Package ‘timeseriesdb’

October 14, 2022

Type Package

Version 1.0.0-1.1.2

Title A Time Series Database for Official Statistics with R and PostgreSQL

Description Archive and manage times series data from official statistics. The 'timeseriesdb' package was designed to manage a large catalog of time series from official statistics which are typically published on a monthly, quarterly or yearly basis. Thus timeseriesdb is optimized to handle updates caused by data revision as well as elaborate, multi-lingual meta information.

Depends R (>= 3.0.0),

Imports RPostgres (>= 1.2.0), jsonlite (>= 1.1), data.table (>= 1.9.4), utils, xts, DBI,

Suggests openxlsx, rstudioapi, dygraphs, rmarkdown, knitr

VignetteBuilder knitr

Date 2022-03-23

License GPL-3

URL https://github.com/mbannert/timeseriesdb

BugReports https://github.com/mbannert/timeseriesdb/issues

LazyData true

RoxygenNote 7.1.2

Encoding UTF-8

Config/testthat/edition 3

NeedsCompilation no

Author Matthias Bannert [aut, cre],
Severin Thöni [aut],
Ioan Gabriel Bucur [ctb]

Maintainer Matthias Bannert <bannert@kof.ethz.ch>

Repository CRAN

Date/Publication 2022-03-23 22:20:02 UTC

R topics documented:

`as.meta` *Convert a List into a Metadata Object*

Description

Create timeseriesdb specific metadata class. Typically one list per natural language is converted to a meta description object.

Usage

```
as.meta(x)
```

Arguments

`x`	list of meta data.

as.tsmeta *Convert a List into a Metadata Object*

Description

Create timeseriesdb specific metadata class. Typically one list per natural language is converted to a meta description object.

Usage

```
as.tsmeta(meta, ...)
```

Arguments

`meta`	list containing meta information. List elements are character strings.
`...`	additional arguments, passed on to metthods below.

change_access_level *Change the Access Level of a Time Series*

Description

Change the Access Level of a Time Series

Usage

```
db_ts_change_access(
  con,
  ts_keys,
  access_level,
  valid_from = NULL,
  schema = "timeseries"
)

db_dataset_change_access(
  con,
  dataset,
  access_level,
  valid_from = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`access_level`	**character** describing the access level of the time series or dataset.
`valid_from`	character representation of a date in the form of 'YYYY-MM-DD'. valid_from starts a new version
`schema`	**character** name of the database schema. Defaults to 'timeseries'
`dataset`	**character** name of the dataset. Datasets are group of time series.

Value

returns a list containing the parsed JSON status feedback from the DB.

returns a list containing the parsed JSON status feedback from the DB.

See Also

Other access levels functions: `db_access_level_create()`, `db_access_level_delete()`, `db_access_level_list()`, `db_access_level_set_default()`, `db_ts_find_keys()`

`create_meta` *Create Meta Data Objects*

Description

Create list based S3 objects to store meta data. Meta data objects can be passed on to a plethora of functions including storing to database.

Usage

```
create_meta(...)
```

Arguments

`...`	arguments passed on the respective method.

`create_tsmeta` *Meta in*

Description

Meta in

Usage

```
create_tsmeta(...)
```

Arguments

`...`	arguments passed on the respective method.

`date_to_index` *Convert date-likes to time index*

Description

Convert date-likes to time index

Usage

```
date_to_index(x)
```

Arguments

`x`	The Date or Y-m-d string to convert

Value

The numeric representation of the date that can be used with ts

Examples

```
## Not run: date_to_index("2020-07-01")
```

`db_access_level_create` *Create a New Role (Access Level)*

Description

Creates a new role in the database. Roles represent access levels and together with the assignment of roles to time series, versions of time series or datasets define who is allowed to access a particular series.

Usage

```
db_access_level_create(
  con,
  access_level_name,
  access_level_description = NULL,
  access_level_default = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`access_level_name`	**character** name of the access level to insert.
`access_level_description`	**character** description of the access level. Defaults to NA.
`access_level_default`	set if the new access level should be the default. Defaults to NA.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

returns a list containing the parsed JSON status feedback from the DB.

See Also

Other access levels functions: `change_access_level`, `db_access_level_delete()`, `db_access_level_list()`, `db_access_level_set_default()`, `db_ts_find_keys()`

`db_access_level_delete` *Delete a role in access levels table*

Description

Delete a role in access levels table

Usage

```
db_access_level_delete(con, access_level, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`access_level`	**character** describing the access level of the time series or dataset.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

returns a list containing the parsed JSON status feedback from the DB.

See Also

Other access levels functions: `change_access_level`, `db_access_level_create()`, `db_access_level_list()`, `db_access_level_set_default()`, `db_ts_find_keys()`

`db_access_level_list` *Get All Access Levels and Their Description*

Description

Gets an overview of roles and shows whether a role (aka access level) is the default level for series stored without an explicitly specified access level.

Usage

```
db_access_level_list(con, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

access levels data.frame with columns ‘role‘ and ‘description‘ and ‘is_default‘

See Also

Other access levels functions: `change_access_level`, `db_access_level_create()`, `db_access_level_delete()`, `db_access_level_set_default()`, `db_ts_find_keys()`

`db_access_level_set_default`

Set the Default Access Level

Description

Changes the default access level. Apparently only one access level can be the default level at a time.

Usage

```
db_access_level_set_default(con, access_level, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`access_level`	**character** describing the access level of the time series or dataset.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

returns a list containing the parsed JSON status feedback from the DB.

See Also

Other access levels functions: `change_access_level`, `db_access_level_create()`, `db_access_level_delete()`, `db_access_level_list()`, `db_ts_find_keys()`

`db_call_function` *Helper to construct SQL function calls*

Description

Calls function ‘schema‘.‘fname‘ with the given ‘args‘, returning the result.

Usage

```
db_call_function(con, fname, args = NULL, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`fname`	character Name of the function to be called
`args`	list of function arguments. A single, unnested list.
`schema`	**character** name of the database schema. Defaults to ’timeseries’

Details

Args may be named to enable postgres to decide which candidate to choose in case of overloaded functions. If any args are named, all of them must be.

Value

value of ‘dbGetQuery(con, "SELECT * FROM schema.fname($args)")$fname‘

`db_collection_add_ts` *Bundles Keys into an Existing Collection or Adds a New Collection*

Description

Collections are user specific compilations of time series keys. Similar to a playlist in a music app, collections help to come back to a previously stored selection of time series. This functions adds more time series to existing bundles (collections).

Usage

```
db_collection_add_ts(
  con,
  collection_name,
  ts_keys,
  description = NULL,
  user = Sys.info()["user"],
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`ts_keys`	**character** vector of time series identifiers.
`description`	**character** description of the collection.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other collections functions: `db_collection_delete()`, `db_collection_get_keys()`, `db_collection_get_last_update`
`db_collection_list()`, `db_collection_remove_ts()`

Examples

```
## Not run:
db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_store(con = connection, kof_ts, schema = "schema")

db_collection_add_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.departure.total",
    "ch.kof.barometer"
  ),
  schema = "schema"
)

## End(Not run)
```

`db_collection_delete` *Remove an Entire Time Series Key Collection*

Description

Remove an Entire Time Series Key Collection

Usage

```
db_collection_delete(
  con,
  collection_name,
  user = Sys.info()["user"],
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other collections functions: `db_collection_add_ts()`, `db_collection_get_keys()`, `db_collection_get_last_update`
`db_collection_list()`, `db_collection_remove_ts()`

Examples

```
## Not run:
db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_store(con = connection, kof_ts, schema = "schema")

db_collection_add_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.departure.total",
    "ch.kof.barometer"
  ),
  schema = "schema"
)

db_collection_delete(
  con = connection,
  collection_name = "barometer and departures zurich",
  schema = "schema"
)

## End(Not run)
```

db_collection_get_keys

Get All Keys in a User Collection

Description

Reads all keys in the given collection and returns them in a vector

Usage

```
db_collection_get_keys(
  con,
  collection_name,
  user = Sys.info()["user"],
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other collections functions: `db_collection_add_ts()`, `db_collection_delete()`, `db_collection_get_last_update()` `db_collection_list()`, `db_collection_remove_ts()`

db_collection_get_last_update

Get the last update of a collection for a specific User

Description

Get the last update of a collection for a specific User

Usage

```
db_collection_get_last_update(
  con,
  collection_name,
  user = Sys.info()["user"],
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other collections functions: `db_collection_add_ts()`, `db_collection_delete()`, `db_collection_get_keys()`, `db_collection_list()`, `db_collection_remove_ts()`

Examples

```
## Not run:

db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_store(con = connection, kof_ts, schema = "schema")

db_collection_add_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.departure.total",
    "ch.kof.barometer"
  ),
  schema = "schema"
)

db_collection_get_last_update(
  con = connection,
  collection_name = "barometer and departures zurich",
  schema = "schema"
)

## End(Not run)
```

`db_collection_list` *List All Available Collections for a Specific User*

Description

List All Available Collections for a Specific User

Usage

```
db_collection_list(con, user = Sys.info()["user"], schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other collections functions: `db_collection_add_ts()`, `db_collection_delete()`, `db_collection_get_keys()`, `db_collection_get_last_update()`, `db_collection_remove_ts()`

Examples

```
## Not run:
ts1 <- list(ts(rnorm(100), start = c(1990, 1), frequency = 4))
names(ts1) <- c("ts1")
db_ts_store(con = connection, ts1, schema = "schema")
db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_store(con = connection, kof_ts, schema = "schema")

db_collection_add_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.departure.total",
    "ch.kof.barometer"
  ),
  schema = "schema"
)

db_collection_add_ts(
  con = connection,
  collection_name = "ts1 and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ts1"
  ),
  schema = "schema"
)

db_collection_list(
  con = connection,
  schema = "schema"
)

## End(Not run)
```

`db_collection_read_metadata`
Read Metadata for a Collection

Description

Read Metadata for a Collection

Usage

```
db_collection_read_metadata(
  con,
  collection_name,
  collection_owner,
  valid_on = NULL,
  locale = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	character name of the collection.
`collection_owner`	character name of the collection owner.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`locale`	**character** indicating the language of the meta information to be store. We recommend to use ISO country codes to represent languages. Defaults to NULL. When local is set to NULL, metadata are stored without localization. Note that, when localizing meta information by assigning a language, multiple meta information objects can be stored for a single time series.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

list of all available meta descriptions for a particular collection and language.

See Also

Other metadata functions: `db_dataset_read_metadata()`, `db_meta_get_latest_validity()`, `db_metadata_read()`, `db_metadata_store()`

`db_collection_read_ts` *Read all Time Series in a User Collection*

Description

Read all Time Series in a User Collection

Usage

```
db_collection_read_ts(
  con,
  collection_name,
  collection_owner,
  valid_on = NULL,
  respect_release_date = FALSE,
  schema = "timeseries",
  chunksize = 10000
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`collection_owner`	**character** username that is the owner of a collection.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`respect_release_date`	**boolean** indicating if it should the release embargo of a time series be respected. Defaults to FALSE. This option makes sense when the function is used in an API. In that sense, users do not have direct access to this function and therefore cannot simply switch parameters.
`schema`	**character** name of the database schema. Defaults to 'timeseries'
`chunksize`	set a limit of the number of time series requested in the function.

Details

Collections are identified by their name and owner. Several collections with the same name but different owners may exist, therefore both need to be supplied in order to uniquely identify a collection.

See Also

Other time series functions: `db_dataset_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_delete()`, `db_ts_get_last_update()`, `db_ts_read_history()`, `db_ts_read()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:

db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_store(con = connection, kof_ts, schema = "schema")

db_collection_add_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.departure.total",
    "ch.kof.barometer"
  ),
  schema = "schema"
)

db_collection_read_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  collection_owner = "user_name",
  schema = "schema"
)

## End(Not run)
```

`db_collection_remove_ts` *Remove Keys From a User's Collection*

Description

Removes a vector of time series keys from a user specific compilation.

Usage

```
db_collection_remove_ts(
  con,
  collection_name,
  ts_keys,
  user = Sys.info()["user"],
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`ts_keys`	**character** vector of time series identifiers.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other collections functions: `db_collection_add_ts()`, `db_collection_delete()`, `db_collection_get_keys()`, `db_collection_get_last_update()`, `db_collection_list()`

Examples

```

## Not run:
db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_store(con = connection, kof_ts, schema = "schema")

db_collection_add_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.departure.total",
    "ch.kof.barometer"
  ),
  schema = "schema"
)

db_collection_remove_ts(
  con = connection,
  collection_name = "barometer and departures zurich",
  ts_keys = "ch.zrh_airport.departure.total",
  schema = "schema"
)

## End(Not run)
```

`db_connection_close` *Close an Existing Database Connection*

Description

Close database connection given a connection object.

Usage

```
db_connection_close(con, ...)
```

Arguments

`con`	RPostgres connection object.
`...`	passed on to RPostgres::dbDisconnect

`db_connection_create` *Create Database Connection*

Description

Connects to the PostgreSQL database backend of timeseriesdb. This function is convenience wrapper around DBI's dbConnect. It's less general than the DBI function and only works for PostgreSQL, but it is a little more convenient because of its defaults / assumptions.

Usage

```
db_connection_create(
  dbname,
  user = Sys.info()[["user"]],
  host = "localhost",
  passwd = NULL,
  passwd_from_file = FALSE,
  line_no = 1,
  passwd_from_env = FALSE,
  connection_description = "timeseriesdb",
  port = 5432
)
```

Arguments

`dbname`	character name of the database.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.
`host`	character denoting the hostname. Defaults to localhost.
`passwd`	character password, file or environment name. Defaults to NULL triggering an R Studio function that asks for your passwords interactively if you are on R Studio. Make sure to adapt the boolean params correspondingly.
`passwd_from_file`	boolean if set to TRUE the passwd param is interpreted as a file location for a password file such as .pgpass. Make sure to be very restrictive with file permissions if you store a password to a file.
`line_no`	integer specify line number of password file that holds the actual password.
`passwd_from_env`	boolean if set to TRUE the passwd param is interpreted as the name of an environment variable from which to get the password
`connection_description`	character connection description describing the application that connects to the database. This is mainly helpful for DB admins and shows up in the pg_stat_activity table. Defaults to 'timeseriesdb'. Avoid spaces as this is a psql option.
`port`	integer defaults to 5432, the PostgreSQL standard port.

`db_dataset_create` *Create a New Dataset*

Description

A dataset is a family of time series that belong to the same topic. By default all series stored with 'db_store_ts' belong to a default set. In order to assign them a different set, it must first be created with 'db_dataset_create' after which the series may be moved with `db_ts_assign_dataset`.

Usage

```
db_dataset_create(
  con,
  set_name,
  set_description = NULL,
  set_md = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`set_name`	**character** name of a dataset.
`set_description`	**character** description about the set. Default to NA.
`set_md`	meta information data about the set. Default to NA.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

character name of the created set

See Also

Other datasets functions: `db_dataset_delete()`, `db_dataset_get_keys()`, `db_dataset_get_last_update()`, `db_dataset_list()`, `db_dataset_trim_history()`, `db_dataset_update_metadata()`, `db_ts_assign_dataset()`, `db_ts_get_dataset()`

Examples

```
## Not run:

db_dataset_create(
  con = connection,
  set_name = "zrh_airport_data",
  set_description = "Zurich airport arrivals and departures ",
  schema = "schema"
)

## End(Not run)
```

`db_dataset_delete` *Irrevocably Delete All Time Series in a Set and the Set Itself*

Description

This function cannot be used in batch mode as it needs user interaction. It asks the user to manually input confirmation to prevent unintentional deletion of datasets.

Usage

```
db_dataset_delete(con, set_name, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`set_name`	**character** name of a dataset.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

character name of the deleted set, NA in case of an error.

See Also

Other datasets functions: `db_dataset_create()`, `db_dataset_get_keys()`, `db_dataset_get_last_update()`, `db_dataset_list()`, `db_dataset_trim_history()`, `db_dataset_update_metadata()`, `db_ts_assign_dataset()`, `db_ts_get_dataset()`

Examples

```
## Not run:

db_dataset_create(
  con = connection,
  set_name = "zrh_airport_data",
  set_description = "Zurich airport arrivals and departures ",
  schema = "schema"
)

db_dataset_delete(
  con = connection,
  set_name = "zrh_airport_data",
  schema = "schema"
)

## End(Not run)
```

`db_dataset_get_keys` *Get All Time Series Keys in a Given Set*

Description

Get All Time Series Keys in a Given Set

Usage

```
db_dataset_get_keys(con, set_name = "default", schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`set_name`	**character** name of a dataset.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

character A vector of ts keys contained in the set

See Also

Other datasets functions: `db_dataset_create()`, `db_dataset_delete()`, `db_dataset_get_last_update()`, `db_dataset_list()`, `db_dataset_trim_history()`, `db_dataset_update_metadata()`, `db_ts_assign_dataset()`, `db_ts_get_dataset()`

Examples

```
## Not run:

db_dataset_get_keys(
  con = connection,
  set_name = "zrh_airport_data",
  set_description = "Zurich airport arrivals and departures ",
  schema = "schema"
)

## End(Not run)
```

`db_dataset_get_last_update` *Get the dataset last update*

Description

Get the dataset last update

Usage

```
db_dataset_get_last_update(con, set_id, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`set_id`	**character** name of the set to get the last update
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other datasets functions: db_dataset_create(), db_dataset_delete(), db_dataset_get_keys(), db_dataset_list(), db_dataset_trim_history(), db_dataset_update_metadata(), db_ts_assign_dataset(), db_ts_get_dataset()

Examples

```
## Not run:

# Storing different versions of the data, use parameter valid_from
# different versions are stored with the same key
ch.kof.barometer <- kof_ts["baro_2019m11"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2019-12-01",
  schema = "schema"
)

ch.kof.barometer <- kof_ts["baro_2019m12"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2020-01-01",
  schema = "schema"
)

db_dataset_create(
  con = connection,
  set_name = "barometer",
  set_description = "KOF Barometer",
  schema = "schema"
)

db_ts_assign_dataset(
  con = connection,
  ts_keys = "ch.kof.barometer",
  set_name = "barometer",
  schema = "schema"
)

db_dataset_get_last_update(
  con = connection,
  set_id = "barometer",
  schema = "schema"
)

## End(Not run)
```

db_dataset_get_latest_release

Get the latest Release for Given Datasets

Description

Get the latest Release for Given Datasets

Usage

```
db_dataset_get_latest_release(con, set_ids, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`set_ids`	Sets to get release dates for
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

data.frame with columns ‘set_id‘, ‘release_id‘, ‘release_date‘

See Also

Other calendar functions: `db_dataset_get_next_release()`, `db_dataset_get_release()`, `db_release_cancel()`, `db_release_create()`, `db_release_list()`, `db_release_update()`

db_dataset_get_next_release

Get Next Release Date for Given Datasets

Description

Get Next Release Date for Given Datasets

Usage

```
db_dataset_get_next_release(con, set_ids, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`set_ids`	Sets to get release dates for
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

data.frame with columns ‘set_id‘, ‘release_id‘, ‘release_date‘

See Also

Other calendar functions: `db_dataset_get_latest_release()`, `db_dataset_get_release()`, `db_release_cancel()`, `db_release_create()`, `db_release_list()`, `db_release_update()`

---

db_dataset_get_release

Get the latest Release for Given Datasets

Description

Get the latest Release for Given Datasets

Usage

```
db_dataset_get_release(
  con,
  set_ids,
  target_year = year(Sys.Date()),
  target_period = month(Sys.Date()),
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`set_ids`	Sets to get release dates for
`target_year`	Year of the desired release
`target_period`	Period of the desired release
`schema`	**character** name of the database schema. Defaults to ’timeseries’

Value

data.frame with columns ‘set_id‘, ‘release_id‘, ‘release_date‘

See Also

Other calendar functions: `db_dataset_get_latest_release()`, `db_dataset_get_next_release()`, `db_release_cancel()`, `db_release_create()`, `db_release_list()`, `db_release_update()`

db_dataset_list *Get All Available Datasets and Their Description*

Description

Get All Available Datasets and Their Description

Usage

```
db_dataset_list(con, schema = "timeseries")
```

Arguments

con	RPostgres connection object.
schema	**character** name of the database schema. Defaults to 'timeseries'

Value

data.frame with columns 'set_id' and 'set_description'

See Also

Other datasets functions: db_dataset_create(), db_dataset_delete(), db_dataset_get_keys(), db_dataset_get_last_update(), db_dataset_trim_history(), db_dataset_update_metadata(), db_ts_assign_dataset(), db_ts_get_dataset()

Examples

```
## Not run:

db_dataset_create(
  con = connection,
  set_name = "zrh_airport_data",
  set_description = "Zurich airport arrivals and departures ",
  schema = "schema"
)

db_dataset_list(
  con = connection,
  schema = "schema"
)

## End(Not run)
```

`db_dataset_read_metadata` *Read Dataset Meta Information*

Description

Read Dataset Meta Information

Usage

```
db_dataset_read_metadata(
  con,
  dataset_id,
  valid_on = NULL,
  locale = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`dataset_id`	character name of the dataset.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`locale`	character ISO-2 country locale.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other metadata functions: `db_collection_read_metadata()`, `db_meta_get_latest_validity()`, `db_metadata_read()`, `db_metadata_store()`

`db_dataset_read_ts` *Read all Time Series in a Dataset*

Description

Read all Time Series in a Dataset

Usage

```
db_dataset_read_ts(
  con,
  datasets,
  valid_on = NULL,
  respect_release_date = FALSE,
  schema = "timeseries",
  chunksize = 10000
)
```

Arguments

con	RPostgres connection object.
datasets	**character** vector of the datasets. Dataset is a group of time series.
valid_on	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
respect_release_date	**boolean** indicating if it should the release embargo of a time series be respected. Defaults to FALSE. This option makes sense when the function is used in an API. In that sense, users do not have direct access to this function and therefore cannot simply switch parameters.
schema	**character** name of the database schema. Defaults to 'timeseries'
chunksize	set a limit of the number of time series requested in the function.

See Also

Other time series functions: `db_collection_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_delete()`, `db_ts_get_last_update()`, `db_ts_read_history()`, `db_ts_read()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:
db_dataset_create(con = connection,
                  set_name = "zrh_airport_data",
                  set_description = "Zurich airport arrivals and departures ",
                  schema = "schema")

db_ts_assign_dataset(con = connection,
                     ts_keys = c("ch.zrh_airport.departure.total",
                                 "ch.zrh_airport.arrival.total"),
                     set_name = "zrh_airport_data",
                     schema = "schema")

db_dataset_read_ts(con = connection,
                   datasets = "zrh_airport_data",
                   schema = "schema")

## End(Not run)
```

db_dataset_trim_history

Remove Vintages from the Beginning of Dataset

Description

Removes any vintages of the given dataset that are older than a specified date.

Usage

```
db_dataset_trim_history(con, set_id, older_than, schema = "timeseries")
```

Arguments

con	RPostgres connection object.
set_id	character Name of the set to trim
older_than	Date cut off point
schema	**character** name of the database schema. Defaults to 'timeseries'

Details

In some cases only the last few versions of time series are of interest. This function can be used to trim off old vintages that are no longer relevant. It may be helpful to use this function with high frequency data to save disk space of versions are not needed.

See Also

Other datasets functions: `db_dataset_create()`, `db_dataset_delete()`, `db_dataset_get_keys()`, `db_dataset_get_last_update()`, `db_dataset_list()`, `db_dataset_update_metadata()`, `db_ts_assign_dataset()`, `db_ts_get_dataset()`

Examples

```
## Not run:

# Storing different versions of the data, use parameter valid_from
# different versions are stored with the same key
ch.kof.barometer <- kof_ts["baro_2019m11"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2019-12-01",
  schema = "schema"
)

ch.kof.barometer <- kof_ts["baro_2019m12"]
```

```
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2020-01-01",
  schema = "schema"
)

db_dataset_create(
  con = connection,
  set_name = "barometer",
  set_description = "KOF Barometer",
  schema = "schema"
)

db_ts_assign_dataset(
  con = connection,
  ts_keys = "ch.kof.barometer",
  set_name = "barometer",
  schema = "schema"
)

db_dataset_trim_history(
  con = connection,
  set_id = "barometer",
  older_than = "2019-12-31",
  schema = "schema"
)

## End(Not run)
```

`db_dataset_update_metadata` *Update Description and/or Metadata of a Dataset*

Description

Update Description and/or Metadata of a Dataset

Usage

```
db_dataset_update_metadata(
  con,
  set_name,
  description = NULL,
  metadata = NULL,
  metadata_update_mode = "update",
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`set_name`	**character** name of a dataset.
`description`	character New description. If set to NA (default) the description is left untouched
`metadata`	**list** Metadata update (see metadata_update_mode)
`metadata_update_mode`	character one of "update" or "overwrite". If set to "update", new fields in the list are added to the existing metadata and existing fields overwritten. If NA nothing happens in update mode. If set to "overwrite" ALL existing metadata is replaced.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other datasets functions: `db_dataset_create()`, `db_dataset_delete()`, `db_dataset_get_keys()`, `db_dataset_get_last_update()`, `db_dataset_list()`, `db_dataset_trim_history()`, `db_ts_assign_dataset()`, `db_ts_get_dataset()`

Examples

```
## Not run:

db_dataset_update_metadata(
  con = connection,
  set_name = "zrh_airport_data",
  description = "updating description Zurich airport arrivals and departures",
  schema = "schema"
)

## End(Not run)
```

`db_get_installed_version` *Get the Currently Installed Version of Timeseriesdb*

Description

Get the Currently Installed Version of Timeseriesdb

Usage

```
db_get_installed_version(con, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

character The version number of timeseriesdb currently installed on the given schema

`db_grant_to_admin` *GRANT all rights on a (temp) table to schema admin*

Description

The SECURITY DEFINER functions do not have access to tables that are stored via dbWriteTable. Usage rights on these tables must be granted for them to be usable inside the db functions

Usage

```
db_grant_to_admin(con, table, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`table`	which table to grant rights on
`schema`	**character** name of the database schema. Defaults to 'timeseries'

`db_metadata_read` *Read Time Series Metadata*

Description

Read meta information given a vector of time series identifiers.

Usage

```
db_metadata_read(
  con,
  ts_keys,
  valid_on = NULL,
  regex = FALSE,
  locale = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`regex`	**boolean** indicating if ts_keys should be interpreted as a regular expression pattern. Defaults to FALSE.
`locale`	**character** indicating the language of the meta information to be store. We recommend to use ISO country codes to represent languages. Defaults to NULL. When local is set to NULL, metadata are stored without localization. Note that, when localizing meta information by assigning a language, multiple meta information objects can be stored for a single time series.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

list of tsmeta objects.

See Also

Other metadata functions: `db_collection_read_metadata()`, `db_dataset_read_metadata()`, `db_meta_get_latest_validity()`, `db_metadata_store()`

`db_metadata_store` *Store Time Series Metadata to PostgreSQL*

Description

The most basic way to store meta information is to assign non-translated (unlocalized) descriptions, but it also can be stored in different languages (localized) using the parameter **locale**. See also basic usage.

Usage

```
db_metadata_store(
  con,
  metadata,
  valid_from,
  locale = NULL,
  on_conflict = "update",
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`metadata`	object of class tsmeta that contains the metadata to be stored.
`valid_from`	**character** representation of a date in the form of 'YYYY-MM-DD'. It should always be explicitly specified.
`locale`	**character** indicating the language of the meta information to be store. We recommend to use ISO country codes to represent languages. Defaults to NULL. When local is set to NULL, metadata are stored without localization. Note that, when localizing meta information by assigning a language, multiple meta information objects can be stored for a single time series.
`on_conflict`	**character** either "update": add new fields and update existing ones or "overwrite": completely replace existing record.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

status list created from DB status return JSON.

See Also

Other metadata functions: `db_collection_read_metadata()`, `db_dataset_read_metadata()`, `db_meta_get_latest_validity()`, `db_metadata_read()`

Examples

```
## Not run:
sum("a")

## End(Not run)
```

`db_meta_get_latest_validity`
Get Latest Validity for Metadata of a Given Time Series

Description

Because metadata are only loosely coupled with their respective time series in order to keep metadata records constant over multiple version of time series if the data description does not change, it comes in handy to find out the last time meta information was updated. This function automagickally finds exactly this date.

Usage

```
db_meta_get_latest_validity(
  con,
  ts_keys,
  regex = FALSE,
  locale = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`regex`	**boolean** indicating if ts_keys should be interpreted as a regular expression pattern. Defaults to FALSE.
`locale`	**character** indicating the language of the meta information to be store. We recommend to use ISO country codes to represent languages. Defaults to NULL. When local is set to NULL, metadata are stored without localization. Note that, when localizing meta information by assigning a language, multiple meta information objects can be stored for a single time series.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

data.table of latest validity

See Also

Other metadata functions: `db_collection_read_metadata()`, `db_dataset_read_metadata()`, `db_metadata_read()`, `db_metadata_store()`

`db_release_cancel` *Cancel a Scheduled Release*

Description

Attempts to cancel a release that has already passed will result in an error.

Usage

```
db_release_cancel(con, release_id, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`release_id`	character ID of the release to cancel
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other calendar functions: `db_dataset_get_latest_release()`, `db_dataset_get_next_release()`, `db_dataset_get_release()`, `db_release_create()`, `db_release_list()`, `db_release_update()`

`db_release_create` *Create an Entry in the Release Calendar*

Description

The idea of the release calendar is to set a release date for some time series that might be in the database already but should not be publicly available before a specific date, e.g., a press release. Since publishing is simply a matter of changing the access level, an update of the access levels could be triggered based on the release information in a release table. Only timeseries admins may create and modify releases.

Usage

```
db_release_create(
  con,
  id,
  title,
  release_date,
  datasets,
  target_year = year(release_date),
  target_period = month(release_date),
  target_frequency = 12,
  note = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`id`	Identifier for the release e.g. 'gdb_may_2020'
`title`	Display title for the release
`release_date`	Timestamp when the release is to occur
`datasets`	**character** vector of the datasets. Dataset is a group of time series.
`target_year`	Year observed in the data
`target_period`	Period observed in the data (e.g. month, quarter)
`target_frequency`	Frequency of the data (e.g. 4 for quarterly)
`note`	Additional remarks about the release.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Details

`target_period` changes meaning depending on the frequency of the release. e.g. period 2 for quarterly data (reference_frequency = 4) means Q2 whereas period 2 for monthly data (frequency 12) means February In other words: `target_year` and `target_period` mark the end of the time series in the release.

Value

a status list

See Also

Other calendar functions: `db_dataset_get_latest_release()`, `db_dataset_get_next_release()`, `db_dataset_get_release()`, `db_release_cancel()`, `db_release_list()`, `db_release_update()`

`db_release_list` *List Data on Registered Releases*

Description

List Data on Registered Releases

Usage

```
db_release_list(con, include_past = FALSE, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`include_past`	Should past releases be included? Defaults to FALSE
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

data.frame with columns ‘id‘, ‘title‘, ‘note‘, ‘release_date‘, ‘reference_year‘, ‘reference_period‘, ‘reference_frequency‘

See Also

Other calendar functions: `db_dataset_get_latest_release()`, `db_dataset_get_next_release()`, `db_dataset_get_release()`, `db_release_cancel()`, `db_release_create()`, `db_release_update()`

`db_release_update` *Update an Existing Release Record*

Description

Any parameters provided to this function will overwrite the corresponding fields in the database. Parameters set to NA (default) will leave the corresponding fields untouched. For details see `db_release_create`.

Usage

```
db_release_update(
  con,
  id,
  title = NULL,
  release_date = NULL,
  datasets = NULL,
  target_year = NULL,
  target_period = NULL,
  target_frequency = NULL,
  note = NULL,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`id`	Identifier for the release e.g. 'gdb_may_2020'
`title`	Display title for the release
`release_date`	Timestamp when the release is to occur
`datasets`	**character** vector of the datasets. Dataset is a group of time series.
`target_year`	Year observed in the data
`target_period`	Period observed in the data (e.g. month, quarter)
`target_frequency`	Frequency of the data (e.g. 4 for quarterly)
`note`	Additional remarks about the release.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Value

a status list

See Also

Other calendar functions: `db_dataset_get_latest_release()`, `db_dataset_get_next_release()`, `db_dataset_get_release()`, `db_release_cancel()`, `db_release_create()`, `db_release_list()`

`db_ts_assign_dataset` *Assign Time Series Identifiers to a Dataset*

Description

'db_ts_assign_dataset' returns a list with status information. status '"ok"' means all went well. status '"warning"' means some keys are not in the catalog. The vector of those keys is in the 'offending_keys' field.

Usage

```
db_ts_assign_dataset(con, ts_keys, set_name, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`set_name`	**character** name of a dataset.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Details

Trying to assign keys to a non-existent dataset is an error.

Value

list A status list

See Also

Other datasets functions: `db_dataset_create()`, `db_dataset_delete()`, `db_dataset_get_keys()`, `db_dataset_get_last_update()`, `db_dataset_list()`, `db_dataset_trim_history()`, `db_dataset_update_metadata`
`db_ts_get_dataset()`

Examples

```
## Not run:

db_dataset_create(
  con = connection,
  set_name = "zrh_airport_data",
  set_description = "Zurich airport arrivals and departures ",
  schema = "schema"
)

db_ts_assign_dataset(
  con = connection,
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.arrival.total"
  ),
  set_name = "zrh_airport_data",
  schema = "schema"
)

## End(Not run)
```

---

db_ts_delete *Remove Time Series from the Database*

Description

This function completely removes a time series from the database, including all vintages and meta-data.

Usage

```
db_ts_delete(con, ts_keys, schema = "timeseries", skip_checks = FALSE)
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`schema`	**character** name of the database schema. Defaults to 'timeseries'
`skip_checks`	boolean should checks be skipped? Use with caution and only in batch mode! Defaults to FALSE.

Details

Due to the potentially severe consequences of such a deletion only timeseries admins may perform this action and should do so very diligently.

See Also

Other time series functions: `db_collection_read_ts()`, `db_dataset_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_get_last_update()`, `db_ts_read_history()`, `db_ts_read()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:
# Store zrh_airport data
db_ts_store(con = connection, zrh_airport, schema = "schema")

# Deleting one key
```

```
db_ts_delete(
  con = connection,
  ts_keys = "ch.zrh_airport.departure.total",
  schema = "schema"
)

# Deleting multiple keys
db_ts_delete(
  con = connection,
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.arrival.total"
  ),
  schema = "schema"
)

## End(Not run)
```

`db_ts_delete_latest_version` *Delete the Latest Vintage of a Time Series*

Description

Vintages of time series should not be deleted as they are versions and represent a former status of a time series that may not be stored elsewhere, even not with their original provider. To benchmark forecasts it is essential to keep the versions to evaluate real time performance of forecasts. However, when operating at current edge of a time series, i.e., its last update, mistakes may happen. Hence timeseriesdb allows to update / delete the last iteration. Do not loop recursively through iterations to delete an entire time series. There are admin level functions for that.

Usage

```
db_ts_delete_latest_version(con, ts_keys, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other time series functions: `db_collection_read_ts()`, `db_dataset_read_ts()`, `db_ts_delete()`, `db_ts_get_last_update()`, `db_ts_read_history()`, `db_ts_read()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:

# Store different versions of the time series data
ch.kof.barometer <- kof_ts["baro_2019m11"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2019-12-01",
  schema = "schema"
)

ch.kof.barometer <- kof_ts["baro_2019m12"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2020-01-01",
  schema = "schema"
)

db_ts_delete_latest_version(
  con = connection,
  ts_keys = "ch.kof.barometer",
  schema = "schema"
)

## End(Not run)
```

`db_ts_find_keys` *Get All keys that follow a pattern*

Description

Get All keys that follow a pattern

Usage

```
db_ts_find_keys(con, pattern, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`pattern`	**character** that represents a regular expression to find keys
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other access levels functions: `change_access_level`, `db_access_level_create()`, `db_access_level_delete()`, `db_access_level_list()`, `db_access_level_set_default()`

Examples

```
## Not run:
db_ts_store(con = connection, zrh_airport, schema = "schema")

# get all keys that start with "ch"
db_ts_find_keys(
  con = connection,
  "^ch",
  schema = "schema")

## End(Not run)
```

db_ts_get_access_level

Find Out About the Access Level of a Vintage

Description

Provide the function with vector of time series keys and find out which access level is necessary to access the supplied keys.

Usage

```
db_ts_get_access_level(con, ts_keys, valid_on = NULL, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

db_ts_get_dataset *Find Datasets Given a Set*

Description

Return set identifiers associated with a vector of keys. If a ts key does not exist in the catalog, set_id will be NA.

Usage

```
db_ts_get_dataset(con, ts_keys, schema = "timeseries")
```

Arguments

con	RPostgres connection object.
ts_keys	**character** vector of time series identifiers.
schema	**character** name of the database schema. Defaults to 'timeseries'

Value

data.frame with columns 'ts_key' and 'set_id'

See Also

Other datasets functions: db_dataset_create(), db_dataset_delete(), db_dataset_get_keys(), db_dataset_get_last_update(), db_dataset_list(), db_dataset_trim_history(), db_dataset_update_metadata(db_ts_assign_dataset()

Examples

```
## Not run:

# one key
db_ts_get_dataset(
  con = connection,
  ts_keys = "ch.zrh_airport.departure.total",
  schema = "schema"
)

# multiple keys
db_ts_get_dataset(
  con = connection,
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.arrival.total"
  ),
  schema = "schema"
)

## End(Not run)
```

db_ts_get_last_update *Get the times series last update*

Description

Get the times series last update

Usage

```
db_ts_get_last_update(con, ts_keys, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other time series functions: `db_collection_read_ts()`, `db_dataset_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_delete()`, `db_ts_read_history()`, `db_ts_read()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:
db_ts_store(con = connection, zrh_airport, schema = "schema")

# get last update for one key
db_ts_get_last_update(
  con = connection,
  ts_keys = "ch.zrh_airport.departure.total",
  schema = "schema")

# get last update for multiple keys
db_ts_get_last_update(
  con = connection,
  ts_keys = c(
    "ch.zrh_airport.departure.total",
    "ch.zrh_airport.arrival.total"
  ),
  schema = "schema"
)

## End(Not run)
```

db_ts_read — *Read Time Series From PostgreSQL into R*

Description

Read specific version of a time series given time series key (unique identifier) and validity. By default, this function returns the most recent version of a time series.

Usage

```
db_ts_read(
  con,
  ts_keys,
  valid_on = NULL,
  regex = FALSE,
  respect_release_date = FALSE,
  schema = "timeseries",
  chunksize = 10000
)
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`regex`	**boolean** indicating if ts_keys should be interpreted as a regular expression pattern. Defaults to FALSE.
`respect_release_date`	**boolean** indicating if it should the release embargo of a time series be respected. Defaults to FALSE. This option makes sense when the function is used in an API. In that sense, users do not have direct access to this function and therefore cannot simply switch parameters.
`schema`	**character** name of the database schema. Defaults to 'timeseries'
`chunksize`	set a limit of the number of time series requested in the function.

Value

list of time series. List elements vary depending on nature of time series, i.e., regular vs. irregular time series.

See Also

Other time series functions: `db_collection_read_ts()`, `db_dataset_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_delete()`, `db_ts_get_last_update()`, `db_ts_read_history()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:
db_ts_store(con = connection, zrh_airport, schema = "schema")
db_ts_read(con = connection, ts_keys = "ch.zrh_airport.departure.total", schema = "schema")

## End(Not run)
```

`db_ts_read_history` *Read the Entire History of a Time Series*

Description

This function returns a list whose keys correspond to the date on which the contained version of the time series took effect.

Usage

```
db_ts_read_history(
  con,
  ts_key,
  respect_release_date = FALSE,
  schema = "timeseries"
)
```

Arguments

`con`	RPostgres connection object.
`ts_key`	character The identifier of the time series to read.
`respect_release_date`	**boolean** indicating if it should the release embargo of a time series be respected. Defaults to FALSE. This option makes sense when the function is used in an API. In that sense, users do not have direct access to this function and therefore cannot simply switch parameters.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other time series functions: `db_collection_read_ts()`, `db_dataset_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_delete()`, `db_ts_get_last_update()`, `db_ts_read()`, `db_ts_store()`, `db_ts_trim_history()`

Examples

```
## Not run:

# Storing different versions of the data, use parameter valid_from
# different versions are stored with the same key
ch.kof.barometer <- kof_ts["baro_2019m11"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(con = connection,
                  ch.kof.barometer,
                  valid_from = "2019-12-01",
                  schema = "schema")

ch.kof.barometer <- kof_ts["baro_2019m12"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(con = connection,
                  ch.kof.barometer,
                  valid_from = "2020-01-01",
                  schema = "schema")

# Reading all versions
db_ts_read_history(con = connection,
                         ts_key = "ch.kof.barometer",
                         schema = "schema")

## End(Not run)
```

`db_ts_rename` *Rename Time Series by Assigning a New Key*

Description

Rename Time Series by Assigning a New Key

Usage

```
db_ts_rename(con, ts_key, ts_key_new, schema = "timeseries")
```

Arguments

con	RPostgres connection object.
ts_key	character Vector of keys to rename
ts_key_new	character Vector of new names
schema	**character** name of the database schema. Defaults to 'timeseries'

db_ts_store *Store a Time Series to the Database*

Description

Stores one or more time series to the database.

Usage

```
db_ts_store(
  con,
  x,
  access = NULL,
  valid_from = NULL,
  release_date = NULL,
  pre_release_access = NULL,
  schema = "timeseries"
)
```

Arguments

con	RPostgres connection object.
x	Object containing time series to store. Single ts or xts objects are allowed as well as objects of type list, tslist, and data.table.
access	character Access level for all ts to be stored. If set to NA (default) the database set it to 'main' access.
valid_from	character representation of a date in the form of 'YYYY-MM-DD'. valid_from starts a new version
release_date	character date from which on this version of the time series should be made available when release date is respected. Applies to all time series in x.
pre_release_access	character Only allow access to the series being stored ahead of the release date to users with this access level. NULL (default) allows everybody. See respect_release_date in db_ts_read.
schema	**character** name of the database schema. Defaults to 'timeseries'

See Also

Other time series functions: db_collection_read_ts(), db_dataset_read_ts(), db_ts_delete_latest_version(), db_ts_delete(), db_ts_get_last_update(), db_ts_read_history(), db_ts_read(), db_ts_trim_history()

Examples

```
## Not run:
# storing zrh_airport data that is a list with two xts objects.
db_ts_store(con = connection, zrh_airport, schema = "schema")

# to store different versions of the data, use parameter valid_from
# different versions are stored with the same key
ch.kof.barometer <- kof_ts["baro_2019m11"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2019-12-01",
  schema = "schema"
)

ch.kof.barometer <- kof_ts["baro_2019m12"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2020-01-01",
  schema = "schema"
)

## End(Not run)
```

`db_ts_trim_history` *Remove Vintages from the Beginning*

Description

Removes any vintages of the given time series that are older than a specified date.

Usage

```
db_ts_trim_history(con, ts_keys, older_than, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`older_than`	Date cut off point
`schema`	**character** name of the database schema. Defaults to 'timeseries'

Details

In some cases only the last few versions of time series are of interest. This function can be used to trim off old vintages that are no longer relevant.

See Also

Other time series functions: `db_collection_read_ts()`, `db_dataset_read_ts()`, `db_ts_delete_latest_version()`, `db_ts_delete()`, `db_ts_get_last_update()`, `db_ts_read_history()`, `db_ts_read()`, `db_ts_store()`

Examples

```
## Not run:

# Store different versions of the time series data
ch.kof.barometer <- kof_ts["baro_2019m11"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2019-12-01",
  schema = "schema"
)

ch.kof.barometer <- kof_ts["baro_2019m12"]
names(ch.kof.barometer) <- c("ch.kof.barometer")
db_ts_store(
  con = connection,
  ch.kof.barometer,
  valid_from = "2020-01-01",
  schema = "schema"
)

db_ts_trim_history(
  con = connection,
  ts_keys = "ch.kof.barometer",
  older_than = "2019-12-31",
  schema = "schema"
)

## End(Not run)
```

`db_with_tmp_read` *Helper to Create and Populate a Temporary Table for Fast Reading*

Description

This function is not exported. It creates a tempory table containing the keys that should be read to join them against the time series storage. This is much faster for larger selections than simple where clauses.

Usage

```
db_with_tmp_read(con, ts_keys, regex = FALSE, code, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`ts_keys`	**character** vector of time series identifiers.
`regex`	logical if set to TRUE, the ts_keys parameter is interpreted as a regular expression pattern.
`code`	expression Code to be evaluated after populating the temporary table on the database of a time series that is valid from the specified date.
`schema`	**character** name of the database schema. Defaults to 'timeseries'

`has_depth_2` *Test if a list has exactly depth 2*

Description

Test if a list has exactly depth 2

Usage

```
has_depth_2(x)
```

Arguments

`x`	The list to check

`index_to_date` *Helper Function for Date Operations*

Description

This function is not exported. Helper function to convert time series indices of the form 2005.75 to a date representation like 2005-07-01. Does not currently support sub-monthly frequencies.

Usage

```
index_to_date(x, as.string = FALSE)
```

Arguments

`x`	numeric A vector of time series time indices (e.g. from stats::time)
`as.string`	logical If as.string is TRUE the string representation of the Date is returned, otherwise a Date object.

Examples

```
## Not run: index_to_date(2020.25)
```

`install_timeseriesdb` *Install timeseriesdb*

Description

Install timeseriesdb in a given PostgreSQL schema. Make sure the database user has sufficient rights to perform the necessary operations on the schema. In the process tables, roles, triggers and functions will be created. Also extensions will be installed and rights will be granted and revoked from the freshly created roles. Note also, that the functions created are created as SECURITY DEFINER roles.

Usage

```
install_timeseriesdb(
  con,
  schema = "timeseries",
  verbose = FALSE,
  install_tables = TRUE,
  install_functions = TRUE
)
```

Arguments

`con`	RPostgres connection object.
`schema`	**character** name of the database schema. Defaults to 'timeseries'
`verbose`	boolean Should progress messages be printed? Default FALSE
`install_tables`	boolean Should the tables be created? Default TRUE
`install_functions`	boolean Should the functions be installed? Default TRUE

Details

`install_tables` and `install_functions` can be used to install components of timeseriesdb independently (e.g. only update function definitions without touching the table structure). They are used mainly for development purposes.

`json_to_ts` *Convert JSON Representation of a Time Series into R Time Series Objects*

Description

This function is not exported.

Usage

```
json_to_ts(jsn, as.dt = FALSE)
```

Arguments

`jsn`	JSON string to convert
`as.dt`	boolean Should the result be returned as a data.table?

Value

R time series representation of class ts, xts or data.table depending on parameter setting and nature of time series. Regular time series can be returned as 'ts' objects whereas irregular time series use 'xts' objects.

`kof_ts` *KOF indicators*

Description

KOF indicators

Usage

```
kof_ts
```

Format

A list with four time series objects:

ch.kof.barometer Indicator for the Swiss Business Cycle.

baro Vintages (versions) of the KOF Barometer Indicator.

ch.kof.ie.retro.ch_total.ind.d11 KOF Employment Indicator for Switzerland

Source

KOF Swiss Economic Institute - KOF indicators. https://kof.ethz.ch/en/forecasts-and-indicators/indicators.html

Examples

```
## Not run:
 kof_ts

## End(Not run)
```

`param_defs` *Common parameters*

Description

Common parameters

Arguments

`con`	RPostgres connection object.
`schema`	**character** name of the database schema. Defaults to 'timeseries'
`ts_keys`	**character** vector of time series identifiers.
`dataset`	**character** name of the dataset. Datasets are group of time series.
`datasets`	**character** vector of the datasets. Dataset is a group of time series.
`valid_on`	**character** representation of a date in the form of 'YYYY-MM-DD'. valid_on selects the version of a time series that is valid at the specified time.
`valid_from`	character representation of a date in the form of 'YYYY-MM-DD'. valid_from starts a new version
`code`	expression Code to be evaluated after populating the temporary table on the database of a time series that is valid from the specified date.
`collection_name`	**character** name of a collection to read. Collection are bookmark lists that contain time series keys.
`access_level`	**character** describing the access level of the time series or dataset.
`set_name`	**character** name of a dataset.
`regex`	**boolean** indicating if ts_keys should be interpreted as a regular expression pattern. Defaults to FALSE.
`locale`	**character** indicating the language of the meta information to be store. We recommend to use ISO country codes to represent languages. Defaults to NULL. When local is set to NULL, metadata are stored without localization. Note that, when localizing meta information by assigning a language, multiple meta information objects can be stored for a single time series.
`respect_release_date`	**boolean** indicating if it should the release embargo of a time series be respected. Defaults to FALSE. This option makes sense when the function is used in an API. In that sense, users do not have direct access to this function and therefore cannot simply switch parameters.

`chunksize`	set a limit of the number of time series requested in the function.
`collection_owner`	**character** username that is the owner of a collection.
`user`	character name of the database user. Defaults to the user of the R session. this is often the user for the database, too so you do not have to specify your username explicitly if that is the case.

`print.meta` *Print Method for meta Object*

Description

Print Method for meta Object

Usage

```
## S3 method for class 'meta'
print(x, ...)
```

Arguments

`x`	a metadata object.
`...`	list of print options.

`setup_sql_extentions` *Install PostgreSQL Schemas and Extensions*

Description

Installs schema, uuid-ossp, btree_gist. This function must be run with a connection of a database level admin.

Usage

```
setup_sql_extentions(con, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object.
`schema`	schema character schema name, defaults to 'timeseries'.

`setup_sql_functions` *Install timeseriesdb System Functions*

Description

Installs functions needed to operated timeseriesdb in a given PostgreSQL schema. The functions uses a default SQL file installed with the package to generate SQL functions. The default schema 'timeseries' can be replaced using the 'schema' parameter.

Usage

```
setup_sql_functions(con, schema = "timeseries", prnt = identity)
```

Arguments

`con`	PostgreSQL connection object created by the RPostgres package.
`schema`	character schema name, defaults to 'timeseries'.
`prnt`	function log printing function

`setup_sql_grant_rights` *Grant execute on timeseriesdb functions*

Description

Grant execute on timeseriesdb functions

Usage

```
setup_sql_grant_rights(con, schema = "timeseries", prnt = identity)
```

Arguments

`con`	RPostgres connection object
`schema`	character schema name, defaults to 'timeseries'
`prnt`	function log printing function

`setup_sql_roles` *Create Roles needed for operation of timeseriesdb*

Description

This function must be run with a connection of a database level admin.

Usage

```
setup_sql_roles(con, schema = "timeseries")
```

Arguments

`con`	RPostgres connection object
`schema`	schema character schema name, defaults to 'timeseries'.

`setup_sql_tables` *Install timeseriesdb System Tables*

Description

Installs tables needed to operated timeseriesdb in a given PostgreSQL schema. The tables use a default SQL file installed with the package to generate SQL tables. The default schema 'timeseries' can be replaced using the 'schema' parameter.

Usage

```
setup_sql_tables(con, schema = "timeseries", prnt = identity)
```

Arguments

`con`	PostgreSQL connection object created by the RPostgres package.
`schema`	character schema name, defaults to 'timeseries'.
`prnt`	function log printing function

`setup_sql_triggers` *Install timeseriesdb Triggers*

Description

Installs functions needed for timeseriesdb triggers and sets up these triggers in a given PostgreSQL schema. The functions uses a default SQL file installed with the package to generate SQL functions. The default schema 'timeseries' can be replaced using the 'schema' parameter.

Usage

```
setup_sql_triggers(con, schema = "timeseries", prnt = identity)
```

Arguments

`con`	PostgreSQL connection object created by the RPostgres package.
`schema`	character schema name, defaults to 'timeseries'.
`prnt`	function log printing function

Index