

# Package ‘bigrquery’

May 12, 2026

**Title** An Interface to Google's 'BigQuery' 'API'

**Version** 1.6.2

**Description** Easily talk to Google's 'BigQuery' database from R.

**License** MIT + file LICENSE

**URL** <https://bigrquery.r-dbi.org>, <https://github.com/r-dbi/bigrquery>

**BugReports** <https://github.com/r-dbi/bigrquery/issues>

**Depends** R (>= 4.1)

**Imports** bit64, brio, cli, clock, curl, DBI, gargle (>= 1.5.0), httr, jsonlite, lifecycle, methods, nanoparquet (>= 0.3.1), prettyunits, rlang (>= 1.1.0), tibble

**Suggests** bigquerystorage (>= 1.2.2), blob, covr, dbplyr (>= 2.4.0), dplyr (>= 1.1.0), hms, readr, sodium, testthat (>= 3.1.5), withr, wk (>= 0.3.2)

**LinkingTo** cli, cpp11, rapidjsonr

**Config/Needs/website** tidyverse/tidytemplate

**Config/testthat/edition** 3

**Config/testthat/parallel** TRUE

**Config/testthat/start-first** bq-table, dplyr, dbi-connection

**Encoding** UTF-8

**Collate** 'bigrquery-package.R' 'bq-auth.R' 'bq-dataset.R' 'bq-download.R' 'bq-field.R' 'bq-job.R' 'bq-param.R' 'bq-parse.R' 'bq-perform.R' 'bq-project.R' 'bq-projects.R' 'bq-query.R' 'bq-refs.R' 'bq-request.R' 'bq-table.R' 'bq-test.R' 'camelCase.R' 'connections-page.R' 'cpp11.R' 'dbi-driver.R' 'dbi-connection.R' 'dbi-result.R' 'dplyr.R' 'gs-object.R' 'import-standalone-obj-type.R' 'import-standalone-s3-register.R' 'import-standalone-types-check.R' 'utils.R' 'zzz.R'

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** yes

**Author** Hadley Wickham [aut, cre] (ORCID: <https://orcid.org/0000-0003-4757-117X>),  
 Jennifer Bryan [aut] (ORCID: <https://orcid.org/0000-0002-6983-2759>),  
 Posit Software, PBC [cph, fnd]

**Maintainer** Hadley Wickham <hadley@posit.co>

**Repository** CRAN

**Date/Publication** 2026-05-12 13:20:09 UTC

## Contents

api-dataset . . . . .	2
api-job . . . . .	4
api-project . . . . .	5
api-table . . . . .	6
bigquery . . . . .	8
bq_auth . . . . .	10
bq_auth_configure . . . . .	12
bq_deauth . . . . .	13
bq_field . . . . .	14
bq_has_token . . . . .	15
bq_projects . . . . .	16
bq_query . . . . .	16
bq_refs . . . . .	18
bq_table_download . . . . .	19
bq_token . . . . .	21
bq_user . . . . .	22
collect.tbl_BigQueryConnection . . . . .	23
src_bigquery . . . . .	24
<b>Index</b>	<b>25</b>

---

api-dataset	<i>BigQuery datasets</i>
-------------	--------------------------

---

## Description

Basic create-read-update-delete verbs for datasets.

## Usage

```
bq_dataset_create(x, location = "US", ...)
```

```
bq_dataset_meta(x, fields = NULL)
```

```
bq_dataset_exists(x)
```

```
bq_dataset_update(x, ...)
```

```
bq_dataset_delete(x, delete_contents = FALSE)
```

```
bq_dataset_tables(x, page_size = 50, max_pages = Inf, warn = TRUE, ...)
```

### Arguments

x	A <a href="#">bq_dataset</a>
location	Dataset location
...	Additional arguments passed on to the underlying API call. snake_case names are automatically converted to camelCase.
fields	An optional field specification for <a href="#">partial response</a>
delete_contents	If TRUE, will recursively delete all tables in the dataset. Set to FALSE by default for safety.
page_size	Number of items per page.
max_pages	Maximum number of pages to retrieve. Use Inf to retrieve all pages (this may take a long time!)
warn	If TRUE, warn when there are unretrieved pages.

### Google BigQuery API documentation

- [get](#)
- [insert](#)
- [delete](#)
- [list](#)

### Examples

```
ds <- bq_dataset(bq_test_project(), "dataset_api")
bq_dataset_exists(ds)

bq_dataset_create(ds)
bq_dataset_exists(ds)
str(bq_dataset_meta(ds))

bq_dataset_delete(ds)
bq_dataset_exists(ds)

# Use bq_test_dataset() to create a temporary dataset that will
# be automatically deleted
ds <- bq_test_dataset()
bq_table_create(bq_table(ds, "x1"))
bq_table_create(bq_table(ds, "x2"))
bq_table_create(bq_table(ds, "x3"))
bq_dataset_tables(ds)
```

api-job

*BigQuery job: retrieve metadata***Description**

To perform a job, see [api-perform](#). These functions all retrieve metadata (in various forms) about an existing job.

**Usage**

```
bq_job_meta(x, fields = NULL)

bq_job_status(x)

bq_job_show_statistics(x)

bq_job_wait(
  x,
  quiet = getOption("bigrquery.quiet"),
  pause = 0.5,
  call = caller_env()
)
```

**Arguments**

x	A <a href="#">bq_job</a>
fields	An optional field specification for <b>partial response</b>
quiet	If FALSE, displays progress bar; if TRUE is silent; if NA picks based on whether or not you're in an interactive context.
pause	amount of time to wait between status requests
call	The execution environment of a currently running function, e.g. <code>caller_env()</code> . The function will be mentioned in error messages as the source of the error. See the <code>call</code> argument of <a href="#">abort()</a> for more information.

**Google BigQuery API documentation**

- [get](#)

**Examples**

```
jobs <- bq_project_jobs(bq_test_project())
jobs[[1]]

# Show statistics about job
bq_job_show_statistics(jobs[[1]])

# Wait for job to complete
```

```
bq_job_wait(jobs[[1]])
```

---

api-project

*BigQuery project methods*

---

## Description

Projects have two primary components: datasets and jobs. Unlike other BigQuery objects, is no accompanying `bq_project` S3 class because a project is a simple string.

## Usage

```
bq_project_datasets(x, page_size = 100, max_pages = 1, warn = TRUE)
```

```
bq_project_jobs(x, page_size = 100, max_pages = 1, warn = TRUE)
```

## Arguments

<code>x</code>	A string giving a project name.
<code>page_size</code>	Number of items per page.
<code>max_pages</code>	Maximum number of pages to retrieve. Use <code>Inf</code> to retrieve all pages (this may take a long time!)
<code>warn</code>	If <code>TRUE</code> , warn when there are unretrieved pages.

## Value

- `bq_project_datasets()`: a list of [bq\\_datasets](#)
- `bq_project_jobs()`: a list of [bq\\_jobs](#).

## Google BigQuery API documentation

- [datasets](#)
- [jobs](#)

## Examples

```
bq_project_datasets("bigquery-public-data")
bq_project_datasets("githubarchive")

bq_project_jobs(bq_test_project(), page_size = 10)
```

---

`api-table`*BigQuery tables*

---

### Description

Basic create-read-update-delete verbs for tables, as well as functions uploading data (`bq_table_upload()`), saving to/loading from Google Cloud Storage (`bq_table_load()`, `bq_table_save()`), and getting various values from the metadata.

### Usage

```
bq_table_create(x, fields = NULL, ...)
```

```
bq_table_meta(x, fields = NULL)
```

```
bq_table_fields(x)
```

```
bq_table_size(x)
```

```
bq_table_nrow(x)
```

```
bq_table_exists(x)
```

```
bq_table_delete(x)
```

```
bq_table_copy(x, dest, ..., quiet = getOption("bigrquery.quiet", NA))
```

```
bq_table_upload(  
  x,  
  values,  
  ...,  
  quiet = getOption("bigrquery.quiet", NA),  
  json_digits = NULL  
)
```

```
bq_table_save(  
  x,  
  destination_uris,  
  ...,  
  quiet = getOption("bigrquery.quiet", NA)  
)
```

```
bq_table_load(x, source_uris, ..., quiet = getOption("bigrquery.quiet", NA))
```

```
bq_table_patch(x, fields)
```

**Arguments**

x	A <a href="#">bq_table</a> , or an object coercible to a <code>bq_table</code> .
fields	A <a href="#">bq_fields</a> specification, or something coercible to it (like a data frame).
...	Additional arguments passed on to the underlying API call. <code>snake_case</code> names are automatically converted to <code>camelCase</code> .
dest	Source and destination <a href="#">bq_tables</a> .
quiet	If FALSE, displays progress bar; if TRUE is silent; if NA picks based on whether or not you're in an interactive context.
values	Data frame of values to insert.
json_digits	Specifies the number of digits for formatting numeric values.
destination_uris	A character vector of fully-qualified Google Cloud Storage URIs where the extracted table should be written. Can export up to 1 Gb of data per file. Use a wild card URI (e.g. <code>gs://[YOUR_BUCKET]/file-name-*.json</code> ) to automatically create any number of files.
source_uris	The fully-qualified URIs that point to your data in Google Cloud. For Google Cloud Storage URIs: Each URI can contain one '*' wildcard character and it must come after the 'bucket' name. Size limits related to load jobs apply to external data sources. For Google Cloud Bigtable URIs: Exactly one URI can be specified and it has to be a fully specified and valid HTTPS URL for a Google Cloud Bigtable table. For Google Cloud Datastore backups: Exactly one URI can be specified. Also, the '*' wildcard character is not allowed.

**Value**

- `bq_table_copy()`, `bq_table_create()`, `bq_table_delete()`, `bq_table_upload()`: an invisible [bq\\_table](#)
- `bq_table_exists()`: either TRUE or FALSE.
- `bq_table_size()`: the size of the table in bytes
- `bq_table_fields()`: a [bq\\_fields](#).

**Google BigQuery API documentation**

- [insert](#)
- [get](#)
- [delete](#)

**Examples**

```
ds <- bq_test_dataset()

bq_mtcars <- bq_table(ds, "mtcars")
bq_table_exists(bq_mtcars)
```

```

bq_table_create(
  bq_mtcars,
  fields = mtcars,
  friendly_name = "Motor Trend Car Road Tests",
  description = "The data was extracted from the 1974 Motor Trend US magazine",
  labels = list(category = "example")
)
bq_table_exists(bq_mtcars)

bq_table_upload(bq_mtcars, mtcars)

bq_table_fields(bq_mtcars)
bq_table_size(bq_mtcars)
str(bq_table_meta(bq_mtcars))

bq_table_delete(bq_mtcars)
bq_table_exists(bq_mtcars)

my_natality <- bq_table(ds, "mynatality")
bq_table_copy("publicdata.samples.natality", my_natality)

```

---

bigquery

*BigQuery DBI driver*


---

## Description

Creates a BigQuery DBI driver for use in `DBI::dbConnect()`.

## Usage

```

## S4 method for signature 'BigQueryDriver'
dbConnect(
  drv,
  project,
  dataset = NULL,
  billing = project,
  page_size = 10000,
  quiet = NA,
  use_legacy_sql = FALSE,
  bigint = c("integer", "integer64", "numeric", "character"),
  labels = getOption("bigquery.labels"),
  ...
)

```

## Arguments

`drv` The result of `bigquery()`.

project, dataset	Project and dataset identifiers
billing	Identifier of project to bill.
page_size	Number of items per page.
quiet	If FALSE, displays progress bar; if TRUE is silent; if NA picks based on whether or not you're in an interactive context.
use_legacy_sql	If TRUE will use BigQuery's legacy SQL format.
bigint	The R type that BigQuery's 64-bit integer types should be mapped to. The default is "integer" which returns R's integer type but results in NA for values above/below +/- 2147483647. "integer64" returns a <code>bit64::integer64</code> , which allows the full range of 64 bit integers.
labels	A named list of strings used to attach <b>BigQuery labels</b> to the resulting job, e.g. <code>list(env = "prod", team = "data")</code> . This is most useful for cost allocation and other FinOps reporting. Defaults to the value of <code>getOption("bigquery.labels")</code> .
...	Other arguments for compatibility with generic; currently ignored.

## Examples

```

con <- DBI::dbConnect(
  bigquery(),
  project = "publicdata",
  dataset = "samples",
  billing = bq_test_project()
)
con
DBI::dbListTables(con)
DBI::dbReadTable(con, "natality", n_max = 10)

# Create a temporary dataset to explore
ds <- bq_test_dataset()
con <- DBI::dbConnect(
  bigquery(),
  project = ds$project,
  dataset = ds$dataset
)
DBI::dbWriteTable(con, "mtcars", mtcars)
DBI::dbReadTable(con, "mtcars")[1:6, ]

DBI::dbGetQuery(con, "SELECT count(*) FROM mtcars")

res <- DBI::dbSendQuery(con, "SELECT cyl, mpg FROM mtcars")
dbColumnInfo(res)
dbFetch(res, 10)
dbFetch(res, -1)
DBI::dbHasCompleted(res)

```

---

bq_auth	<i>Authorize bigquery</i>
---------	---------------------------

---

### Description

Authorize bigquery to view and manage your BigQuery projects. This function is a wrapper around `gargle::token_fetch()`.

By default, you are directed to a web browser, asked to sign in to your Google account, and to grant bigquery permission to operate on your behalf with Google BigQuery. By default, with your permission, these user credentials are cached in a folder below your home directory, from where they can be automatically refreshed, as necessary. Storage at the user level means the same token can be used across multiple projects and tokens are less likely to be synced to the cloud by accident.

### Usage

```
bq_auth(
  email = gargle::gargle_oauth_email(),
  path = NULL,
  scopes = c("https://www.googleapis.com/auth/bigquery",
             "https://www.googleapis.com/auth/cloud-platform"),
  cache = gargle::gargle_oauth_cache(),
  use_oob = gargle::gargle_oob_default(),
  token = NULL
)
```

### Arguments

email	<p>Optional. If specified, email can take several different forms:</p> <ul style="list-style-type: none"> <li>• "jane@gmail.com", i.e. an actual email address. This allows the user to target a specific Google identity. If specified, this is used for token lookup, i.e. to determine if a suitable token is already available in the cache. If no such token is found, email is used to pre-select the targeted Google identity in the OAuth chooser. (Note, however, that the email associated with a token when it's cached is always determined from the token itself, never from this argument).</li> <li>• "*@example.com", i.e. a domain-only glob pattern. This can be helpful if you need code that "just works" for both alice@example.com and bob@example.com.</li> <li>• TRUE means that you are approving email auto-discovery. If exactly one matching token is found in the cache, it will be used.</li> <li>• FALSE or NA mean that you want to ignore the token cache and force a new OAuth dance in the browser.</li> </ul> <p>Defaults to the option named "gargle_oauth_email", retrieved by <code>gargle_oauth_email()</code> (unless a wrapper package implements different default behavior).</p>
path	<p>JSON identifying the service account, in one of the forms supported for the txt argument of <code>jsonlite::fromJSON()</code> (typically, a file path or JSON string).</p>

scopes	A character vector of scopes to request. Pick from those listed at <a href="https://developers.google.com/identity/protocols/oauth2/scopes">https://developers.google.com/identity/protocols/oauth2/scopes</a> .
cache	Specifies the OAuth token cache. Defaults to the option named "gargle_oauth_cache", retrieved via <code>gargle_oauth_cache()</code> .
use_oob	Whether to use out-of-band authentication (or, perhaps, a variant implemented by gargle and known as "pseudo-OOB") when first acquiring the token. Defaults to the value returned by <code>gargle_oob_default()</code> . Note that (pseudo-)OOB auth only affects the initial OAuth dance. If we retrieve (and possibly refresh) a cached token, use_oob has no effect.  If the OAuth client is provided implicitly by a wrapper package, its type probably defaults to the value returned by <code>gargle_oauth_client_type()</code> . You can take control of the client type by setting <code>options(gargle_oauth_client_type = "web")</code> or <code>options(gargle_oauth_client_type = "installed")</code> .
token	A token with class <code>Token2.0</code> or an object of <code>httr</code> 's class <code>request</code> , i.e. a token that has been prepared with <code>httr::config()</code> and has a <code>Token2.0</code> in the <code>auth_token</code> component.

## Details

Most users, most of the time, do not need to call `bq_auth()` explicitly – it is triggered by the first action that requires authorization. Even when called, the default arguments often suffice.

However, when necessary, `bq_auth()` allows the user to explicitly:

- Declare which Google identity to use, via an email specification.
- Use a service account token or workload identity federation via path.
- Bring your own token.
- Customize scopes.
- Use a non-default cache folder or turn caching off.
- Explicitly request out-of-band (OOB) auth via `use_oob`.

If you are interacting with R within a browser (applies to RStudio Server, Posit Workbench, Posit Cloud, and Google Colaboratory), you need OOB auth or the pseudo-OOB variant. If this does not happen automatically, you can request it explicitly with `use_oob = TRUE` or, more persistently, by setting an option via `options(gargle_oob_default = TRUE)`.

The choice between conventional OOB or pseudo-OOB auth is determined by the type of OAuth client. If the client is of the "installed" type, `use_oob = TRUE` results in conventional OOB auth. If the client is of the "web" type, `use_oob = TRUE` results in pseudo-OOB auth. Packages that provide a built-in OAuth client can usually detect which type of client to use. But if you need to set this explicitly, use the "gargle\_oauth\_client\_type" option:

```
options(gargle_oauth_client_type = "web")      # pseudo-OOB
# or, alternatively
options(gargle_oauth_client_type = "installed") # conventional OOB
```

For details on the many ways to find a token, see `gargle::token_fetch()`. For deeper control over auth, use `bq_auth_configure()` to bring your own OAuth client or API key. To learn more about gargle options, see `gargle::gargle_options`.

## See Also

Other auth functions: [bq\\_auth\\_configure\(\)](#), [bq\\_deauth\(\)](#)

## Examples

```
## Not run:
## load/refresh existing credentials, if available
## otherwise, go to browser for authentication and authorization
bq_auth()

## force use of a token associated with a specific email
bq_auth(email = "jenny@example.com")

## force a menu where you can choose from existing tokens or
## choose to get a new one
bq_auth(email = NA)

## use a 'read only' scope, so it's impossible to change data
bq_auth(
  scopes = "https://www.googleapis.com/auth/devstorage.read_only"
)

## use a service account token
bq_auth(path = "foofy-83ee9e7c9c48.json")

## End(Not run)
```

---

[bq\\_auth\\_configure](#)

*Edit and view auth configuration*

---

## Description

These functions give more control over and visibility into the auth configuration than [bq\\_auth\(\)](#) does. [bq\\_auth\\_configure\(\)](#) lets the user specify their own:

- OAuth client, which is used when obtaining a user token.

See the vignette("get-api-credentials", package = "gargle") for more. If the user does not configure these settings, internal defaults are used.

[bq\\_oauth\\_client\(\)](#) retrieves the currently configured OAuth client.

## Usage

```
bq_auth_configure(client, path)
```

```
bq_oauth_client()
```

**Arguments**

client	A Google OAuth client, presumably constructed via <code>gargle::gargle_oauth_client_from_json()</code> . Note, however, that it is preferred to specify the client with JSON, using the path argument.
path	JSON downloaded from <a href="#">Google Cloud Console</a> , containing a client id and secret, in one of the forms supported for the <code>txt</code> argument of <code>jsonlite::fromJSON()</code> (typically, a file path or JSON string).

**Value**

- `bq_auth_configure()`: An object of R6 class `gargle::AuthState`, invisibly.
- `bq_oauth_client()`: the current user-configured OAuth client.

**See Also**

Other auth functions: `bq_auth()`, `bq_deauth()`

**Examples**

```
# see and store the current user-configured OAuth client (probably `NULL`)
(original_client <- bq_oauth_client())

# the preferred way to configure your own client is via a JSON file
# downloaded from Google Developers Console
# this example JSON is indicative, but fake
path_to_json <- system.file(
  "extdata", "data", "client_secret_123.googleusercontent.com.json",
  package = "bigrquery"
)
bq_auth_configure(path = path_to_json)

# confirm the changes
bq_oauth_client()

# restore original auth config
bq_auth_configure(client = original_client)
```

---

bq\_deauth

*Clear current token*


---

**Description**

Clears any currently stored token. The next time bigrquery needs a token, the token acquisition process starts over, with a fresh call to `bq_auth()` and, therefore, internally, a call to `gargle::token_fetch()`. Unlike some other packages that use gargle, bigrquery is not usable in a de-authorized state. Therefore, calling `bq_deauth()` only clears the token, i.e. it does NOT imply that subsequent requests are made with an API key in lieu of a token.

**Usage**

```
bq_deauth()
```

**See Also**

Other auth functions: [bq\\_auth\(\)](#), [bq\\_auth\\_configure\(\)](#)

**Examples**

```
## Not run:
bq_deauth()

## End(Not run)
```

---

bq_field	<i>BigQuery field (and fields) class</i>
----------	--

---

**Description**

bq\_field() and bq\_fields() create; as\_bq\_field() and as\_bq\_fields() coerce from lists.

**Usage**

```
bq_field(name, type, mode = "NULLABLE", fields = list(), description = NULL)
```

```
bq_fields(x)
```

```
as_bq_field(x)
```

```
as_bq_fields(x)
```

**Arguments**

name	The field name. The name must contain only letters (a-z, A-Z), numbers (0-9), or underscores (_), and must start with a letter or underscore. The maximum length is 300 characters.
type	The field data type. Possible values include: "STRING", "BYTES", "INTEGER", "FLOAT", "BOOLEAN", "TIMESTAMP", "DATE", "TIME", "DATETIME", "GEOGRAPHY", "NUMERIC", "BIGNUMERIC", "JSON", "RECORD".
mode	The field mode. Possible values include: "NULLABLE", "REQUIRED", and "REPEATED".
fields	For a field of type "record", a list of sub-fields.
description	The field description. The maximum length is 1,024 characters.
x	A list of bq_fields

**See Also**

bq\_field() corresponds to a TableFieldSchema, see <https://docs.cloud.google.com/bigquery/docs/reference/rest/v2/tables#TableFieldSchema> for more details.

**Examples**

```
bq_field("name", "string")

as_bq_fields(list(
  list(name = "name", type = "string"),
  bq_field("age", "integer")
))

# as_bq_fields() can also take a data frame
as_bq_fields(mtcars)
```

---

bq_has_token	<i>Is there a token on hand?</i>
--------------	----------------------------------

---

**Description**

Reports whether bigquery has stored a token, ready for use in downstream requests.

**Usage**

```
bq_has_token()
```

**Value**

Logical.

**See Also**

Other low-level API functions: [bq\\_token\(\)](#)

**Examples**

```
bq_has_token()
```

---

bq_projects	<i>List available projects</i>
-------------	--------------------------------

---

### Description

List all projects that you have access to. You can also work with [public datasets](#), but you will need to provide a billing project whenever you perform any non-free operation.

### Usage

```
bq_projects(page_size = 100, max_pages = 1, warn = TRUE)
```

### Arguments

page_size	Number of items per page.
max_pages	Maximum number of pages to retrieve. Use Inf to retrieve all pages (this may take a long time!)
warn	If TRUE, warn when there are unretrieved pages.

### Value

A character vector.

### Google BigQuery API documentation

- [list](#)

### Examples

```
bq_projects()
```

---

bq_query	<i>Submit query to BigQuery</i>
----------	---------------------------------

---

### Description

These submit a query (using [bq\\_perform\\_query\(\)](#)) and then wait for it complete (with [bq\\_job\\_wait\(\)](#)). All BigQuery queries save their results into a table (temporary or otherwise), so these functions return a [bq\\_table](#) which you can then query for more information.

**Usage**

```
bq_project_query(
  x,
  query,
  destination_table = NULL,
  ...,
  quiet = getOption("bigrquery.quiet", NA)
)
```

```
bq_dataset_query(
  x,
  query,
  destination_table = NULL,
  ...,
  billing = NULL,
  quiet = getOption("bigrquery.quiet", NA)
)
```

**Arguments**

x	Either a project (a string) or a <a href="#">bq_dataset</a> .
query	SQL query string.
destination_table	A <a href="#">bq_table</a> where results should be stored. If not supplied, results will be saved to a temporary table that lives in a special dataset. You must supply this parameter for large queries (> 128 MB compressed).
...	Passed on to <a href="#">bq_perform_query()</a>
quiet	If FALSE, displays progress bar; if TRUE is silent; if NA picks based on whether or not you're in an interactive context.
billing	If you query a dataset that you only have read access for, such as a public dataset, you must also submit a billing project.

**Value**

A [bq\\_table](#)

**Examples**

```
# Querying a project requires full name in query
tb <- bq_project_query(
  bq_test_project(),
  "SELECT count(*) FROM publicdata.samples.natality"
)
bq_table_fields(tb)
bq_table_download(tb)

# Querying a dataset sets default dataset so you can use bare table name,
# but for public data, you'll need to set a project to bill.
```

```

ds <- bq_dataset("publicdata", "samples")
tb <- bq_dataset_query(ds,
  query = "SELECT count(*) FROM natality",
  billing = bq_test_project()
)
bq_table_download(tb)

tb <- bq_dataset_query(ds,
  query = "SELECT count(*) FROM natality WHERE state = @state",
  parameters = list(state = "KS"),
  billing = bq_test_project()
)
bq_table_download(tb)

```

---

bq\_refs

*S3 classes for BigQuery datasets, tables and jobs*


---

## Description

Create references to BigQuery datasets, jobs, and tables. Each class has a constructor function (`bq_dataset()`, `bq_table()`, `bq_job()`) and a coercion function (`as_bq_dataset()`, `as_bq_table()`, `as_bq_job()`). The coercion functions come with methods for strings (which find components by splitting on `.`), and lists (which look for named components like `projectId` or `project_id`).

All `bq_table_`, `bq_dataset_` and `bq_job_` functions call the appropriate coercion functions on their first argument, allowing you to flexibly specify their inputs.

## Usage

```
bq_dataset(project, dataset)
```

```
as_bq_dataset(x, ..., error_arg = caller_arg(x), error_call = caller_env())
```

```
bq_table(project, dataset, table = NULL, type = "TABLE")
```

```
as_bq_table(x, ..., error_arg = caller_arg(x), error_call = caller_env())
```

```
bq_job(project, job, location = "US")
```

```
as_bq_job(x, ..., error_arg = caller_arg(x), error_call = caller_env())
```

## Arguments

`project`, `dataset`, `table`, `job`, `type`

Individual project, dataset, table, job identifiers and table type (strings).

For `bq_table()`, you if supply a `bq_dataset` as the first argument, the 2nd argument will be interpreted as the table

x	An object to coerce to a bq_job, bq_dataset, or bq_table. Built-in methods handle strings and lists.
...	Other arguments passed on to methods.
error_arg	An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
error_call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of <a href="#">abort()</a> for more information.
location	Job location

### See Also

[api-job](#), [api-perform](#), [api-dataset](#), and [api-table](#) for functions that work with these objects.

### Examples

```
# Creation -----
samples <- bq_dataset("publicdata", "samples")
natality <- bq_table("publicdata", "samples", "natality")
natality

# Or
bq_table(samples, "natality")

bq_job("bigquery-examples", "m0SgFu2ycbbge6jgcvzvf1BJ_Wft")

# Coercion -----
as_bq_dataset("publicdata.shakespeare")
as_bq_table("publicdata.samples.natality")

as_bq_table(list(
  project_id = "publicdata",
  dataset_id = "samples",
  table_id = "natality"
))

as_bq_job(list(
  projectId = "bigquery-examples",
  jobId = "job_m0SgFu2ycbbge6jgcvzvf1BJ_Wft",
  location = "US"
))
```

## Description

This function provides two ways to download data from BigQuery, transferring data using either JSON or arrow, depending on the `api` argument. If `bigrquyrstorage` is installed, `api = "arrow"` will be used (because it's so much faster, but see the limitations below), otherwise you can select deliberately by using `api = "json"` or `api = "arrow"`.

### Arrow API:

The arrow API is much faster, but has heavier dependencies: `bigrquyrstorage` requires the arrow package, which can be tricky to compile on Linux (but you usually should be able to get a binary from [Posit Public Package Manager](#)).

There's one known limitation of `api = "arrow"`: when querying public data, you'll now need to provide a billing project.

### JSON API:

The JSON API retrieves rows in chunks of `page_size`. It is most suitable for results of smaller queries (<100 MB, say). Unfortunately due to limitations in the BigQuery API, you may need to vary this parameter depending on the complexity of the underlying data.

The JSON API will convert nested and repeated columns in to list-columns as follows:

- Repeated values (arrays) will become a list-column of vectors.
- Records will become list-columns of named lists.
- Repeated records will become list-columns of data frames.

## Usage

```
bq_table_download(
  x,
  n_max = Inf,
  page_size = NULL,
  start_index = 0L,
  max_connections = 6L,
  quiet = getOption("bigrquery.quiet", NA),
  bigint = c("integer", "integer64", "numeric", "character"),
  api = c("json", "arrow"),
  billing = x$project
)
```

## Arguments

<code>x</code>	A <a href="#">bq_table</a>
<code>n_max</code>	Maximum number of results to retrieve. Use <code>Inf</code> to retrieve all rows.
<code>page_size</code>	(JSON only) The number of rows requested per chunk. It is recommended to leave this unspecified until you have evidence that the <code>page_size</code> selected automatically by <code>bq_table_download()</code> is problematic.  When <code>page_size = NULL</code> <code>bigrquery</code> determines a conservative, natural chunk size empirically. If you specify the <code>page_size</code> , it is important that each chunk fits on one page, i.e. that the requested row limit is low enough to prevent the API from paginating based on response size.

start_index	(JSON only) Starting row index (zero-based).
max_connections	(JSON only) Number of maximum simultaneous connections to BigQuery servers.
quiet	If FALSE, displays progress bar; if TRUE is silent; if NA picks based on whether or not you're in an interactive context.
bigint	The R type that BigQuery's 64-bit integer types should be mapped to. The default is "integer", which returns R's integer type, but results in NA for values above/below +/- 2147483647. "integer64" returns a <code>bit64::integer64</code> , which allows the full range of 64 bit integers.
api	Which API to use? The "json" API works where ever bigquery does, but is slow and can require fiddling with the <code>page_size</code> parameter. The "arrow" API is faster and more reliable, but only works if you have also installed the <code>bigquerystorage</code> package. Because the "arrow" API is so much faster, it will be used automatically if the <code>bigquerystorage</code> package is installed.
billing	(Arrow only) Project to bill; defaults to the project of <code>x</code> , and typically only needs to be specified if you're working with public datasets.

### Value

Because data retrieval may generate list-columns and the `data.frame` print method can have problems with list-columns, this method returns a tibble. If you need a `data.frame`, coerce the results with `as.data.frame()`.

### Google BigQuery API documentation

- [list](#)

### Examples

```
df <- bq_table_download("publicdata.samples.natality", n_max = 35000, billing = bq_test_project())
```

---

bq_token	<i>Produce configured token</i>
----------	---------------------------------

---

### Description

For internal use or for those programming around the BigQuery API. Returns a token pre-processed with `httr::config()`. Most users do not need to handle tokens "by hand" or, even if they need some control, `bq_auth()` is what they need. If there is no current token, `bq_auth()` is called to either load from cache or initiate OAuth2.0 flow. If auth has been deactivated via `bq_deauth()`, `bq_token()` returns NULL.

### Usage

```
bq_token()
```

**Value**

A request object (an S3 class provided by [httr](#)).

**See Also**

Other low-level API functions: [bq\\_has\\_token\(\)](#)

**Examples**

```
## Not run:  
bq_token()  
  
## End(Not run)
```

---

bq\_user

*Get info on current user*

---

**Description**

Reveals the email address of the user associated with the current token. If no token has been loaded yet, this function does not initiate auth.

**Usage**

```
bq_user()
```

**Value**

An email address or, if no token has been loaded, NULL.

**See Also**

[gargle::token\\_userinfo\(\)](#), [gargle::token\\_email\(\)](#), [gargle::token\\_tokeninfo\(\)](#)

**Examples**

```
## Not run:  
bq_user()  
  
## End(Not run)
```

---

```
collect.tbl_BigQueryConnection
      Collect a BigQuery table
```

---

## Description

This collect method is specialised for BigQuery tables, generating the SQL from your dplyr commands, then calling `bq_project_query()` or `bq_dataset_query()` to run the query, then `bq_table_download()` to download the results. Thus the arguments are a combination of the arguments to `dplyr::collect()`, `bq_project_query()/bq_dataset_query()`, and `bq_table_download()`.

## Usage

```
## S3 method for class 'tbl_BigQueryConnection'
collect(
  x,
  ...,
  n = Inf,
  api = c("json", "arrow"),
  page_size = NULL,
  max_connections = 6L
)
```

## Arguments

<code>x</code>	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from <code>dbplyr</code> or <code>dtplyr</code> ). See <i>Methods</i> , below, for more details.
<code>...</code>	Other arguments passed on to <code>bq_project_query()/bq_project_query()</code>
<code>n</code>	Maximum number of results to retrieve. The default, <code>Inf</code> , will retrieve all rows.
<code>api</code>	Which API to use? The "json" API works where ever bigquery does, but is slow and can require fiddling with the <code>page_size</code> parameter. The "arrow" API is faster and more reliable, but only works if you have also installed the <code>bigquerystorage</code> package. Because the "arrow" API is so much faster, it will be used automatically if the <code>bigquerystorage</code> package is installed.
<code>page_size</code>	(JSON only) The number of rows requested per chunk. It is recommended to leave this unspecified until you have evidence that the <code>page_size</code> selected automatically by <code>bq_table_download()</code> is problematic. When <code>page_size = NULL</code> bigquery determines a conservative, natural chunk size empirically. If you specify the <code>page_size</code> , it is important that each chunk fits on one page, i.e. that the requested row limit is low enough to prevent the API from paginating based on response size.
<code>max_connections</code>	(JSON only) Number of maximum simultaneous connections to BigQuery servers.

---

src_bigquery	<i>A BigQuery data source for dplyr.</i>
--------------	--

---

### Description

Create the connection to the database with `DBI::dbConnect()` then use `dplyr::tbl()` to connect to tables within that database. Generally, it's best to provide the fully qualified name of the table (i.e. `project.dataset.table`) but if you supply a default dataset in the connection, you can use just the table name. (This, however, will prevent you from making joins across datasets.)

### Usage

```
src_bigquery(project, dataset, billing = project, max_pages = 10)
```

### Arguments

project	project id or name
dataset	dataset name
billing	billing project, if different to project
max_pages	(IGNORED) maximum pages returned by a query

### Examples

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

# To run this example, replace billing with the id of one of your projects
# set up for billing
con <- DBI::dbConnect(bigquery(), project = bq_test_project())

shakespeare <- con %>% tbl(I("publicdata.samples.shakespeare"))
shakespeare
shakespeare %>%
  group_by(word) %>%
  summarise(n = sum(word_count, na.rm = TRUE)) %>%
  arrange(desc(n))

## End(Not run)
```

# Index

## \* auth functions

bq\_auth, 10  
bq\_auth\_configure, 12  
bq\_deauth, 13

## \* low-level API functions

bq\_has\_token, 15  
bq\_token, 21

abort(), 4, 19

api-dataset, 2, 19

api-job, 4, 19

api-perform, 4, 19

api-project, 5

api-table, 6, 19

as.data.frame(), 21

as\_bq\_dataset (bq\_refs), 18

as\_bq\_field (bq\_field), 14

as\_bq\_fields (bq\_field), 14

as\_bq\_job (bq\_refs), 18

as\_bq\_table (bq\_refs), 18

bigquery, 8

bit64::integer64, 9, 21

bq\_auth, 10

bq\_auth(), 12–14, 21

bq\_auth\_configure, 12

bq\_auth\_configure(), 11, 12, 14

bq\_dataset, 3, 5, 17

bq\_dataset (bq\_refs), 18

bq\_dataset\_create (api-dataset), 2

bq\_dataset\_delete (api-dataset), 2

bq\_dataset\_exists (api-dataset), 2

bq\_dataset\_meta (api-dataset), 2

bq\_dataset\_query (bq\_query), 16

bq\_dataset\_query(), 23

bq\_dataset\_tables (api-dataset), 2

bq\_dataset\_update (api-dataset), 2

bq\_deauth, 13

bq\_deauth(), 12, 13, 21

bq\_field, 14

bq\_fields, 7

bq\_fields (bq\_field), 14

bq\_has\_token, 15

bq\_has\_token(), 22

bq\_job, 4, 5

bq\_job (bq\_refs), 18

bq\_job\_meta (api-job), 4

bq\_job\_show\_statistics (api-job), 4

bq\_job\_status (api-job), 4

bq\_job\_wait (api-job), 4

bq\_job\_wait(), 16

bq\_oauth\_client (bq\_auth\_configure), 12

bq\_perform\_query(), 16, 17

bq\_project\_datasets (api-project), 5

bq\_project\_jobs (api-project), 5

bq\_project\_query (bq\_query), 16

bq\_project\_query(), 23

bq\_projects, 16

bq\_query, 16

bq\_refs, 18

bq\_table, 7, 16, 17, 20

bq\_table (bq\_refs), 18

bq\_table\_copy (api-table), 6

bq\_table\_create (api-table), 6

bq\_table\_delete (api-table), 6

bq\_table\_download, 19

bq\_table\_download(), 23

bq\_table\_exists (api-table), 6

bq\_table\_fields (api-table), 6

bq\_table\_load (api-table), 6

bq\_table\_meta (api-table), 6

bq\_table\_nrow (api-table), 6

bq\_table\_patch (api-table), 6

bq\_table\_save (api-table), 6

bq\_table\_size (api-table), 6

bq\_table\_upload (api-table), 6

bq\_token, 21

bq\_token(), 15

bq\_user, 22

`collect.tbl_BigQueryConnection`, [23](#)

`dbConnect`, `BigQueryDriver`-method  
    (`bigquery`), [8](#)

`DBI::dbConnect()`, [8](#)

`dbi_driver` (`bigquery`), [8](#)

`dplyr::collect()`, [23](#)

`dplyr::tbl()`, [24](#)

`gargle::AuthState`, [13](#)

`gargle::gargle_oauth_client_from_json()`,  
    [13](#)

`gargle::gargle_options`, [11](#)

`gargle::token_email()`, [22](#)

`gargle::token_fetch()`, [10](#), [11](#), [13](#)

`gargle::token_tokeninfo()`, [22](#)

`gargle::token_userinfo()`, [22](#)

`gargle_oauth_cache()`, [11](#)

`gargle_oauth_client_type()`, [11](#)

`gargle_oauth_email()`, [10](#)

`gargle_oob_default()`, [11](#)

`httr`, [22](#)

`httr::config()`, [11](#), [21](#)

`jsonlite::fromJSON()`, [10](#), [13](#)

`src_bigquery`, [24](#)

`Token2.0`, [11](#)