Package: PEcAn.DB (via r-universe)

June 27, 2024

Type Package

Title PEcAn Functions Used for Ecological Forecasts and Reanalysis

Version 1.7.2.9000

Date 2021-10-04

Description The Predictive Ecosystem Carbon Analyzer (PEcAn) is a scientific workflow management tool that is designed to simplify the management of model parameterization, execution, and analysis. The goal of PECAn is to streamline the interaction between data and models, and to improve the efficacy of scientific investigation.

Imports curl, DBI, dbplyr (>= 2.4.0), dplyr (>= 1.1.2), fs, glue, lubridate, magrittr, methods, ncdf4, PEcAn.logger, PEcAn.remote, PEcAn.utils, purrr, R.utils, rlang, tibble, tidyr, units, XML

Suggests bit64, data.table, here, knitr (>= 1.42), mockery (>= 0.4.3), RPostgreSQL, RPostgres, RSQLite, rcrossref, rmarkdown (>= 2.19), testthat (>= 2.0.0), tidyverse, withr

License BSD_3_clause + file LICENSE

VignetteBuilder knitr

Copyright Authors

LazyLoad yes

LazyData FALSE

Encoding UTF-8

RoxygenNote 7.3.1

Repository https://pecanproject.r-universe.dev

RemoteUrl https://github.com/PecanProject/pecan

RemoteRef HEAD

RemoteSha d5c7bffdf233077968945a182c11240b5d76e42d

Contents

`append.covariate` *Append covariate data as a column within a table*

Description

`append.covariate` appends a data frame of covariates as a new column in a data frame of trait data. In the event a trait has several covariates available, the first one found (i.e. lowest row number) will take precedence

Usage

```
append.covariate(data, column.name, covariates.data)
```

Arguments

`data`	trait dataframe that will be appended to.
`column.name`	name of the covariate as it will appear in the appended column
`covariates.data`	one or more tables of covariate data, ordered by the precedence they will assume in the event a trait has covariates across multiple tables. All tables must contain an 'id' and 'level' column, at minimum.

Author(s)

Carl Davidson, Ryan Kelly

`arrhenius.scaling.traits` *Apply Arrhenius scaling to 25 degC for temperature-dependent traits*

Description

Apply Arrhenius scaling to 25 degC for temperature-dependent traits

Usage

```
arrhenius.scaling.traits(
  data,
  covariates,
  temp.covariates,
  new.temp = 25,
  missing.temp = 25
)
```

Arguments

`data`	data frame of data to scale, as returned by query.data()
`covariates`	data frame of covariates, as returned by query.covariates(). Note that data with no matching covariates will be unchanged.
`temp.covariates`	names of covariates used to adjust for temperature; if length > 1, order matters (first will be used preferentially)
`new.temp`	the reference temperature for the scaled traits. Curerntly 25 degC
`missing.temp`	the temperature assumed for traits with no covariate found. Curerntly 25 degC

Author(s)

Carl Davidson, David LeBauer, Ryan Kelly

assign.treatments *assign.treatments*

Description

Change treatments to sequential integers

Usage

```
assign.treatments(data)
```

Arguments

data	input data

Details

Assigns all control treatments the same value, then assigns unique treatments within each site. Each site is required to have a control treatment. The algorithm (incorrectly) assumes that each site has a unique set of experimental treatments. This assumption is required by the data in BETYdb that does not always consistently name treatments or quantity them in the managements table. Also it avoids having the need to estimate treatment by site interactions in the meta analysis model. This model uses data in the control treatment to estimate model parameters so the impact of the assumption is minimal.

Value

dataframe with sequential treatments

Author(s)

David LeBauer, Carl Davidson, Alexey Shiklomanov

bety2pecan *Convert BETY variable names to MsTMIP and subsequently PEcAn standard names*

Description

Convert BETY variable names to MsTMIP and subsequently PEcAn standard names

Usage

```
bety2pecan(vars_bety)
```

Arguments

`vars_bety`	data frame with variable names and units

Author(s)

Betsy Cowdery

`betyConnect` *Connect to bety using current PEcAn configuration*

Description

Connect to bety using current PEcAn configuration

Usage

```
betyConnect(php.config = "../../web/config.php")
```

Arguments

`php.config`	Path to ‘config.php‘

`check.lists` *Compares two lists*

Description

Check two lists. Identical does not work since one can be loaded from the database and the other from a CSV file.

Usage

```
check.lists(x, y, filename = "species.csv")
```

Arguments

`x`	first list
`y`	second list
`filename`	one of "species.csv" or "cultivars.csv"

Value

true if two lists are the same

Author(s)

Rob Kooper

`clone_pft` *Duplicate existing pft with associated priors, species, and cultivars*

Description

Creates a new pft that is a duplicate of an existing pft, including relationships with priors, species, and cultivars (if any) of the existing pft. This function mimics the 'clone pft' button in the PFTs record view page in the BETYdb web interface for PFTs that aggregate >=1 species, but adds the ability to clone the cultivar associations.

Usage

```
clone_pft(parent.pft.name, new.pft.name, new.pft.definition, settings)
```

Arguments

`parent.pft.name`	name of PFT to duplicate
`new.pft.name`	name for new PFT. Must not be the same as parent.pft.name
`new.pft.definition`	text for the new PFT's definition field.
`settings`	PEcAn settings list, used only for BETYdb connection parameters

Value

ID of the newly created pft in database, creates new PFT as a side effect

Author(s)

David LeBauer, Chris Black

Examples

```
## Not run:
clone_pft(parent.pft.name    = "tempdecid",
          new.pft.name       = "mytempdecid",
          new.pft.definition = "mytempdecid is a new pft",
          settings = pecan_settings_list)

## End(Not run)
```

convert_input *Convert between formats, reusing existing files where possible*

Description

`convert_input` is a relatively generic function that applies the function `fcn` and inserts a record of it into the database. It is primarily designed for converting meteorological data between formats and can be used on observed data, forecasts, and ensembles of forecasts. To minimize downloading and storing duplicate data, it first checks to see if a given file is already in the database before applying `fcn`.

Usage

```
convert_input(
  input.id,
  outfolder,
  formatname,
  mimetype,
  site.id,
  start_date,
  end_date,
  pkg,
  fcn,
  con = con,
  host,
  browndog,
  write = TRUE,
  format.vars,
  overwrite = FALSE,
  exact.dates = FALSE,
  allow.conflicting.dates = TRUE,
  insert.new.file = FALSE,
  pattern = NULL,
  forecast = FALSE,
  ensemble = FALSE,
  ensemble_name = NULL,
  dbparms = NULL,
  ...
)
```

Arguments

`input.id`	The database id of the input file of the parent of the file being processed here. The parent will have the same data, but in a different format.
`outfolder`	The directory where files generated by functions called by convert_input will be placed

`formatname`	data product specific format name
`mimetype`	data product specific file format
`site.id`	The id of the site
`start_date`	Start date of the data being requested or processed
`end_date`	End date of the data being requested or processed
`pkg`	The package that the function being executed is in (as a string)
`fcn`	The function to be executed if records of the output file aren't found in the database. (as a string)
`con`	Database connection object
`host`	Named list identifying the machine where conversion should be performed. Currently only `host$name` and `host$Rbinary` are used by `convert_input`, but the whole list is passed to other functions
`browndog`	List of information related to browndog conversion. NULL if browndog is not to be used for conversion
`write`	Logical: Write new file records to the database?
`format.vars`	Passed on as arguments to `fcn`
`overwrite`	Logical: If a file already exists, create a fresh copy? Passed along to fcn.
`exact.dates`	Ignore time-span appending and enforce exact start and end dates on the database input file? (logical)
`allow.conflicting.dates`	Should overlapping years ignore time-span appending and exist as separate input files? (logical)
`insert.new.file`	Logical: force creation of a new database record even if one already exists?
`pattern`	A regular expression, passed to `dbfile.input.check`, used to match the name of the input file.
`forecast`	Logical: Is the data product a forecast?
`ensemble`	An integer representing the number of ensembles, or FALSE if it data product is not an ensemble.
`ensemble_name`	If convert_input is being called iteratively for each ensemble, ensemble_name contains the identifying name/number for that ensemble.
`dbparms`	list of parameters to use for opening a database connection
`...`	Additional arguments, passed unchanged to `fcn`

Value

A list of two BETY IDs (input.id, dbfile.id) identifying a pre-existing file if one was available, or a newly created file if not. Each id may be a vector of ids if the function is processing an entire ensemble at once.

Executing the function

convert_input executes the function fcn in package pkg via PEcAn.remote::remote.execute.R. All additional arguments passed to convert_input (...) are in turn passed along to fcn as arguments. In addition, several named arguments to convert_input are passed along to fcn. The command to execute fcn is built as a string.

Database files

There are two kinds of database records (in different tables) that represent a given data file in the file system. An input file contains information about the contents of the data file. A dbfile contains machine spacific information for a given input file, such as the file path. Because duplicates of data files for a given input can be on multiple different machines, there can be more than one dbfile for a given input file.

Time-span appending

By default, convert_input tries to optimize the download of most data products by only downloading the years of data not present on the current machine. (For example, if files for 2004-2008 exist for a given data product exist on this machine and the user requests 2006-2010, the function will only download data for 2009 and 2010). In year-long data files, each year exists as a separate file. The database input file contains records of the bounds of the range stored by those years. The data optimization can be turned off by overriding the default values for exact.dates and allow.conflicting.dates.

Forecast data

If the flag forecast is TRUE, convert_input treats data as if it were forecast data. Forecast data do not undergo time span appending.

Ensembles

convert_input has the capability to handle ensembles of met data. If ensemble = an integer > 1, convert_input checks the database for records of all ensemble members, and calls fcn if at least one is missing. convert_input assumes that fcn will return records for all ensembles. convert_input can also be called iterately for each ensemble member. In this case ensemble_name contains the unique identifying name/number of the ensemble to be processed.

Author(s)

Betsy Cowdery, Michael Dietze, Ankur Desai, Tony Gardella, Luke Dramko

db.close *Generic function to close a database connection*

Description

Close a previously opened connection to a database.

Usage

```
db.close(con, showWarnings = TRUE)
```

Arguments

`con`	database connection to be closed
`showWarnings`	logical: report possible issues with connection?

Value

'TRUE', invisibly (see [DBI::dbDisconnect()])

Author(s)

Rob Kooper

Examples

```
## Not run:
db.close(con)

## End(Not run)
```

db.exists *Test connection to database*

Description

Useful to only run tests that depend on database when a connection exists

Usage

```
db.exists(params, write = TRUE, table = NA)
```

Arguments

`params`	database connection information
`write`	logical: test whether we have write access?
`table`	name of database table to check

Value

TRUE if database connection works; else FALSE

Author(s)

David LeBauer, Rob Kooper

`db.getShowQueries` *db.getShowQueries*

Description

Returns if the queries should be shown that are being executed

Usage

```
db.getShowQueries()
```

Value

will return TRUE if queries are shown

Author(s)

Rob Kooper

`db.open` *Open a database connection*

Description

Create a connection to a database using the specified parameters. The ‘params‘ list will be passed as arguments to [DBI::dbConnect()].

Usage

```
db.open(params)
```

Arguments

`params`	Named list of database connection options. See details

Details

Typical arguments are as follows: - ‘driver‘ – The name of the database driver. Only ‘"PostgreSQL"‘ and ‘"Postgres"‘ are supported. If no driver is specified, default to ‘"PostgreSQL"‘. - ‘user‘ – The database username. For local instances of PEcAn, this is usually ‘"bety"‘. - ‘password‘ – The database password. For local instances of PEcAn, this is usually ‘"bety"‘. - ‘host‘ – The database hostname. For local instances of PEcAn, this is usually ‘"localhost"‘. Inside the PEcAn Docker stack, this may be ‘"postgres"‘. - ‘port‘ (optional) – The port for accessing the database. If omitted, this will use the PostgreSQL default (5432).

Value

Database connection object

Author(s)

Rob Kooper

Examples

```
## Not run:
db.open(settings$database$bety)

## End(Not run)
```

db.print.connections *Debug leaked connections*

Description

Prints the number of connections opened as well as any connections that have never been closes.

Usage

```
db.print.connections()
```

Author(s)

Rob Kooper

Examples

```
## Not run:
db.print.connections()

## End(Not run)
```

`db.query` *Generic function to query database*

Description

Given a connection and a query, will return a query as a data frame. Either con or params need to be specified. If both are specified it will use con.

Usage

```
db.query(query, con = NULL, params = NULL, values = NULL)
```

Arguments

`query`	SQL query string
`con`	Database connection object
`params`	Named list of database connection parameters. See ‘params‘ argument to [db.open()].
`values`	If using prepared statements, a list of values to substitute into the query. If ‘NULL‘ (default), execute the query directly. See this function’s "Details", and documentation for [DBI::dbBind()].

Details

This function supports prepared statements, which provide a way to pass data into SQL queries without the risk of SQL injection attacks. Whenever you are tempted to do something like this:

“‘ db.query(paste0("SELECT * FROM table WHERE mycol = ", somevalue, " AND othercol = ", othervalue), con = con) “‘

...use a prepared query instead:

“‘ db.query("SELECT * FROM table WHERE mycol = $1 AND othercol = $2", values = list(somevalue, othervalue), con = con) “‘

Besides preventing SQL injections, prepared statements also ensure that the input and target types are compatible.

Prepared statements provide an efficient way to operate on multiple values at once. For example, the following will return all the models whose revision is either "git", "46", or "unk":

“‘ db.query("SELECT * FROM models WHERE revision = $1", values = list(c("git", "46", "unk")), con = con) “‘

...and here is an example of inserting multiple values of a given trait for a given species:

“‘ db.query("INSERT INTO traits (specie_id, variable_id, mean, n) VALUES ($1, $2, $3)", values = list(938, 396, c(1.7, 3.9, 4.5), 1), con = con) “‘

Note that prepared statements **can not be used to select tables or columns**. In other words, the following _will not work_ because of the following placeholders, the only valid one is ‘$5‘:

“‘ # This will not work! db.query("SELECT $1, $2 FROM $3 WHERE $4 = $5", values = list("col1", "col2", "mytable", "somecolumn", "somevalue")) “‘

Note that prepared statements **are not supported by the ‘RPostgreSQL‘ package**; only by the newer ‘RPostgres‘ package.

Value

data frame with query results

Author(s)

Rob Kooper, Alexey Shiklomanov

Examples

```
## Not run:
db.query("SELECT count(id) FROM traits;", params = settings$database$bety)

# Prepared statements
con <- db.open(settings$database$bety)
db.query(
  "SELECT * FROM table WHERE mycol = $1 AND othercol = $2",
  values = list(somevalue, othervalue),
  con = con
)

# Select multiple values at once; rbind the result
db.query(
  "SELECT * FROM models WHERE revision = $1",
  values = list(c("git", "46", "unk")),
  con = con
)

# Efficiently insert multiple values into a table
db.query(
  "INSERT INTO traits (specie_id, variable_id, mean, n) VALUES ($1, $2, $3, $4)",
  values = list(938, 396, rnorm(1000), 1),
  con = con
)

## End(Not run)
```

`db.showQueries` *db.showQueries*

Description

Sets if the queries should be shown that are being executed

Usage

```
db.showQueries(show)
```

Arguments

`show`	set to TRUE to show the queries, FALSE by default

Details

Useful to print queries when debuging SQL statements

Author(s)

Rob Kooper

`dbfile.check` *List files associated with a container and machine exist in ‘dbfiles‘ table*

Description

List files associated with a container and machine exist in ‘dbfiles‘ table

Usage

```
dbfile.check(
  type,
  container.id,
  con,
  hostname = PEcAn.remote::fqdn(),
  machine.check = TRUE,
  return.all = FALSE
)
```

Arguments

`type`	The type of ‘dbfile‘, as a character. Must be either "Input", "Posterior", or "Model".
`container.id`	the ID of container type. E.g. if ‘type‘ is "Input", this will correspond to the ‘id‘ column from the ‘inputs‘ table.
`con`	database connection object
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine
`machine.check`	setting to check for file on named host, otherwise will check for any file given container id
`return.all`	(Logical) If ‘TRUE‘, return all files. If ‘FALSE‘, return only the most recent files.

Value

‘data.frame‘ with the id, filename and pathname of all the files that are associated

Author(s)

Rob Kooper, Alexey Shiklomanov

Examples

```
## Not run:
  dbfile.check("Input", 7, dbcon)

## End(Not run)
```

---

dbfile.file *Convert between file paths and ids*

Description

These functions check the dbfiles and machines tables to see if the file exists, and return the container_id (`dbfile.id`) or full filename with path (`dbfile.file`) to the first one found. If none is found, both will return NA.

Usage

```
dbfile.file(type, id, con, hostname = PEcAn.remote::fqdn())

dbfile.id(type, file, con, hostname = PEcAn.remote::fqdn())
```

Arguments

`type`	the type of dbfile (Input, Posterior)
`id`	the id of container type
`con`	database connection object
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine
`file`	the full pathname to the file

Value

filename on host, or NA if none found

Functions

- `dbfile.file()`: Return full path to file from the dbfiles table
- `dbfile.id()`: Return id to container type given a filename

Author(s)

Rob Kooper

Examples

```
## Not run:
  dbfile.file('Input', 7, dbcon)

## End(Not run)
## Not run:
  dbfile.id('Model', '/usr/local/bin/sipnet', dbcon)

## End(Not run)
```

`dbfile.input.check` *Check for a file in the input/dbfiles tables*

Description

Function to check to see if a file exists in the dbfiles table as an input

Usage

```
dbfile.input.check(
  siteid,
  startdate = NULL,
  enddate = NULL,
  mimetype,
  formatname,
  parentid = NA,
  con,
  hostname = PEcAn.remote::fqdn(),
  exact.dates = FALSE,
  pattern = NULL,
  return.all = FALSE
)
```

Arguments

`siteid`	the id of the site that this data is applicable to
`startdate`	the start date of the data stored in the file
`enddate`	the end date of the data stored in the file
`mimetype`	the mime-type of the file
`formatname`	the name of the format to distinguish between simmilair mime-types
`parentid`	the id of the parent of the input
`con`	database connection object
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine
`exact.dates`	setting to include start and end date in input query

`pattern`	text to seach for in the file name (default NULL = no check).
`return.all`	(Logical) If 'TRUE', return all files. If 'FALSE', return only the most recent files.

Details

This will check the dbfiles, inputs, machines and formats tables to see if the file exists

Value

data.frame with the id, filename and pathname of the input that is requested

Author(s)

Rob Kooper, Tony Gardella, Hamze Dokoohaki

Examples

```
## Not run:
  dbfile.input.check(siteid, startdate, enddate, 'application/x-RData', 'traits', dbcon)

## End(Not run)
```

`dbfile.input.insert` *Insert file into tables*

Description

Function to insert a file into the dbfiles table as an input

Usage

```
dbfile.input.insert(
  in.path,
  in.prefix,
  siteid,
  startdate,
  enddate,
  mimetype,
  formatname,
  parentid = NA,
  con,
  hostname = PEcAn.remote::fqdn(),
  allow.conflicting.dates = FALSE,
  ens = FALSE
)
```

Arguments

`in.path`	path to the directory containing the file to be inserted
`in.prefix`	initial portion of the filename that does not vary by date. Does not include directory; specify that as part of in.path
`siteid`	the id of the site that this data is applicable to
`startdate`	the start date of the data stored in the file
`enddate`	the end date of the data stored in the file
`mimetype`	the mime-type of the file
`formatname`	the name of the format to distinguish between simmilair mime-types
`parentid`	the id of the parent of the input
`con`	database connection object
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine
`allow.conflicting.dates`	Whether to allow a new input record with same siteid, name, and format but different start/end dates
`ens`	In case of ensembles we could let to have more than one file associated with one input.

Details

This will write into the dbfiles, inputs, machines and formats the required data to store the file

Value

data.frame with the id, filename and pathname of the input that is requested

Author(s)

Rob Kooper, Betsy Cowdery

Examples

```
## Not run:
  dbfile.input.insert(
    in.path = 'trait.data.Rdata',
    in.prefix = siteid,
    startdate = startdate,
    enddate = enddate,
    mimetype = 'application/x-RData',
    formatname = 'traits',
    con = dbcon)

## End(Not run)
```

dbfile.insert *Insert file into tables*

Description

Function to insert a file into the dbfiles table

Usage

```
dbfile.insert(
  in.path,
  in.prefix,
  type,
  id,
  con,
  reuse = TRUE,
  hostname = PEcAn.remote::fqdn()
)
```

Arguments

`in.path`	Path to file directory
`in.prefix`	Filename prefix (not including directory)
`type`	One of "Model", "Posterior", "Input"
`id`	container_id of the input to be modified
`con`	database connection object
`reuse`	logical: If a record already exists, use it or create a new one?
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine

Details

This will write into the dbfiles and machines the required data to store the file

Value

id of the file that is written

Author(s)

Rob Kooper, Ryan Kelly

Examples

```
## Not run:
  dbfile.insert('somefile.txt', 'Input', 7, dbcon)

## End(Not run)
```

dbfile.move *Move files to new location*

Description

This function will move dbfiles - clim or nc - from one location to another on the same machine and update BETY

Usage

```
dbfile.move(old.dir, new.dir, file.type, siteid = NULL, register = FALSE)
```

Arguments

`old.dir`	directory with files to be moved
`new.dir`	directory where files should be moved
`file.type`	what type of files are being moved
`siteid`	needed to register files that arent already in BETY
`register`	if file isn't already in BETY, should it be registered?

Value

print statement of how many files were moved, registered, or have symbolic links

Author(s)

kzarada

Examples

```
## Not run:
  dbfile.move(
  old.dir = "/fs/data3/kzarada/pecan.data/dbfiles/NOAA_GEFS_site_0-676",
  new.dir = '/projectnb/dietzelab/pecan.data/dbfiles/NOAA_GEFS_site_0-676'
  file.type= clim,
  siteid = 676,
  register = TRUE
  )

## End(Not run)
```

`dbfile.posterior.check` *Check for a file in the input/dbfiles tables*

Description

Function to check to see if a file exists in the dbfiles table as an input

Usage

```
dbfile.posterior.check(
  pft,
  mimetype,
  formatname,
  con,
  hostname = PEcAn.remote::fqdn()
)
```

Arguments

`pft`	the name of the pft that this data is applicable to
`mimetype`	the mime-type of the file
`formatname`	the name of the format to distinguish between simmilair mime-types
`con`	database connection object
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine

Details

This will check the dbfiles, inputs, machines and formats tables to see if the file exists

Value

data.frame with the id, filename and pathname of the posterior that is requested

Author(s)

Rob Kooper

Examples

```
## Not run:
  dbfile.posterior.check(pft, 'application/x-RData', 'traits', dbcon)

## End(Not run)
```

dbfile.posterior.insert

Insert file into tables

Description

Function to insert a file into the dbfiles table as a posterior

Usage

```
dbfile.posterior.insert(
  filename,
  pft,
  mimetype,
  formatname,
  con,
  hostname = PEcAn.remote::fqdn()
)
```

Arguments

`filename`	the name of the file to be inserted
`pft`	the name of the pft that this data is applicable to
`mimetype`	the mime-type of the file
`formatname`	the name of the format to distinguish between simmilair mime-types
`con`	database connection object
`hostname`	the name of the host where the file is stored, this will default to the name of the current machine

Details

This will write into the dbfiles, posteriors, machines and formats the require data to store the file

Value

data.frame with the id, filename and pathname of the posterior that is requested

Author(s)

Rob Kooper

Examples

```
## Not run:
 dbfile.posterior.insert('trait.data.Rdata', pft, 'application/x-RData', 'traits', dbcon)

## End(Not run)
```

dbHostInfo *Database host information*

Description

Database host information

Usage

```
dbHostInfo(bety)
```

Arguments

bety	BETYdb connection, as opened by ‘betyConnect()‘

db_merge_into *Merge local data frame into SQL table*

Description

Merge local data frame into SQL table

Usage

```
db_merge_into(values, table, con, by = NULL, drop = FALSE, ...)
```

Arguments

values	‘data.frame‘ of values to write to SQL database
table	Name of target SQL table, as character
con	Database connection object
by	Character vector of columns by which to perform merge. Defaults to all columns in ‘values‘
drop	logical. If ‘TRUE‘ (default), drop columns not found in SQL table.
...	Arguments passed on to insert_table `coerce_col_class` logical, whether or not to coerce local data columns to SQL classes. Default = ‘TRUE.‘

Value

Data frame: Inner join of SQL table and input data frame (as unevaluated "lazy query" table)

Examples

```
irisdb <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
dplyr::copy_to(irisdb, iris[1:10,], name = "iris", overwrite = TRUE)
db_merge_into(iris[1:12,], "iris", irisdb)
dplyr::tbl(irisdb, "iris") %>% dplyr::count()
```

`default_hostname` *default_hostname*

Description

Convenience function to fix hostname if localhost

Usage

```
default_hostname(hostname)
```

Arguments

`hostname`	character

Value

hostname

`derive.trait` *Performs an arithmetic function, FUN, over a series of traits and returns the result as a derived trait.*

Description

Performs an arithmetic function, FUN, over a series of traits and returns the result as a derived trait. Traits must be specified as either lists or single row data frames, and must be either single data points or normally distributed. In the event one or more input traits are normally distributed, the resulting distribution is approximated by numerical simulation. The output trait is effectively a copy of the first input trait with modified mean, stat, and n.

Usage

```
derive.trait(FUN, ..., input = list(...), var.name = NA, sample.size = 10^6)
```

Arguments

`FUN`	arithmetic function
`...`	traits that will be supplied to FUN as input
`input`	list of trait inputs. See examples
`var.name`	name to use in output
`sample.size`	number of random samples generated by rnorm for normally distributed trait input

Value

a copy of the first input trait with mean, stat, and n reflecting the derived trait

Examples

```
input <- list(x = data.frame(mean = 1, stat = 1, n = 1))
derive.trait(FUN = identity, input = input, var.name = 'x')
```

`derive.traits` *Performs an arithmetic function, FUN, over a series of traits and returns the result as a derived trait.*

Description

Equivalent to derive.trait(), but operates over a series of trait datasets, as opposed to individual trait rows. See `derive.trait`; for more information.

Usage

```
derive.traits(
  FUN,
  ...,
  input = list(...),
  match.columns = c("citation_id", "site_id", "specie_id"),
  var.name = NA,
  sample.size = 10^6
)
```

Arguments

`FUN`	arithmetic function
`...`	trait datasets that will be supplied to FUN as input
`input`	list of trait inputs. See examples in `derive.trait`
`match.columns`	in the event more than one trait dataset is supplied, this specifies the columns that identify a unique data point
`var.name`	name to use in output
`sample.size`	where traits are normally distributed with a given

Value

a copy of the first input trait with modified mean, stat, and n

---

`dplyr.count` *Count rows of a data frame*

Description

Count rows of a data frame

Usage

```
dplyr.count(df)
```

Arguments

`df`	Data frame of which to count length

`fancy_scientific` *Convert number to scientific notation pretty expression*

Description

Convert number to scientific notation pretty expression

Usage

```
fancy_scientific(l)
```

Arguments

`l`	Number to convert to scientific notation

`fetch.stats2se` *Fetch data and transform stats to SE*

Description

Queries data from the trait database and transforms statistics to SE

Usage

```
fetch.stats2se(connection, query)
```

Arguments

`connection`	connection to trait database
`query`	to send to databse

Details

Performs query and then uses `transformstats` to convert miscellaneous statistical summaries to SE

Value

dataframe with trait data

Author(s)

<unknown>

See Also

used in `query.trait.data`; `transformstats` performs transformation calculations

`filter_sunleaf_traits` *Function to filter out upper canopy leaves*

Description

Function to filter out upper canopy leaves

Usage

```
filter_sunleaf_traits(data, covariates)
```

Arguments

`data`	input data
`covariates`	covariate data

Author(s)

David LeBauer

`get.id` *get.id*

Description

Retrieve id from a table matching query

Usage

```
get.id(table, colnames, values, con, create = FALSE, dates = TRUE)
```

Arguments

`table`	name of table
`colnames`	names of one or more columns used in where clause
`values`	values to be queried in fields corresponding to colnames
`con`	database connection object,
`create`	logical: make a record if none found?
`dates`	Ignored. Formerly indicated whether to set created_at and updated_at timestamps when ‘create‘ was TRUE, but the database now always sets them automatically

Value

will numeric

Author(s)

David LeBauer

Examples

```
## Not run:
pftid <- get.id("pfts", "name", "salix", con)
pftid <- get.id("pfts", c("name", "modeltype_id"), c("ebifarm.salix", 1), con)

## End(Not run)
```

get.trait.data *Get trait data from the database.*

Description

This will use the following items from settings: - ‘settings$pfts‘ - ‘settings$model$type‘ - ‘settings$database$bety‘ - ‘settings$database$dbfiles‘ - ‘settings$meta.analysis$update‘

Usage

```
get.trait.data(
  pfts,
  modeltype,
  dbfiles,
  database,
  forceupdate,
  write = FALSE,
  trait.names = NULL
)
```

Arguments

`pfts`	the list of pfts to get traits for
`modeltype`	type of model that is used, this is is used to distinguish between different PFTs with the same name.
`dbfiles`	location where previous results are found
`database`	database connection parameters (see ‘params‘ argument to [db.query()])
`forceupdate`	(Logical) If ‘TRUE‘, force a database update whether or not it is needed. If ‘FALSE‘, only update if an update is needed.
`write`	(Logical) If ‘TRUE‘ updated posteriors will be written to BETYdb. Defaults to FALSE.
`trait.names`	Character vector of trait names to search. If ‘NULL‘ (default), use all traits that have a prior for at least one of the ‘pfts‘.

Value

list of PFTs with update posteriorids

Author(s)

David LeBauer, Shawn Serbin, Alexey Shiklomanov

`get.trait.data.pft` *Get trait data from the database for a single PFT*

Description

Get trait data from the database for a single PFT

Usage

```
get.trait.data.pft(
  pft,
  modeltype,
  dbfiles,
  dbcon,
  trait.names,
  forceupdate = FALSE,
  write = FALSE
)
```

Arguments

`pft`	list of settings for the pft whose traits to retrieve. See details
`modeltype`	type of model that is used, this is used to distinguish between different pfts with the same name.
`dbfiles`	location where previous results are found
`dbcon`	database connection
`trait.names`	list of trait names to retrieve
`forceupdate`	set this to true to force an update, auto will check to see if an update is needed.
`write`	(Logical) If ‘TRUE‘ updated posteriors will be written to BETYdb. Defaults to FALSE.

Details

‘pft‘ should be a list containing at least ‘name‘ and ‘outdir‘, and optionally ‘posteriorid‘ and ‘constants‘. BEWARE: All existing files in ‘outir‘ will be deleted!

Value

updated pft with posteriorid

Author(s)

David LeBauer, Shawn Serbin, Rob Kooper

get_postgres_envvars *Look up Postgres connection parameters from environment variables*

Description

Retrieves database connection parameters stored in any of the environment variables known by Postgres, using defaults from '...' for parameters not set in the environment. In a standard PEcAn installation only a few of these parameters will ever be set, but we check all of them anyway in case you need to do anything unusual.

Usage

```
get_postgres_envvars(...)
```

Arguments

`...`	defaults for parameters not found in the environment, in 'name = value' form

Details

The list of environment variables we check is taken from the [Postgres 12 manual](https://postgresql.org/docs/12/libpq-envars.html), but it should apply to older Postgres versions as well. Note that this function only looks for environment variables that control connection parameters; it does not retrieve any of the variables related to per-session behavior (e.g. PGTZ, PGSYSCONFDIR).

Value

list of connection parameters suitable for passing on to 'db.open'

Examples

```
host <- Sys.getenv("PGHOST") # to restore environment after demo

Sys.unsetenv("PGHOST")
get_postgres_envvars()$host # NULL
get_postgres_envvars(host = "default", port = 5432)$host # "default"
# defaults are ignored for a variable that exists
Sys.setenv(PGHOST = "localhost")
get_postgres_envvars()$host # "localhost"
get_postgres_envvars(host = "postgres")$host # still "localhost"

# To override a set variable, edit the returned list before using it
con_parms <- get_postgres_envvars()
con_parms$host # "localhost"
con_parms$host <- "postgres"
# db.open(con_parms)

Sys.setenv(PGHOST = host)
```

`get_run_ids` *Get vector of run IDs for a given workflow ID*

Description

Get vector of run IDs for a given workflow ID

Usage

```
get_run_ids(bety, workflow_id)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘
`workflow_id`	Workflow ID

`get_users` *Get data frame of users and IDs*

Description

Get data frame of users and IDs

Usage

```
get_users(bety)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘

`get_var_names` *Get vector of variable names for a particular workflow and run ID*

Description

Get vector of variable names for a particular workflow and run ID

Usage

```
get_var_names(bety, workflow_id, run_id, remove_pool = TRUE)
```

Arguments

`bety`	BETYdb connection, as opened by 'betyConnect()'
`workflow_id`	Workflow ID
`run_id`	Run ID
`remove_pool`	logical: ignore variables with 'pools' in their names?

`get_workflow_ids` *Get vector of workflow IDs*

Description

Get vector of workflow IDs

Usage

```
get_workflow_ids(bety, query, all.ids = FALSE)
```

Arguments

`bety`	BETYdb connection, as opened by 'betyConnect()'
`query`	Named vector or list of workflow IDs
`all.ids`	logical: return a list of all workflow_ids in the BETY database, or just those that are part of the query?

`insert.format.vars` *Insert Format and Format-Variable Records*

Description

Insert Format and Format-Variable Records

Usage

```
insert.format.vars(
  con,
  format_name,
  mimetype_id,
  notes = NULL,
  header = TRUE,
  skip = 0,
  formats_variables = NULL,
  suppress = TRUE
)
```

Arguments

`con`	SQL connection to BETYdb
`format_name`	The name of the format. Type: character string.
`mimetype_id`	The id associated with the mimetype of the format. Type: integer.
`notes`	Additional description of the format: character string.
`header`	Boolean that indicates the presence of a header in the format. Defaults to "TRUE".
`skip`	Integer that indicates the number of lines to skip in the header. Defaults to 0.
`formats_variables`	A 'tibble' consisting of entries that correspond to columns in the formats-variables table. See Details for further information.
`suppress`	Boolean that suppresses or allows a test for an existing variable id. This test is inconvenient in applications where the variable_ids are already known.

Details

The formats_variables argument must be a 'tibble' and be structured in a specific format so that the SQL query functions properly. All arguments should be passed as vectors so that each entry will correspond with a specific row. All empty values should be specified as NA.

variable_id (Required) Vector of integers.

name (Optional) Vector of character strings. The variable name in the imported data need only be specified if it differs from the BETY variable name.

unit (Optional) Vector of type character string. Should be in a format parseable by the udunits library and need only be secified if the units of the data in the file differ from the BETY standard.

storage_type (Optional) Vector of character strings. Storage type need only be specified if the variable is stored in a format other than would be expected (e.g. if numeric values are stored as quoted character strings). Additionally, storage_type stores POSIX codes that are used to store any time variables (e.g. a column with a 4-digit year would be %Y). See also `[base::strptime]`

column_number Vector of integers that list the column numbers associated with variables in a dataset. Required for text files that lack headers.

Value

format_id

Author(s)

Liam Burke (liam.burke24@gmail.com)

Examples

```
## Not run:
con <- PEcAn.DB::betyConnect()

formats_variables_tibble <- tibble::tibble(
  variable_id = c(411, 135, 382),
  name = c("NPP", NA, "YEAR"),
  unit = c("g C m-2 yr-1", NA, NA),
  storage_type = c(NA, NA, "%Y"),
  column_number = c(2, NA, 4))

insert.format.vars(
  con = con,
  format_name = "LTER-HFR-103",
  mimetype_id = 1090,
  notes = "NPP from Harvard Forest.",
  header = FALSE,
  skip = 0,
  formats_variables = formats_variables_tibble)

## End(Not run)
```

---

insert_table *Insert R data frame into SQL database*

Description

First, subset to matching columns. Then, make sure the local and SQL column classes match, coercing local to SQL as necessary (or throwing an error). Then, build an SQL string for the insert statement. Finally, insert into the database.

Usage

```
insert_table(values, table, con, coerce_col_class = TRUE, drop = TRUE)
```

Arguments

`values`	‘data.frame‘ of values to write to SQL database
`table`	Name of target SQL table, as character
`con`	Database connection object
`coerce_col_class`	logical, whether or not to coerce local data columns to SQL classes. Default = ‘TRUE.‘
`drop`	logical. If ‘TRUE‘ (default), drop columns not found in SQL table.

Value

data frame with query results

Examples

```
irisdb <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
dplyr::copy_to(irisdb, iris[1,], name = "iris", overwrite = TRUE)
insert_table(iris[-1,], "iris", irisdb)
dplyr::tbl(irisdb, "iris")
```

`load_data_single_run` *Load data for a single run of the model*

Description

Load data for a single run of the model

Usage

```
load_data_single_run(bety, workflow_id, run_id)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘
`workflow_id`	Workflow ID
`run_id`	Run ID

`match_colnames` *Match names of local data frame to SQL table*

Description

Match names of local data frame to SQL table

Usage

```
match_colnames(values, table, con)
```

Arguments

`values`	‘data.frame‘ of values to write to SQL database
`table`	Name of target SQL table, as character
`con`	Database connection object

match_dbcols — *Match column names and classes between local and SQL table*

Description

Match column names and classes between local and SQL table

Usage

```
match_dbcols(values, table, con, coerce_col_class = TRUE, drop = TRUE)
```

Arguments

`values`	‘data.frame‘ of values to write to SQL database
`table`	Name of target SQL table, as character
`con`	Database connection object
`coerce_col_class`	logical, whether or not to coerce local data columns to SQL classes. Default = ‘TRUE.‘
`drop`	logical. If ‘TRUE‘ (default), drop columns not found in SQL table.

Value

‘values‘ ‘data.frame‘ with column names and classes matched to SQL

met_inputs — *Retrieve available met inputs for the given site, model, and hostname*

Description

This is identical to the query performed by ‘web/03-inputs.php‘ to populate the list of available sites. It may be useful for debugging, or otherwise replicating PEcAn web interface behavior.

Usage

```
met_inputs(dbcon, site_id, model_id, hostname)
```

Arguments

`dbcon`	Database connection object
`site_id`	Site ID (from ‘sites‘ table)
`model_id`	Model ID (from ‘models‘) table
`hostname`	Hostname of machine

Value

‘data.frame‘ of available met inputs

Author(s)

Alexey Shiklomanov

`ncdays2date` *Convert netcdf number of days to a datetime*

Description

Convert netcdf number of days to a datetime

Usage

```
ncdays2date(time, unit)
```

Arguments

`time`	number of time units elapsed since origin
`unit`	string containing CF-style time unit including origin (e.g. "days since 2010-01-01")

`pft.add.spp` *Associate species with a PFT.*

Description

adds a list of species to a pft based on USDA Plants acronyms

Usage

```
pft.add.spp(pft, acronym = NULL, ID = NULL, test = TRUE, con = NULL, ...)
```

Arguments

`pft`	String name of the PFT in the database
`acronym`	Specie’s Symbols. see `http://plants.usda.gov`
`ID`	Species IDs in Bety. You can provide either IDs or Symbols as input, if you provide both ID and acronym, only acronym will be used.
`test`	Runs the function in test mode. No species are actually added, but checks are run for existing species-PFT pairs, unmatched acronyms, missing species, or duplicate species
`con`	Database connection object.
`...`	optional arguements for connecting to database (e.g. password, user name, database)

Details

This function is used to add PFT-Species pairs to the database table 'pfts_species'. In the initial implementation the PFT has to be defined already and the species are added based on their USDA Symbol (genus/species acronym). Multiple species can be added at once but only one PFT at a time.

The Symbols object are

Value

Function does not return a value but does print out diagnostic statements.

Author(s)

Michael C. Dietze, Dongchen Zhang

`query.covariates` *Queries covariates from database for a given vector of trait id's*

Description

Queries covariates from database for a given vector of trait id's

Usage

```
query.covariates(trait.ids, con = NULL, ...)
```

Arguments

`trait.ids`	list of trait ids
`con`	database connection
`...`	extra arguments

Author(s)

David LeBauer

query.data — *Query data and transform stats to SE by calling* `fetch.stats2se`*;*

Description

Function to query data from database for specific species and convert stat to SE

Usage

```
query.data(
  trait,
  spstr,
  con,
  extra.columns = paste("ST_X(ST_CENTROID(sites.geometry)) AS lon,",
    "ST_Y(ST_CENTROID(sites.geometry)) AS lat, "),
  store.unconverted = FALSE,
  ids_are_cultivars = FALSE,
  ...
)
```

Arguments

`trait`	trait to query from the database
`spstr`	IDs of species to query from, as a single comma-separated string
`con`	database connection
`extra.columns`	other query terms to pass in. If unspecified, retrieves latitude and longitude
`store.unconverted`	determines whether or not a copy of the mean and stat fields are returned with _unconverted appended to the column names
`ids_are_cultivars`	if TRUE, ids is a vector of cultivar IDs, otherwise they are species IDs
`...`	extra arguments

Author(s)

David LeBauer, Carl Davidson

See Also

used in `query.trait.data`; `fetch.stats2se`; `transformstats` performs transformation calculations

query.file.path — *Get file path given id and machine*

Description

Get file path given id and machine

Usage

```
query.file.path(input.id, host_name, con)
```

Arguments

`input.id`	database id of the file ("container") to find
`host_name`	character: machine where the file lives
`con`	: database connection

Author(s)

Betsy Cowdery

query.format.vars — *Look up names and units of input variables from a format id or input id*

Description

Look up names and units of input variables from a format id or input id

Usage

```
query.format.vars(bety, input.id = NA, format.id = NA, var.ids = NA)
```

Arguments

`bety`	database connection
`input.id, format.id`	numeric. Defaults to format.id if both provided
`var.ids`	optional vector of variable IDs. If provided, limits results to these variables

Author(s)

Betsy Cowdery, Ankur Desai, Istem Fer

`query.pft_cultivars` *Select cultivars associated with a PFT*

Description

Given a PFT name and optionally a modeltype, finds its pft_id and returns the cultivars associated with it.

Usage

```
query.pft_cultivars(pft, modeltype = NULL, con)
```

Arguments

`pft`	string pft name
`modeltype`	type of model that is used, this is used to distinguish between different pfts with the same name.
`con`	database connection

Details

A PFT is allowed to have associated species or associated cultivars, but not both. If this function returns no results, try checking your PFT with `query.pft_species` instead. Note that the cultivars associated with one PFT *are* allowed to come from multiple species, if desired.

Value

tibble containing names and ids for each cultivar and the species it comes from

`query.pft_species` *Query species given pft name*

Description

select plant id's associated with pft

Usage

```
query.pft_species(pft, modeltype = NULL, con)
```

Arguments

`pft`	string pft name
`modeltype`	type of model that is used, this is used to distinguish between different pfts with the same name.
`con`	database connection

Value

data.frame containing id, genus, species, scientificname of each species associated with pft

Author(s)

David LeBauer

Examples

```
## Not run:
query.pft_species('ebifarm.pavi')
query.pft_species(settings = read.settings("pecan.xml"))

## End(Not run)
```

---

query.priors *Query Priors*

Description

Query priors associated with a plant functional type and a set of traits.

Usage

```
query.priors(pft, trstr = NULL, con = NULL, ...)
```

Arguments

`pft`	ID number of the PFT in the database
`trstr`	String of traits to query priors for. If passed as a character vector, it will be concatenated to a single string using [PEcAn.utils::vecpaste()].
`con`	Database connection object.
`...`	Optional arguments for connecting to database (e.g. password, user name, database).

Details

If neither ‘con‘ nor ‘...‘ are provided, this will try to connect to BETY using a ‘settings‘ object in the current environment.

Value

‘data.frame‘ of priors for each trait and the given PFT.

Author(s)

David LeBauer, Alexey Shiklomanov

Examples

```
## Not run:
  con <- db.open(...)
  query.priors("ebifarm.pavi", c("SLA", "Vcmax", "leaf_width"), con = con)

## End(Not run)
```

`query.site` *Given site_id, return site table*

Description

Given site_id, return site table

Usage

```
query.site(site.id, con)
```

Arguments

`site.id`	The id of the site
`con`	: database connection

Author(s)

Betsy Cowdery

`query.trait.data` *Extract trait data from database*

Description

Extracts data from database for a given trait and set of species, converts all statistics to summary statistics, and prepares a dataframe for use in meta-analysis. For Vcmax and SLA data, only data collected between April and July are queried, and only data collected from the top of the canopy (canopy height > 0.66). For Vcmax and root_respiration_rate, data are scaled converted from measurement temperature to 25^oC via the arrhenius equation.

Usage

```
query.trait.data(
  trait,
  spstr,
  con = NULL,
  update.check.only = FALSE,
  ids_are_cultivars = FALSE,
  ...
)
```

Arguments

`trait`	is the trait name used in the database, stored in variables.name
`spstr`	is the species.id integer or string of integers associated with the species
`con`	database connection object
`update.check.only`	if TRUE, returns results but does not print summaries
`ids_are_cultivars`	if TRUE, the IDs in spstr are cultivar IDs, otherwise they are species IDs. Passed on to `query.data`
`...`	unused currently

Value

dataframe ready for use in meta-analysis

Author(s)

David LeBauer, Carl Davidson, Shawn Serbin

Examples

```
## Not run:
settings <- read.settings()
query.trait.data("Vcmax", "938", con = con)

## End(Not run)
```

`query.traits` *Query trait data*

Description

Query available trait data associated with a given pft and a list of traits

Usage

```
query.traits(
  ids,
  priors,
  con,
  update.check.only = FALSE,
  ids_are_cultivars = FALSE
)
```

Arguments

`ids`	vector of species or cultivar id's from trait database
`priors`	vector of parameters for which priors have been specified
`con`	database connection object
`update.check.only`	if TRUE, returns results but does not print summaries
`ids_are_cultivars`	if TRUE, ids is a vector of cultivar IDs, otherwise they are species IDs

Value

list of dataframes, each with data for one trait

Author(s)

David LeBauer, Carl Davidson, Shawn Serbin

See Also

`query.trait.data`

Examples

```
## Not run:
con <- db.open(your_settings_here)
species <- query.pft_species('ebifarm.c4crop')
spstr <- vecpaste(species$id)
trvec <- c('leafN', 'SLA')
trait.data <- query.traits(spstr, trvec, con)

## End(Not run)
```

`query.yields` *Query yield data and transform stats to SE by calling* `fetch.stats2se`*;*

Description

Function to query yields data from database for specific species and convert stat to SE

Usage

```
query.yields(
  trait = "yield",
  spstr,
  extra.columns = "",
  con = NULL,
  ids_are_cultivars = FALSE,
  ...
)
```

Arguments

`trait`	yield trait to query
`spstr`	species to query for yield data
`extra.columns`	other query terms to pass in. Optional
`con`	database connection
`ids_are_cultivars`	if TRUE, spstr contains cultivar IDs, otherwise they are species IDs
`...`	extra arguments

Author(s)

<unknown>

See Also

used in `query.trait.data`; `fetch.stats2se`; `transformstats` performs transformation calculations

`query_pfts` *Retrieve PFT ID, name, and type from BETY*

Description

Retrieve PFT ID, name, and type from BETY

Usage

```
query_pfts(dbcon, pft_names, modeltype = NULL, strict = FALSE)
```

Arguments

`dbcon`	Database connection object
`pft_names`	character vector of PFT names
`modeltype`	character. If specified, only returns PFTs matching this modeltype. If NULL, considers all modeltypes.
`strict`	(Logical) If ‘TRUE‘, throw an error if any of the input ‘pft_names/ids‘ or ‘traits‘ are missing from the output. If ‘FALSE‘ (default), only throw a warning.

Value

‘data.frame‘ containing PFT ID (‘id‘), type (‘pft_type‘), and name (‘name‘).

Author(s)

Alexey Shiklomanov, Chris Black

query_priors *Query priors using prepared statements*

Description

Query priors using prepared statements

Usage

```
query_priors(
  pft_names = NULL,
  traits = NULL,
  pft_ids = NULL,
  expand = TRUE,
  strict = FALSE,
  ...
)
```

Arguments

`pft_names`	Character vector of PFT names (‘name‘ column of BETY ‘pfts‘ table). You cannot pass both this and ‘pft_ids‘.
`traits`	Character vector of trait names (‘name‘ column of BETY ‘traits‘ table). If ‘NULL‘ (default), return information for all traits available for that PFT.
`pft_ids`	Numeric vector of PFT IDs (‘id‘ column of BETY ‘pfts‘ table). You cannot pass both this and ‘pft_names‘.
`expand`	(Logical) If ‘TRUE‘ (default), search every trait-PFT combination. If ‘FALSE‘, assume that input traits and PFTs are paired.
`strict`	(Logical) If ‘TRUE‘, throw an error if any of the input ‘pft_names/ids‘ or ‘traits‘ are missing from the output. If ‘FALSE‘ (default), only throw a warning.
`...`	Additional arguments to [db.query()]

Value

‘data.frame‘ containing prior information for the given PFTs and traits.

Examples

```
## Not run:
  con <- db.open(...)

  # No trait provided, so return all available traits
  pdat <- query_priors(
    c("temperate.Early_Hardwood", "temperate.North_Mid_Hardwood",
      "temperate.Late_Hardwood"),
    con = con
  )

  # Traits provided, so restrict to only those traits. Note that
  # because `expand = TRUE`, this will search for these traits for
  # every PFT.
  pdat2 <- query_priors(
    c("Optics.Temperate_Early_Hardwood",
      "Optics.Temperate_Mid_Hardwood",
      "Optics.Temperate_Late_Hardwood"),
    c("leaf_reflect_vis", "leaf_reflect_nir"),
    con = con
  )

  # With `expand = FALSE`, search the first trait for the first PFT,
  # the second trait for the second PFT, etc. Note that this means
  # PFT and trait input vectors must be the same length.
  pdat2 <- query_priors(
    c("Optics.Temperate_Early_Hardwood",
      "Optics.Temperate_Early_Hardwood",
      "Optics.Temperate_Mid_Hardwood",
      "Optics.Temperate_Late_Hardwood"),
    c("leaf_reflect_vis",
      "leaf_reflect_nir",
      "leaf_reflect_vis",
      "leaf_reflect_nir"),
    con = con
  )

## End(Not run)
```

`runs` *Get table of runs corresponding to a workflow*

Description

Get table of runs corresponding to a workflow

Usage

```
runs(bety, workflow_id)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘
`workflow_id`	Workflow ID

`search_references` *Perform crossref search for a list of references*

Description

Requires the ‘rcrossref‘ package.

Usage

```
search_references(queries, ...)
```

Arguments

`queries`	Character vector of queries
`...`	Arguments passed on to `search_reference_single` `query` Citation string (length 1) to search for DOI `min_score` Minimum match score. Default (85) is fairly strict. `limit` Number of results to return

Value

‘data.frame‘ containing crossref information converted to match bety citations table.

`search_reference_single`
Perform crossref search for a single reference

Description

Requires the ‘rcrossref‘ package.

Usage

```
search_reference_single(query, limit = 1, min_score = 85)
```

Arguments

`query`	Citation string (length 1) to search for DOI
`limit`	Number of results to return
`min_score`	Minimum match score. Default (85) is fairly strict.

Value

'data.frame' containing crossref information converted to match bety citations table.

`stamp_started` *Stamp start and stop times of runs*

Description

Stamp start and stop times of runs

Usage

```
stamp_started(con, run)

stamp_finished(con, run)
```

Arguments

`con`	BETY database connection
`run`	(numeric) run ID

Value

'NULL'

`symmetric_setdiff` *Symmetric set difference of two data frames*

Description

Symmetric set difference of two data frames

Usage

```
symmetric_setdiff(
  x,
  y,
  xname = "x",
  yname = "y",
  namecol = "source",
  simplify_types = TRUE
)
```

Arguments

`x, y`	‘data.frame‘s to compare
`xname`	Label for data in x but not y. Default = "x"
`yname`	Label for data in y but not x. Default = "y"
`namecol`	Name of label column. Default = "source".
`simplify_types`	(Logical) If ‘TRUE‘, coerce anything that isn’t numeric to character, to facilitate comparison.

Value

‘data.frame‘ of data not common to x and y, with additional column (‘namecol‘) indicating whether data are only in x (‘xname‘) or y (‘yname‘)

Examples

```
xdf <- data.frame(a = c("a", "b", "c"),
                  b = c(1, 2, 3),
                  stringsAsFactors = FALSE)
ydf <- data.frame(a = c("a", "b", "d"),
                  b = c(1, 2.5, 3),
                  stringsAsFactors = FALSE)
symmetric_setdiff(xdf, ydf)
```

`take.samples` *Sample from normal distribution, given summary stats*

Description

sample from normal distribution, given summary stats

Usage

```
take.samples(summary, sample.size = 10^6)
```

Arguments

`summary`	data.frame with values of mean and sd
`sample.size`	number of samples to take

Value

sample of length sample.size

Author(s)

David LeBauer, Carl Davidson

Examples

```
## return the mean when stat = NA
take.samples(summary = data.frame(mean = 10, stat = NA))
## return vector of length \code{sample.size} from N(mean,stat)
take.samples(summary = data.frame(mean = 10, stat = 10), sample.size = 10)
```

`try2sqlite` *Convert TRY text file to SQLite database*

Description

The TRY file is huge and unnecessarily long, which makes it difficult to work with. The resulting SQLite database is much smaller on disk, and can be read much faster thanks to lazy evaluation.

Usage

```
try2sqlite(try_files, sqlite_file = "try.sqlite")
```

Arguments

`try_files`	Character vector of file names containing TRY data. Multiple files are combined with ‘data.table::rbindlist‘.
`sqlite_file`	Target SQLite database file name, as character.

Details

The resulting TRY SQLite database contains the following tables: - ‘values‘ – The actual TRY data. Links to all other tables through ID columns. - ‘traits‘ – Description of trait and data names. Links to ‘values‘ through ‘DataID‘. Similar to BETY ‘variables‘ table. - ‘datasets‘ – Description of datasets and references/citations. Links to ‘values‘ through ‘DatasetID‘ and ‘ReferenceID‘. - ‘species‘ – Species. Links to ‘values‘ through ‘AccSpeciesID‘.

`var_names_all` *Get vector of variable names for a particular workflow and run ID*

Description

Get vector of variable names for a particular workflow and run ID

Usage

```
var_names_all(bety, workflow_id, run_id)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘
`workflow_id`	Workflow ID
`run_id`	Run ID

`workflow` *Get single workflow by workflow_id*

Description

Get single workflow by workflow_id

Usage

```
workflow(bety, workflow_id)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘
`workflow_id`	Workflow ID

`workflows` *list of workflows that exist*

Description

list of workflows that exist

Usage

```
workflows(bety, ensemble = FALSE)
```

Arguments

`bety`	BETYdb connection, as opened by ‘betyConnect()‘
`ensemble`	Logical. Use workflows from ensembles table.

Index