_copy_table`, `bqr_delete_table`, `bqr_list_tables`, `bqr_patch_table`, `bqr_table_data`, `bqr_table_meta`

`bqr_delete_table` *Delete a Table*

Description

Delete a Table

Usage

```
bqr_delete_table(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), tableId)
```

Arguments

`projectId`
: The BigQuery project ID.

`datasetId`
: A datasetId within projectId.

`tableId`
: Name of table you want to delete.

Details

Deletes a BigQuery table

Value

TRUE if deleted, FALSE if not.

See Also

Other Table meta functions: Table, bqr_copy_table, bqr_create_table, bqr_list_tables, bqr_patch_table, bqr_table_data, bqr_table_meta

---

`bqr_download_extract` *Download extract data*

Description

After extracting data via bqr_extract_data download the extract from the Google Storage bucket.

If more than 1GB, will save multiple .csv files with prefix "N_" to filename.

Usage

```
bqr_download_extract(extractJob, filename = NULL)
```

Arguments

`extractJob`	An extract job from bqr_extract_data
`filename`	Where to save the csv file. If NULL then uses objectname.

Value

TRUE if successfully downloaded

See Also

Other BigQuery asynch query functions: bqr_extract_data, bqr_get_job, bqr_grant_extract_access, bqr_query_asynch, bqr_wait_for_job

---

`bqr_download_query` *Download data from BigQuery to local folder*

Description

Requires you to make a bucket at https://console.cloud.google.com/storage/browser

Usage

```
bqr_download_query(query = NULL, target_folder = "data",
  result_file_name = NULL, refetch = FALSE, useLegacySql = FALSE,
  clean_intermediate_results = TRUE,
  global_project_name = bqr_get_global_project(),
  global_dataset_name = bqr_get_global_dataset(),
  global_bucket_name = googleCloudStorageR::gcs_get_global_bucket())
```

Arguments

query	The query you want to run.
target_folder	Target folder on your local computer.
result_file_name	Name of your downloaded file.
refetch	Boolean, whether you would like to refetch previously downloaded data.
useLegacySql	Boolean, whether to use Legacy SQL. Default is FALSE.
clean_intermediate_results	Boolean, whether to keep intermediate files on BigQuery and Google Cloud Storage.
global_project_name	BigQuery project name (where you would like to save your file during download).
global_dataset_name	BigQuery dataset name (where you would like to save your file during download).
global_bucket_name	Google Cloud Storage bucket name (where you would like to save your file during download).

Value

a data.table.

Examples

```
## Not run:
library(bigQueryR)

## Auth with a project that has at least BigQuery and Google Cloud Storage scope
bqr_auth()

# Create a bucket at Google Cloud Storage at
# https://console.cloud.google.com/storage/browser

bqr_download_query(query = "select * from `your_project.your_dataset.your_table`")


## End(Not run)
```

bqr_extract_data *Extract data asynchronously*

Description

Use this instead of bqr_query for big datasets. Requires you to make a bucket at https://console.cloud.google.com/storage/brow

Usage

```
bqr_extract_data(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), tableId, cloudStorageBucket,
  filename = paste0("big-query-extract-", gsub(" |:|-", "", Sys.time()),
  "-*.csv"), compression = c("NONE", "GZIP"),
  destinationFormat = c("CSV", "NEWLINE_DELIMITED_JSON", "AVRO"),
  fieldDelimiter = ",", printHeader = TRUE)
```

Arguments

projectId	The BigQuery project ID.
datasetId	A datasetId within projectId.
tableId	ID of table you wish to extract.
cloudStorageBucket	URI of the bucket to extract into.
filename	Include a wildcard (*) if extract expected to be > 1GB.
compression	Compression of file.
destinationFormat	Format of file.
fieldDelimiter	fieldDelimiter of file.
printHeader	Whether to include header row.

Value

A Job object to be queried via bqr_get_job

See Also

https://cloud.google.com/bigquery/exporting-data-from-bigquery

Other BigQuery asynch query functions: bqr_download_extract, bqr_get_job, bqr_grant_extract_access, bqr_query_asynch, bqr_wait_for_job

Examples

```
## Not run:
library(bigQueryR)

## Auth with a project that has at least BigQuery and Google Cloud Storage scope
bqr_auth()

## make a big query
job <- bqr_query_asynch("your_project",
                        "your_dataset",
                        "SELECT * FROM blah LIMIT 9999999",
                        destinationTableId = "bigResultTable")

## poll the job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job)

##once done, the query results are in "bigResultTable"
## extract that table to GoogleCloudStorage:
# Create a bucket at Google Cloud Storage at
# https://console.cloud.google.com/storage/browser

job_extract <- bqr_extract_data("your_project",
                                "your_dataset",
                                "bigResultTable",
                                "your_cloud_storage_bucket_name")

## poll the extract job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job_extract$jobReference$jobId)

You should also see the extract in the Google Cloud Storage bucket
googleCloudStorageR::gcs_list_objects("your_cloud_storage_bucket_name")

## to download via a URL and not logging in via Google Cloud Storage interface:
## Use an email that is Google account enabled
## Requires scopes:
##  https://www.googleapis.com/auth/devstorage.full_control
##  https://www.googleapis.com/auth/cloud-platform

download_url <- bqr_grant_extract_access(job_extract, "your@email.com")

## download_url may be multiple if the data is > 1GB


## End(Not run)
```

bqr_get_global_dataset

Get global dataset name

Description

dataset name set this session to use by default

Usage

```
bqr_get_global_dataset()

bq_get_global_dataset()
```

Details

Set the dataset name via bq_global_dataset

Value

dataset name

See Also

Other dataset functions: bqr_global_dataset

bqr_get_global_project

Get global project name

Description

project name set this session to use by default

Usage

```
bqr_get_global_project()

bq_get_global_project()
```

Details

Set the project name via bq_global_project

Value

project name

See Also

Other project functions: bqr_global_project

---

bqr_get_job *Poll a jobId*

Description

Poll a jobId

Usage

```
bqr_get_job(jobId = .Last.value, projectId = bqr_get_global_project())
```

Arguments

jobId	jobId to poll, or a job Object
projectId	projectId of job

Value

A Jobs resource

See Also

Other BigQuery asynch query functions: bqr_download_extract, bqr_extract_data, bqr_grant_extract_access, bqr_query_asynch, bqr_wait_for_job

Examples

```
## Not run:
library(bigQueryR)

## Auth with a project that has at least BigQuery and Google Cloud Storage scope
bqr_auth()

## make a big query
job <- bqr_query_asynch("your_project",
                        "your_dataset",
                        "SELECT * FROM blah LIMIT 9999999",
                        destinationTableId = "bigResultTable")

## poll the job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job$jobReference$jobId)

##once done, the query results are in "bigResultTable"
```

```
## extract that table to GoogleCloudStorage:
# Create a bucket at Google Cloud Storage at
# https://console.cloud.google.com/storage/browser

job_extract <- bqr_extract_data("your_project",
                                "your_dataset",
                                "bigResultTable",
                                "your_cloud_storage_bucket_name")

## poll the extract job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job_extract$jobReference$jobId)

## to download via a URL and not logging in via Google Cloud Storage interface:
## Use an email that is Google account enabled
## Requires scopes:
##  https://www.googleapis.com/auth/devstorage.full_control
##  https://www.googleapis.com/auth/cloud-platform
## set via options("bigQueryR.scopes") and reauthenticate if needed

download_url <- bqr_grant_extract_access(job_extract, "your@email.com")

## download_url may be multiple if the data is > 1GB


## End(Not run)
```

`bqr_global_dataset` *Set global dataset name*

Description

Set a dataset name used for this R session

Usage

```
bqr_global_dataset(dataset)

bq_global_dataset(dataset)
```

Arguments

`dataset`	dataset name you want this session to use by default, or a dataset object

Details

This sets a dataset to a global environment value so you don't need to supply the dataset argument to other API calls.

Value

The dataset name (invisibly)

See Also

Other dataset functions: bqr_get_global_dataset

---

bqr_global_project *Set global project name*

Description

Set a project name used for this R session

Usage

```
bqr_global_project(project)

bq_global_project(project)
```

Arguments

project	project name you want this session to use by default, or a project object

Details

This sets a project to a global environment value so you don't need to supply the project argument to other API calls.

Value

The project name (invisibly)

See Also

Other project functions: bqr_get_global_project

`bqr_grant_extract_access` *Grant access to an extract on Google Cloud Storage*

Description

To access the data created in bqr_extract_data. Requires the Google account email of the user.

Usage

```
bqr_grant_extract_access(extractJob, email)
```

Arguments

`extractJob`	An extract job from bqr_extract_data
`email`	email of the user to have access

Details

Uses cookie based auth.

Value

URL(s) to download the extract that is accessible by email

See Also

Other BigQuery asynch query functions: `bqr_download_extract`, `bqr_extract_data`, `bqr_get_job`, `bqr_query_asynch`, `bqr_wait_for_job`

Examples

```
## Not run:
library(bigQueryR)

## Auth with a project that has at least BigQuery and Google Cloud Storage scope
bqr_auth()

## make a big query
job <- bqr_query_asynch("your_project",
                        "your_dataset",
                        "SELECT * FROM blah LIMIT 9999999",
                        destinationTableId = "bigResultTable")

## poll the job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job$jobReference$jobId)
```

```
##once done, the query results are in "bigResultTable"
## extract that table to GoogleCloudStorage:
# Create a bucket at Google Cloud Storage at
# https://console.cloud.google.com/storage/browser

job_extract <- bqr_extract_data("your_project",
                                "your_dataset",
                                "bigResultTable",
                                "your_cloud_storage_bucket_name")

## poll the extract job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job_extract$jobReference$jobId)

## to download via a URL and not logging in via Google Cloud Storage interface:
## Use an email that is Google account enabled
## Requires scopes:
##  https://www.googleapis.com/auth/devstorage.full_control
##  https://www.googleapis.com/auth/cloud-platform
## set via options("bigQueryR.scopes") and reauthenticate if needed

download_url <- bqr_grant_extract_access(job_extract, "your@email.com")

## download_url may be multiple if the data is > 1GB


## End(Not run)
```

`bqr_list_datasets` *List BigQuery datasets*

Description

Each projectId can have multiple datasets.

Usage

```
bqr_list_datasets(projectId = bqr_get_global_project())
```

Arguments

`projectId`	The BigQuery project ID

See Also

Other bigQuery meta functions: `bqr_list_projects`

Examples

```
## Not run:
  library(bigQueryR)

  ## this will open your browser
  ## Authenticate with an email that has access to the BigQuery project you need
  bqr_auth()

  ## verify under a new user
  bqr_auth(new_user=TRUE)

  ## get projects
  projects <- bqr_list_projects()

  my_project <- projects[1]

  ## for first project, get datasets
  datasets <- bqr_list_datasets[my_project]


## End(Not run)
```

`bqr_list_jobs` *List BigQuery jobs*

Description

List the BigQuery jobs for the projectId

Usage

```
bqr_list_jobs(projectId = bqr_get_global_project(), allUsers = FALSE,
  projection = c("full", "minimal"), stateFilter = c("done", "pending",
  "running"))
```

Arguments

`projectId`	projectId of job
`allUsers`	Whether to display jobs owned by all users in the project.
`projection`	"full" - all job data, "minimal" excludes job configuration.
`stateFilter`	Filter for job status.

Details

Lists all jobs that you started in the specified project. Job information is available for a six month period after creation. The job list is sorted in reverse chronological order, by job creation time. Requires the Can View project role, or the Is Owner project role if you set the allUsers property.

Value

A list of jobs resources

`bqr_list_projects` *List Google Dev Console projects you have access to*

Description

Example: bqr_list_projects()

Usage

```
bqr_list_projects()
```

Value

A dataframe of the projects you have access to under the authentication

See Also

Other bigQuery meta functions: `bqr_list_datasets`

Examples

```
## Not run:
  library(bigQueryR)

  ## this will open your browser
  ## Authenticate with an email that has access to the BigQuery project you need
  bqr_auth()

  ## verify under a new user
  bqr_auth(new_user=TRUE)

  ## get projects
  projects <- bqr_list_projects()


## End(Not run)
```

bqr_list_tables *List BigQuery tables in a dataset*

Description

List BigQuery tables in a dataset

Usage

```
bqr_list_tables(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), maxResults = -1)
```

Arguments

projectId	The BigQuery project ID
datasetId	A datasetId within projectId
maxResults	Number of results to return, default -1 returns all results

Value

dataframe of tables in dataset

See Also

Other Table meta functions: Table, bqr_copy_table, bqr_create_table, bqr_delete_table, bqr_patch_table, bqr_table_data, bqr_table_meta

Examples

```
## Not run:
 bqr_list_tables("publicdata", "samples")

## End(Not run)
```

bqr_partition *Convert date-sharded tables to a single partitioned table*

Description

Moves the old style date-sharded tables such as [TABLE_NAME]_YYYYMMDD to the new date partitioned format.

Usage

```
bqr_partition(sharded, partition, projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset())
```

Arguments

sharded	The prefix of date-sharded tables to merge into one partitioned table
partition	Name of partitioned table. Will create if not present already
projectId	The project ID
datasetId	The dataset ID

Details

Performs lots of copy table operations via bqr_copy_table

Before partitioned tables became available, BigQuery users would often divide large datasets into separate tables organized by time period; usually daily tables, where each table represented data loaded on that particular date.

Dividing a dataset into daily tables helped to reduce the amount of data scanned when querying a specific date range. For example, if you have a a year's worth of data in a single table, a query that involves the last seven days of data still requires a full scan of the entire table to determine which data to return. However, if your table is divided into daily tables, you can restrict the query to the seven most recent daily tables.

Daily tables, however, have several disadvantages. You must manually, or programmatically, create the daily tables. SQL queries are often more complex because your data can be spread across hundreds of tables. Performance degrades as the number of referenced tables increases. There is also a limit of 1,000 tables that can be referenced in a single query. Partitioned tables have none of these disadvantages.

Value

A list of copy jobs for the sharded tables that will be copied to one partitioned table

See Also

Partitioned Tables Help

Examples

```
## Not run:

 bqr_partition("ga_sessions_", "ga_partition")


## End(Not run)
```

bqr_patch_table *Update a Table*

Description

This uses PATCH semantics to alter an existing table. You need to create the Table object first to pass in using Table which will be transformed to JSON

Usage

```
bqr_patch_table(Table)
```

Arguments

Table	A Table object as created by Table

See Also

Definition of tables

Other Table meta functions: Table, bqr_copy_table, bqr_create_table, bqr_delete_table, bqr_list_tables, bqr_table_data, bqr_table_meta

bqr_query *Query a BigQuery Table*

Description

MaxResults is how many results to return per page of results, which can be less than the total results you have set in your query using LIMIT. Google recommends for bigger datasets to set maxResults = 1000, but this will use more API calls.

Usage

```
bqr_query(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), query, maxResults = 1000,
  useLegacySql = TRUE, useQueryCache = TRUE)
```

Arguments

projectId	The BigQuery project ID
datasetId	A datasetId within projectId
query	BigQuery SQL. You can also supply a file location of your query ending with .sql
maxResults	Max number per page of results. Set total rows with LIMIT in your query.
useLegacySql	Whether the query you pass is legacy SQL or not. Default TRUE
useQueryCache	Whether to use the query cache. Default TRUE, set to FALSE for realtime queries.

Value

a data.frame. If there is an SQL error, a data.frame with additional class "bigQueryR_query_error" and the problem in the data.frame$message

See Also

BigQuery SQL reference

Examples

```
## Not run:

bqr_query("big-query-r","samples",
          "SELECT COUNT(repository.url) FROM [publicdata:samples.github_nested]")


## End(Not run)
```

---

bqr_query_asynch *BigQuery query asynchronously*

Description

Use for big results > 10000 that write to their own destinationTableId.

Usage

```
bqr_query_asynch(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), query, destinationTableId,
  useLegacySql = TRUE, writeDisposition = c("WRITE_EMPTY",
  "WRITE_TRUNCATE", "WRITE_APPEND"))
```

Arguments

projectId	projectId to be billed.
datasetId	datasetId of where query will execute.
query	The BigQuery query as a string.
destinationTableId	Id of table the results will be written to.
useLegacySql	Whether the query you pass is legacy SQL or not. Default TRUE
writeDisposition	Behaviour if destination table exists. See Details.

Details

For bigger queries, asynchronous queries save the results to another BigQuery table. You can check the progress of the job via bqr_get_job

You may now want to download this data. For large datasets, this is best done via extracting the BigQuery result to Google Cloud Storage, then downloading the data from there.

You can read how to create a bucket at Google Cloud Storage at https://cloud.google.com/storage/docs/cloud-console

writeDisposition - behaviour if destinationTable already exists:

- WRITE_TRUNCATE: BigQuery overwrites the table data.
- WRITE_APPEND: BigQuery appends the data to the table
- WRITE_EMPTY: If contains data, a 'duplicate' error is returned

Value

A Job object to be queried via bqr_get_job

See Also

Other BigQuery asynch query functions: bqr_download_extract, bqr_extract_data, bqr_get_job, bqr_grant_extract_access, bqr_wait_for_job

Examples

```
## Not run:
library(bigQueryR)

## Auth with a project that has at least BigQuery and Google Cloud Storage scope
bqr_auth()

## make a big query
job <- bqr_query_asynch("your_project",
                        "your_dataset",
                        "SELECT * FROM blah LIMIT 9999999",
                        destinationTableId = "bigResultTable")

## poll the job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job$jobReference$jobId)

##once done, the query results are in "bigResultTable"
## extract that table to GoogleCloudStorage:
# Create a bucket at Google Cloud Storage at
# https://console.cloud.google.com/storage/browser

job_extract <- bqr_extract_data("your_project",
                                "your_dataset",
                                "bigResultTable",
                                "your_cloud_storage_bucket_name")
```

```
## poll the extract job to check its status
## its done when job$status$state == "DONE"
bqr_get_job("your_project", job_extract$jobReference$jobId)

## to download via a URL and not logging in via Google Cloud Storage interface:
## Use an email that is Google account enabled
## Requires scopes:
##  https://www.googleapis.com/auth/devstorage.full_control
##  https://www.googleapis.com/auth/cloud-platform
## set via options("bigQueryR.scopes") and reauthenticate if needed

download_url <- bqr_grant_extract_access(job_extract, "your@email.com")

## download_url may be multiple if the data is > 1GB


## End(Not run)
```

`bqr_table_data` *Get BigQuery Table's data list*

Description

Get BigQuery Table's data list

Usage

```
bqr_table_data(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), tableId, maxResults = 1000)
```

Arguments

`projectId`	The BigQuery project ID
`datasetId`	A datasetId within projectId
`tableId`	The tableId within the datasetId
`maxResults`	Number of results to return

Value

data.frame of table data

This won't work with nested datasets, for that use bqr_query as that flattens results.

See Also

Other Table meta functions: `Table`, `bqr_copy_table`, `bqr_create_table`, `bqr_delete_table`, `bqr_list_tables`, `bqr_patch_table`, `bqr_table_meta`

`bqr_table_meta` *Get BigQuery Table meta data*

Description

Get BigQuery Table meta data

Usage

```
bqr_table_meta(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), tableId)
```

Arguments

`projectId`	The BigQuery project ID
`datasetId`	A datasetId within projectId
`tableId`	The tableId within the datasetId

Value

list of table metadata

See Also

Other Table meta functions: `Table`, `bqr_copy_table`, `bqr_create_table`, `bqr_delete_table`, `bqr_list_tables`, `bqr_patch_table`, `bqr_table_data`

Examples

```
## Not run:
  bqr_table_meta("publicdata", "samples", "github_nested")

## End(Not run)
```

bqr_upload_data *Upload data to BigQuery*

Description

Upload data to BigQuery

Usage

```
bqr_upload_data(projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), tableId, upload_data,
  create = c("CREATE_IF_NEEDED", "CREATE_NEVER"),
  writeDisposition = c("WRITE_TRUNCATE", "WRITE_APPEND", "WRITE_EMPTY"),
  schema = NULL, sourceFormat = c("CSV", "DATASTORE_BACKUP",
  "NEWLINE_DELIMITED_JSON", "AVRO"), wait = TRUE, autodetect = FALSE,
  nullMarker = NULL, maxBadRecords = NULL, allowJaggedRows = FALSE,
  allowQuotedNewlines = FALSE, fieldDelimiter = NULL)
```

Arguments

projectId	The BigQuery project ID.
datasetId	A datasetId within projectId.
tableId	ID of table where data will end up.
upload_data	The data to upload, a data.frame object or a Google Cloud Storage URI
create	Whether to create a new table if necessary, or error if it already exists.
writeDisposition	How to add the data to a table.
schema	If upload_data is a Google Cloud Storage URI, supply the data schema. For CSV a helper function is available by using schema_fields on a data sample
sourceFormat	If upload_data is a Google Cloud Storage URI, supply the data format. Default is CSV
wait	If uploading a data.frame, whether to wait for it to upload before returning
autodetect	Experimental feature that auto-detects schema for CSV and JSON files
nullMarker	Specifies a string that represents a null value in a CSV file. For example, if you specify \N, BigQuery interprets \N as a null value when loading a CSV file. The default value is the empty string.
maxBadRecords	The maximum number of bad records that BigQuery can ignore when running the job
allowJaggedRows	Whether to allow rows with variable length columns
allowQuotedNewlines	Whether to allow datasets with quoted new lines
fieldDelimiter	The separator for fields in a CSV file. Default is comma - ,

Details

A temporary csv file is created when uploading from a local data.frame

For larger file sizes up to 5TB, upload to Google Cloud Storage first via gcs_upload then supply the object URI of the form `gs://project-name/object-name` to the `upload_data` argument.

You also need to supply a data schema. Remember that the file should not have a header row.

Value

TRUE if successful, FALSE if not.

See Also

urlhttps://cloud.google.com/bigquery/loading-data-post-request

Examples

```
## Not run:

 library(googleCloudStorageR)
 library(bigQueryR)

 gcs_global_bucket("your-project")

 ## custom upload function to ignore quotes and column headers
 f <- function(input, output) {
   write.table(input, sep = ",", col.names = FALSE, row.names = FALSE,
               quote = FALSE, file = output, qmethod = "double")}

 ## upload files to Google Cloud Storage
 gcs_upload(mtcars, name = "mtcars_test1.csv", object_function = f)
 gcs_upload(mtcars, name = "mtcars_test2.csv", object_function = f)

 ## create the schema of the files you just uploaded
 user_schema <- schema_fields(mtcars)

 ## load files from Google Cloud Storage into BigQuery
 bqr_upload_data(projectId = "your-project",
                 datasetId = "test",
                 tableId = "from_gcs_mtcars",
                 upload_data = c("gs://your-project/mtcars_test1.csv",
                                 "gs://your-project/mtcars_test2.csv"),
                 schema = user_schema)

 ## for big files, its helpful to create your schema on a small sample
 ## a quick way to do this on the command line is:
 # "head bigfile.csv > head_bigfile.csv"

## upload nested lists as JSON
the_list <- list(list(col1 = "yes", col2 = "no",
                      col3 = list(nest1 = 1, nest2 = 3), col4 = "oh"),
```

```
                  list(col1 = "yes2",
                       col2 = "n2o", col3 = list(nest1 = 5, nest2 = 7),
                       col4 = "oh2"),
                  list(col1 = "yes3", col2 = "no3",
                       col3 = list(nest1 = 7, nest2 = 55), col4 = "oh3"))

bqr_upload_data(datasetId = "test",
                tableId = "nested_list_json",
                upload_data = the_list,
                autodetect = TRUE)


## End(Not run)
```

`bqr_wait_for_job` *Wait for a bigQuery job*

Description

Wait for a bigQuery job to finish.

Usage

```
bqr_wait_for_job(job, wait = 5)
```

Arguments

`job`	A job object
`wait`	The number of seconds to wait between checks
	Use this function to do a loop to check progress of a job running

Value

After a while, a completed job

See Also

Other BigQuery asynch query functions: `bqr_download_extract`, `bqr_extract_data`, `bqr_get_job`, `bqr_grant_extract_access`, `bqr_query_asynch`

schema_fields *Create data schema for upload to BigQuery*

Description

Use this on a sample of the data you want to load from Google Cloud Storage

Usage

```
schema_fields(data)
```

Arguments

data	An example of the data to create a schema from

Details

This is taken from insert_upload_job

Value

A schema object suitable to pass within the schema argument of bqr_upload_data

Author(s)

Hadley Wickham <hadley@rstudio.com>

Table *Table Object*

Description

Configure table objects as documented by the Google docs for Table objects

Usage

```
Table(tableId, projectId = bqr_get_global_project(),
  datasetId = bqr_get_global_dataset(), clustering = NULL,
  description = NULL, encryptionConfiguration = NULL,
  expirationTime = NULL, friendlyName = NULL, labels = NULL,
  materializedView = NULL, rangePartitioning = NULL,
  requirePartitionFilter = NULL, schema = NULL,
  timePartitioning = NULL, view = NULL)
```

Arguments

tableId	tableId
projectId	projectId
datasetId	datasetId
clustering	[Beta] Clustering specification for the table
description	[Optional] A user-friendly description of this table
encryptionConfiguration	Custom encryption configuration (e
expirationTime	[Optional] The time when this table expires, in milliseconds since the epoch
friendlyName	[Optional] A descriptive name for this table
labels	The labels associated with this table - a named list of key = value
materializedView	[Optional] Materialized view definition
rangePartitioning	[TrustedTester] Range partitioning specification for this table
requirePartitionFilter	[Beta] [Optional] If set to true, queries over this table require a partition filter that can be used for partition elimination to be specified
schema	[Optional] Describes the schema of this table
timePartitioning	Time-based partitioning specification for this table
view	[Optional] The view definition

Details

A table object to be used within bqr_patch_table

Value

Table object

See Also

Other Table meta functions: bqr_copy_table, bqr_create_table, bqr_delete_table, bqr_list_tables, bqr_patch_table, bqr_table_data, bqr_table_meta

Index

∗ **BigQuery asynch query functions**
bqr_download_extract, 8
bqr_extract_data, 10
bqr_get_job, 13
bqr_grant_extract_access, 16
bqr_query_asynch, 23
bqr_wait_for_job, 29
∗ **BigQuery query functions**
bqr_query, 22
∗ **Table meta functions**
bqr_copy_table, 5
bqr_create_table, 6
bqr_delete_table, 7
bqr_list_tables, 20
bqr_patch_table, 22
bqr_table_data, 25
bqr_table_meta, 26
Table, 30
∗ **bigQuery meta functions**
bqr_list_datasets, 17
bqr_list_projects, 19
∗ **bigQuery upload functions**
bqr_upload_data, 27
∗ **dataset functions**
bqr_get_global_dataset, 12
bqr_global_dataset, 14
∗ **project functions**
bqr_get_global_project, 12
bqr_global_project, 15

bigQueryR, 2
bigQueryR-package *(*bigQueryR*)*, 2
bq_get_global_dataset *(*bqr_get_global_dataset*)*, 12
bq_get_global_project *(*bqr_get_global_project*)*, 12
bq_global_dataset, *12*
bq_global_dataset *(*bqr_global_dataset*)*, 14
bq_global_project, *12*
bq_global_project *(*bqr_global_project*)*, 15
bqr_auth, 3
bqr_copy_dataset, 4
bqr_copy_table, *4*, 5, *7, 8*, *20–22*, *25, 26*, *31*
bqr_create_table, *6*, 6, *8*, *20*, *22*, *25, 26*, *31*
bqr_delete_table, *6, 7*, 7, *20*, *22*, *25, 26*, *31*
bqr_download_extract, 8, *10*, *13*, *16*, *24*, *29*
bqr_download_query, 8
bqr_extract_data, *8*, 10, *13*, *16*, *24*, *29*
bqr_get_global_dataset, 11, *15*
bqr_get_global_project, 12, *15*
bqr_get_job, *8*, *10*, 13, *16*, *24*, *29*
bqr_global_dataset, *12*, 14
bqr_global_project, *13*, 15
bqr_grant_extract_access, *8*, *10*, *13*, 16, *24*, *29*
bqr_list_datasets, 17, *19*
bqr_list_jobs, 18
bqr_list_projects, *17*, 19
bqr_list_tables, *6–8*, 20, *22*, *25, 26*, *31*
bqr_partition, 20
bqr_patch_table, *6–8*, *20*, 22, *25, 26*, *31*
bqr_query, *10*, 22, *25*
bqr_query_asynch, *8*, *10*, *13*, *16*, 23, *29*
bqr_table_data, *6–8*, *20*, *22*, 25, *26*, *31*
bqr_table_meta, *6–8*, *20*, *22*, *25*, 26, *31*
bqr_upload_data, 27, *30*
bqr_wait_for_job, *8*, *10*, *13*, *16*, *24*, 29

gcs_upload, *28*

insert_upload_job, *30*

schema_fields, *27*, 30

Table, *6–8*, *20*, *22*, *25, 26*, 30