

Package ‘basedosdados’

March 15, 2022

Title 'Base Dos Dados' R Client

Version 0.2.1

Description An R interface to the 'Base dos Dados' API <https://basedosdados.github.io/mais/py_reference_api/>. Authenticate your project, query our tables, save data to disk and memory, all from R.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.2

Imports purrr (>= 0.3.4), dplyr (>= 1.0.6), tibble (>= 3.1.1), httr (>= 1.4.2), googleAuthR (>= 1.4.0), cli (>= 2.5.0), magrittr (>= 2.0.1), readr (>= 1.4.0), stringr (>= 1.4.0), dotenv (>= 1.0.2), bigquery (>= 1.4.0), glue (>= 1.4.2), rlang (>= 0.4.0), writexl (>= 1.4.0), fs (>= 1.5.0), dbplyr (>= 2.1.1), scales (>= 1.1.1), DBI (>= 1.1.1), typed (>= 0.0.1), Rcpp (>= 1.0.7), xml2 (>= 1.3.0)

Suggests rmarkdown, knitr, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Pedro Cavalcante [aut, cre]

Maintainer Pedro Cavalcante <pedrocolrj@gmail.com>

Repository CRAN

Date/Publication 2022-03-15 08:10:15 UTC

R topics documented:

bdplyr	2
bd_collect	4
bd_write	5
download	8
get_billing_id	9
get_dataset_description	10

get_table_description	10
list_dataset_tables	11
partition_table	12
read_sql	13
set_billing_id	14

Index	15
--------------	-----------

bdplyr

Compatibility with dplyr verbs without using SQL language

Description

Allow you to explore and perform operation with Base dos Dados' datasets without using SQL language. The `bdplyr()` function creates lazy variables that will be connected directly to the desired table from Base dos Dados at Google BigQuery and can be handled with the `dplyr::dplyr-package`'s verbs as traditionally done as local bases. See also: `bigquery::src_bigquery`.

Therefore, it is possible (without using SQL) to perform, for example, column selection with `dplyr::select()`, filter rows with `dplyr::filter()`, operations with `dplyr::mutate()`, joins with `dplyr::left_join()` and other verbs from `{dplyr}` package.

The data will be automatically be downloaded from Google BigQuery in the background as it if necessary, but will not be loaded into your virtual memory nor recorded on disk unless expressly requested.

For this, the functions such as `bd_collect()` or `bd_write()` should be used. To load the data handled locally in your virtual memory, use `bd_collect()`. To save the result in disk use the broader function `bd_write()` or its derivatives `bd_write_csv()` or `bd_write_rds()` to save, respectively in `.csv` or `.rds` format.

Usage

```
bdplyr(
  table,
  billing_project_id = basedosdados::get_billing_id(),
  query_project_id = "basedosdados"
)
```

Arguments

`table` String in the format `(dataset_name).(table_name)`. You can optionally input a project before the dataset name.

`billing_project_id` a string containing your billing project id. If you've run `set_billing_id()` then feel free to leave this empty.

`query_project_id` The project name at GoogleBigQuery. By default `basedosdados`. You do not need to inform this if project is used on `table` parameter.

Value

A lazy tibble, which can be handled (almost) as if were a local database. After satisfactorily handled, the result must be loaded into memory using `bd_collect()` or written to disk using `bd_write()` or its derivatives.

See Also

`bd_collect()`, `bd_write()`, `bd_write_rds()`, `bd_write_rds()`, `bigquery::src_bigquery`

Examples

```
## Not run:

# set project billing id
basedosdados::set_billing_id("avalidprojectbillingid")

# connects to the remote table I want
base_sim <- bdplyr("br_ms_sim.municipio_causa_idade")

# connects to another remote table
municipios <- bdplyr("br_bd_diretorios_brasil.municipio")

# explore data
base_sim %>%
  dplyr::glimpse()

# use normal `{dplyr}` operations
municipios %>%
  head()

# filter
base_sim_acre <- base_sim %>%
  dplyr::mutate(ano = as.numeric(ano)) %>%
  dplyr::filter(sigla_uf == "AC", ano >= 2018)

municipios_acre <- municipios %>%
  dplyr::filter(sigla_uf == "AC") %>%
  dplyr::select(id_municipio, municipio, regioao)

# join
base_junta <- base_sim_acre %>%
  dplyr::left_join(municipios_acre,
                  by = "id_municipio")

# tests whether the result is satisfactory
base_junta

# collect the result
base_final <- base_junta %>%
  basedosdados::bd_collect()
```

```
# alternatively, write in disk the result

base_final %>%
  basedosdados::bd_write_rds(path = "data-raw/data.rds")

## End(Not run)
```

bd_collect	<i>Collects the results of a remote table called via <code>bdplyr()</code></i>
------------	--

Description

After `bdplyr()` is used to create the remote connection, this function allows you to collect the result of the manipulations carried out with the `dplyr`'s verbs and thus use it in local memory completely.

Alternatively, you can also save to disk directly using `bd_write()` function or its derivatives: `bd_write_csv()` or `bd_write_rds()`.

Usage

```
bd_collect(
  .lazy_tbl,
  billing_project_id = basedosdados::get_billing_id(),
  show_query = FALSE
)
```

Arguments

<code>.lazy_tbl</code>	A variable that contains a database that was previously connected through the <code>bdplyr()</code> function. Typically, it will be called after performing the desired operations with the <code>{dplyr}</code> verbs.
<code>billing_project_id</code>	a string containing your billing project id. If you've run <code>set_billing_id()</code> then feel free to leave this empty.
<code>show_query</code>	If TRUE will show the SQL query calling <code>dplyr::show_query()</code> . Is useful for diagnosing performance problems.

Value

A tibble.

Examples

```
## Not run:

# setup billing
basedosdados::set_billing_id("billing-project-id")

# select a cool database at Base dos Dados
bd_table <- basedosdados::bdplyr(
  "basedosdados.br_sp_gov_ssp.ocorrencias_registradas")

# quick look
bd_table %>%
  dplyr::glimpse()

# filter, select and group the remote data
bd_ssp <- bd_table %>%
  dplyr::filter(ano >= 2019) %>%
  dplyr::select(ano, mes, homicidio_doloso) %>%
  dplyr::group_by(ano, mes)

# make some plots
library(ggplot2)

bd_ssp %>%
  # collect the data to continue the analisis
  basedosdados::bd_collect() %>%
  dplyr::summarise(homicidios_sum = sum(homicidio_doloso,
                                       na.rm = TRUE)) %>%
  ggplot(aes(x = mes, y = homicidios_sum, fill = ano)) +
  geom_col(position = "dodge")

## End(Not run)
```

bd_write

Writes the result of operations with `bdplyr()` to disk

Description

Writes a remote table to disk that was called via `bdplyr`. It will collect the data and write to disk in the chosen format. You will only need this function if you have not yet collected the data using the `bd_collect()`.

The comprehensive function `bd_write()` takes as a parameter `.write_fn`, which will be the name of some function (without parentheses) capable of writing a tibble to disk.

As helpers, the `bd_write_rds()` and `bd_write_csv()` functions make it easier to write in these formats, more common in everyday life, calling writing functions from `{readr}` package.

Usage

```
bd_write(
  .lazy_tbl,
  .write_fn = `?(typed::Function()),
  path = `?(typed::Character(length = 1)),
  overwrite = `?(FALSE, typed::Logical(1)),
  ...
)
```

```
bd_write_rds(.lazy_tbl, path, overwrite = FALSE, compress = "none", ...)
```

```
bd_write_csv(
  .lazy_tbl,
  path = `?(typed::Character(1)),
  overwrite = `?(FALSE, typed::Logical(1)),
  ...
)
```

Arguments

<code>.lazy_tbl</code>	A lazy tibble, typically the output of <code>bdplyr()</code> .
<code>.write_fn</code>	A function for writing the result of a tibble to disk. Do not use <code>()</code> after the function's name, the function <i>object</i> should be passed. Some functions the user might consider are: <code>writexl::write_xlsx</code> , <code>jsonlite::write_json</code> , <code>foreign::write_dta</code> , <code>arrow::write_feather</code> , etc.
<code>path</code>	String containing the path for the file to be created. The desired folders must already exist and the file should normally end with the corresponding extension.
<code>overwrite</code>	FALSE by default. Indicates whether the local file should be overwritten if it already exists. Use with care.
<code>...</code>	Parameters passed to the <code>.write_fn</code> function.
<code>compress</code>	For <code>bd_write_rds()</code> only. Compression method to use: "none" (default), "gz", "bz", or "xz", in ascending order of compression. Remember that the higher the compression, the smaller the file size on disk, ut also the longer the time to load the data. See also: <code>readr::write_rds()</code> .

Value

String containing the path to the created file.

Examples

```
## Not run:

cool_db <- basedosdados::

# setup billing
basedosdados::set_billing_id("MY-BILLING-ID")
```

```
# connect with a Base dos Dados db

cool_db_ssp <- basedosdados::bdplyr(
  "basedosdados.br_sp_gov_ssp.ocorrencias_registradas")

# subset the data
my_subset <- cool_db_ssp %>%
  dplyr::filter(ano == 2021, mes == 04)

# write it in csv - generic function

basedosdados::bd_write(.lazy_tbl = my_subset,
  .write_fn = write.csv,
  "data-raw/ssp_subset.csv"
)

# write in .xlsx
basedosdados::bd_write(.lazy_tbl = my_subset,
  .write_fn = writexl::write_xlsx,
  "data-raw/ssp_subset.xlsx"
)

# using the derivatives functions
# to csv
basedosdados::bd_write_csv(.lazy_tbl = my_subset,
  "data-raw/ssp_subset2.csv"
)

#' # to rds
basedosdados::bd_write_rds(.lazy_tbl = my_subset,
  "data-raw/ssp_subset.rds"
)

# to rds - with compression
basedosdados::bd_write_rds(.lazy_tbl = my_subset,
  "data-raw/ssp_subset2.rds",
  compress = "gz"
)

# to rds - with HARD compression
basedosdados::bd_write_rds(.lazy_tbl = my_subset,
  "data-raw/ssp_subset3.rds",
  compress = "xz"
)

## using other write functions

# json
basedosdados::bd_write(.lazy_tbl = my_subset,
  .write_fn = jsonlite::write_json,
  "data-raw/ssp_subset.json"
)
```

```

# dta
basedosdados::bd_write(.lazy_tbl = my_subset,
                        .write_fn = foreign::write.dta,
                        "data-raw/ssp_subset.dta")
)

# feather
basedosdados::bd_write(.lazy_tbl = my_subset,
                        .write_fn = arrow::write_feather,
                        "data-raw/ssp_subset.feather"
)

## End(Not run)

```

download

Write the results of a query locally to a comma-separated file.

Description

Write the results of a query locally to a comma-separated file.

Usage

```

download(
  query = NULL,
  table = NULL,
  path,
  billing_project_id = get_billing_id(),
  .na = " "
)

```

Arguments

query	a string containing a valid SQL query.
table	defaults to NULL. If a table name is provided then it'll be concatenated with "basedosdados." and the whole table will be returned.
path	String with the output file's name. If running an R Project relative location can be provided. Passed to <code>readr::write_csv</code> 's file argument.
billing_project_id	a string containing your billing project id. If you've run <code>set_billing_id</code> then feel free to leave this empty.
.na	how should missing values be written in the resulting file? Value passed to <code>na</code> argument of <code>readr::write_csv</code> . Defaults to a whitespace.

Details

Currently there's only support for UTF-8 encoding. Users requiring more control over writing should use `read_sql` to get the data in memory and custom code from there.

Value

Invisibly returns the query's output in a tibble. Intended to be used for side-effects. If you simply want to load a query's result in memory, use `read_sql`.

Examples

```
## Not run:

path <- file.path(tempdir(), "pib_per_capita.csv")

bare_query <- "SELECT *
FROM basedosdados.br_tse_eleicoes.bens_candidato
WHERE ano = 2020
AND sigla_uf = \'TO\'"

download(query = bare_query, path = path)

# or download the entire table
download(table = "br_tse_eleicoes.bens_candidato", path = path)

## End(Not run)
```

`get_billing_id`*Internal functions for project billing management*

Description

Retrieves the project's billing Id.

Usage

```
get_billing_id()
```

Value

a string with the project's billing id.

```
get_dataset_description
      Describe a dataset
```

Description

Describe a dataset

Usage

```
get_dataset_description(dataset_id = `?(typed::Character(1))
```

Arguments

dataset_id	a dataset name e.g. if addressing table "br_sp_alesp.deputado" then table_id is br_sp_alesp
------------	---

Value

A tibble describing the specified dataset

Examples

```
## Not run:
get_dataset_description("br_sp_alesp")
## End(Not run)
```

```
get_table_description Describe a table within a dataset
```

Description

Describe a table within a dataset

Usage

```
get_table_description(
  dataset_id = `?(typed::Character(1)),
  table_id = `?(typed::Character(1))
)
```

Arguments

dataset_id a dataset name e.g. if addressing table "br_sp_alesp.deputado" then table_id is br_sp_alesp

table_id a table name e.g. if addressing table "br_sp_alesp.deputado" then table_id is deputado

Value

A tibble describing the specified table

Examples

```
## Not run:  
get_table_description("br_sp_alesp", "deputado")  
  
## End(Not run)
```

list_dataset_tables *List tables in a dataset*

Description

List tables in a dataset

Usage

```
list_dataset_tables(dataset_id)
```

Arguments

dataset_id a dataset name e.g. if addressing table "br_sp_alesp.deputado" then table_id is br_sp_alesp

Value

A tibble listing all tables in a given dataset

Examples

```
## Not run:  
list_dataset_tables("br_sp_alesp")  
  
## End(Not run)
```

partition_table	<i>Slice a big data frame into smaller csv files by grouping variables Still in development</i>
-----------------	---

Description

partition_table populates a folder

Usage

```
partition_table(.data, dir, ...)
```

Arguments

.data	a tibble.
dir	directory where to write the csv files. Must exist before function call.
...	comma-separated variables used to define groupings.

Value

invisibly returns all written files' addresses.

Examples

```
## Not run:  
  
tibble(  
  x = rnorm(1000),  
  y = runif(1000) + x,  
  group = sample(letters, 1000, replace = TRUE)) %>%  
  partition_table(tempdir())  
  
## End(Not run)
```

read_sql	<i>Query our datalake and get results in a tibble</i>
----------	---

Description

read_sql is given either a fully-written SQL query through the query argument or a valid table name through the table argument.

Usage

```
read_sql(query, billing_project_id = get_billing_id())
```

Arguments

query a string containing a valid SQL query.
billing_project_id a string containing your billing project id. If you've run set_billing_id then feel free to leave this empty.

Value

A tibble containing the query's output.

Examples

```
## Not run:  
  
set_billing_id("<your id here>")  
  
query <- "SELECT  
pib.id_municipio,  
pop.ano,  
pib.PIB / pop.populacao * 1000 as pib_per_capita  
FROM `basedosdados.br_ibge_pib.municipio` as pib  
JOIN `basedosdados.br_ibge_populacao.municipio` as pop  
ON pib.id_municipio = pop.id_municipio  
LIMIT 5 "  
  
data <- read_sql(query)  
  
# in case you want to write your data on disk as a .xlsx, .csv or .Rds file.  
  
library(writexl)  
library(readr)  
  
dir <- tempdir()  
  
write_xlsx(data, file.path(dir, "data.xlsx"))
```

```
write_csv(data, file.path(dir, "data.csv"))
saveRDS(data, file.path(dir, "data.Rds"))

## End(Not run)
```

set_billing_id *Define your Project Id*

Description

Define your project billing ids here so all your queries are authenticated and return data, not errors. If using in production or leaving code available at public repositories, dotenv is highly recommended.

Usage

```
set_billing_id(billing_project_id = NULL)
```

Arguments

billing_project_id
a single character value containing the string. Vectors with longer lengths and non-vectors will trigger an error.

Value

No return.

Examples

```
## Not run:
set_billing_id("my_billing_project_id")

# or load from an .env file

library(dotenv)

load_dot_env("keys.env")
print(Sys.getenv("billing_project_id"))

set_billing_id(Sys.getenv("billing_project_id"))

## End(Not run)
```

Index

`arrow::write_feather`, 6

`bd_collect`, 4
`bd_collect()`, 2, 3, 5
`bd_write`, 5
`bd_write()`, 2–5
`bd_write_csv` (`bd_write`), 5
`bd_write_csv()`, 2, 4, 5
`bd_write_rds` (`bd_write`), 5
`bd_write_rds()`, 2–6
`bdplyr`, 2
`bdplyr()`, 2, 4–6
`bigquery::src_bigquery`, 2, 3

`download`, 8
`dplyr::dplyr-package`, 2
`dplyr::filter()`, 2
`dplyr::left_join()`, 2
`dplyr::mutate()`, 2
`dplyr::select()`, 2
`dplyr::show_query()`, 4

`foreign::write.dta`, 6

`get_billing_id`, 9
`get_dataset_description`, 10
`get_table_description`, 10

`jsonlite::write_json`, 6

`list_dataset_tables`, 11

`partition_table`, 12

`read_sql`, 13
`readr::write_rds()`, 6

`set_billing_id`, 14
`set_billing_id()`, 2, 4

`writexl::write_xlsx`, 6