Package ‘mfdb’

April 14, 2020

Type Package
Title MareFrame DB Querying Library
Encoding UTF-8
Version 6.2-0
Date 2020-04-10
Maintainer Jamie Lentin <lentinj@shuttlethread.com>
Description Creates and manages a PostgreSQL database suitable for storing fisheries data
License GPL-3
Depends R (>= 3.1.1)
Imports logging (>= 0.7-103), DBI (>= 0.3.1), getPass (>= 0.1-1),
       RPostgreSQL (>= 0.4)
Suggests unittest (>= 1.2), dplyr (>= 0.7.1), dbplyr (>= 1.0.0)
LazyData true
NeedsCompilation no
Author Jamie Lentin [aut, cre, cph],
       Bjarki Thor Elvarsson [aut]
Repository CRAN
Date/Publication 2020-04-14 11:00:10 UTC

R topics documented:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mfdb-package</td>
<td>2</td>
</tr>
<tr>
<td>ewe_model</td>
<td>5</td>
</tr>
<tr>
<td>gadget_areafile</td>
<td>6</td>
</tr>
<tr>
<td>gadget_directory</td>
<td>7</td>
</tr>
<tr>
<td>gadget_file</td>
<td>8</td>
</tr>
<tr>
<td>gadget_fleetfile</td>
<td>10</td>
</tr>
<tr>
<td>gadget_likelihood_component</td>
<td>12</td>
</tr>
<tr>
<td>gadget_stockfile</td>
<td>15</td>
</tr>
</tbody>
</table>
Description

Tools to query a MareFrame DB and reformat results in forms useful for GADGET and EwE models.

Introduction & Schema description

Before doing anything with mfdb, it is worth knowing a bit about how data is stored. Broadly, there are 2 basic types of table in mfdb, taxonomy and measurement tables.

The measurement tables store all forms of sample data supported, at the finest available detail. These are then aggregated when using any of the mfdb query functions. All measurement data is separated by case study, so multiple case studies can be loaded into a database without conflicts.

Taxonomy tables store all possible values for terms and their meaning, to ensure consistency in the data. For example, ‘species’ stores short-names and full latin names of all known species to MFDB, to ensure consistency in naming.

Most Taxonomies have defaults which are populated when the database is created, and their definitions are stored as data attached to this package. See mfdb-data for more information on these. Others, such as ‘areacell’ and ‘sampling_type’ are case study specific, and you will need to define your terms before you can import data.

Importing data

Unless you are working with a remote database, you will need to populate the database at least once before you are able to do any querying. The steps your script needs to do are:

Connect to database: Use the mfdb() function. This will create tables / populate taxonomies if necessary.
Define areas & divisions: mfdb models space in the following way:

**areacell**  The finest level of detail stored in the database. Every measurement (e.g. temperature, length sample) is assigned to an areacell. This will generally correspond to ICES gridcells, however there is no requirement to do so. You might augment gridcell information with depth, or include divisions when the measurement doesn’t correlate to a specific areacell.

**division**  Collections of areacells, e.g. ICES subdivisions, or whatever is appropriate.

Finally, when querying, divisions are grouped together into named collections, for instance `mfdb_group(north = 1:3, south = 4:6)` will put anything in divisions 1–3 under an area named "north", 4–5 under an area named "south".

Before you can upload any measurements, you have to define the areacells that they will use. You do this using the `mfdb_import_area()` function. This allows you to import tables of area/division information, such as:

```r
mfdb_import_area(mdb, data.frame(area = c('101', '102', '103', '401', '402', '403'), division = c('1', '1', '1', '4', '4', '4')))```

If you want areas to be part of multiple divisions, then you can use `mfdb_import_division()` to import extra revisions.

Define sampling types: Any survey data can have a sampling type defined, which then can be used when querying data. If you want to use a sampling type, then define it using `mfdb_import_sampling_type()`.

Import temperature data: At this point, you can start uploading actual measurements. The easiest of which is temperature. Upload a table of areacell/month/temperature data using `mfdb_import_temperature()`.

Import survey data: Finally, import any survey data using `mfdb_import_survey()`. Ideally upload your data in separate chunks. For example, if you have length and age-length data, don’t combine them in R, upload them separately and both will be used when querying for length data. This keeps the process simple, and allows you to swap out data as necessary.

Import stomach survey: Stomach surveys are imported in much the same way, however there are 2 data.frames, one representing predators, one preys. The column ‘stomach_name’ links the two, which can contain any numeric / character value, as long as it is unique for predators and prey measurements are assigned to the correct stomach.

See `mfdb_import_survey` for more information or the demo directory for concrete examples.

Dumping / Restoring a DB: You can also dump/import a dump from another host using the postgres `pg_dump` and `pg_restore` commands. You can dump/restore individual schemas (i.e. the case study you give to the `mfdb()` command), to list all the schemas installed run `SELECT DISTINCT(table_schema) FROM information_schema.tables from psql`. Note that if you use `mfdb('Baltic')`, the Postgres schema name will be lower-cased.

Create a dump of your chosen schema with the following command:

```sh
pg_dump --schema=baltic -Fc mf > baltic.dump
```

This will make a dump of the “baltic” case study into “baltic.tar”. It can then be restored onto another computer with the following:

```sh
pg_restore --clean -d mf baltic.dump
```

If you already have a baltic schema you wish to preserve, you can rename it first by issuing `ALTER SCHEMA baltic RENAME TO baltic_o` in psql. Once the restore is done you can rename the new schema and put the name of the old schema back.
Querying data

There are a selection of querying functions available, all of which work same way. You give a set of parameters, each of which can be a vector of data you wish returned, for instance year = 1998:2000 or species = c('COD').

If also grouping by this column (i.e. 'year', 'timestep', 'area' and any other columns given, e.g. 'age'), then the parameter will control how this grouping works, e.g. maturity_stage = mfdb_group(imm = 1, mat = 2:5) will result in the maturity_stage column having either 'imm' or 'mat'. These will also be used to generate GADGET aggregation files later.

For example, the following queries the temperature table:

```r
defaults <- list(
  area = mfdb_group("101" = ),
  timestep = mfdb_timestep_quarterly, # Group months to create 2 timesteps for each year
  year = 1996:2005)
agg_data <- mfdb_temperature(mdb, defaults)
```

All functions will result in a list of data.frame result tables (generally only one, unless you requested bootstrapping). Each are suitable for feeding into a gadget function to output into model files.

See mfdb_sample_count for more information or the demo directory for concrete examples.

Creating GADGET files

Finally, there are a set of functions that turn the output of queries into GADGET model files. These work on a gadget_directory object, which can either be an existing GADGET model to alter, or an empty / nonexistant directory.

Generally, the result of an mfdb query will be enough to create a corresponding GADGET file, for instance, the following will create a GADGET area file in your gadget directory:

```r
gadget_dir_write(gd,gadget_areafile(
  size = mfdb_area_size(mdb, defaults)[[1]],
  temperature = mfdb_temperature(mdb, defaults)[[1]])
```

See gadget_areafile or gadget_likelihood_component for more information or the demo directory for concrete examples.

Stock and fleet files: Stocks and fleets aren’t explicitly defined in the database. Instead, they are defined by querying on a column that differentiates them. For example, if your “immature cod” stock is defined as cod that is between maturity stages 1 and 2, then if querying for a stockdistribution component, one could do:

```r
mfdb_sample_count(mdb, c('maturity_stage', 'age', 'length'), list(
  species = 'COD',
  maturity_stage = c(imm = 1:2, mat = 3:5),
  ...
))
```

...and the maturity_stage column will be treated as the stock.
Acknowledgements

This project has received funding from the European Union’s Seventh Framework Programme for research, technological development and demonstration under grant agreement no.613571.

Author(s)

Jamie Lentin

Maintainer: Jamie Lentin <jamie.lentin@shuttlethread.com>

See Also

rgadget, Gadget user guide

Description

Transform the results of MFDB queries for use in an EwE model

Usage

ewe_stanza_group(survey_data)
ewe_stanzas(survey_data)
ewe_model(area_data, survey_data, catch_data = NULL)
ewe_diet(consumption_data)
ewe_pedigree(survey_data, catch_data = NULL)

Arguments

area_data Results of an mfdb_area_size query, aggregating the whole area
survey_data Results of an mfdb_sample_totalweight query, normally for one year, aggregated by the model’s functional groups
catch_data Results of an mfdb_sample_totalweight query, normally for one year, aggregated by the model’s functional groups and 'vessel'
consumption_data Results of an mfdb_stomach_preyweightratio query, aggregated by functional groups
Details

EwE requires stanzas and groups of stanzas, these are made up using the first and any other groupings in MFDB. For example, if survey_data was made with a query like \texttt{mfdb\_sample\_totalweight(mdb,c('species','age'),...)}, then the species will make up the generated stanza\_groups, and age will make up the stanzas within those groups.

\texttt{catch\_data} requires data that is also aggregated by vessel, this will be ignored for the purposes of deciding the stanza\_stanza\_group.

\texttt{consumption\_data} treats prey groupings separate to predator groupings, and all will be added to the diet matrix.

See \texttt{mfdb\_sample\_totalweight} for more information on how groupings can be used in queries.

Value

All return \texttt{data.frame} objects matching the EwE output, apart from diet which returns the diet matrix as a matrix.

Examples

# See demo/example-ewe.R for a full-length example

gadget_areafile

---
gadget_areafile | Gadget area files

Description

Structures representing a GADGET area file

Usage

gadget\_areafile(size, temperature, area = \texttt{attr(size, 'area')})

Arguments

- \texttt{size} \texttt{data.frame} as produced by \texttt{mfdb\_area\_size}
- \texttt{temperature} \texttt{data.frame} as produced by \texttt{mfdb\_temperature}
- \texttt{area} Optional. \texttt{mfdb\_group} that you used to specify area. By default pulls it from annotations on the size object.

Details

Once formed, you can then use \texttt{gadget\_dir\_write} to write this out to a GADGET areafile.

Value

List of class ‘gadget\_areafile’ that represents the area file contents.
## Not run: # NB: Requires a PostgreSQL installation, see README

# Any example could be added to a file with the following:--
mdb <- mfdb('examples')
area_group <- mfdb_group(
  divA = c("divA"),
  divB = c("divB"),
  divAB = c("divA", "divB")
)

# Create an areafile from 2 mfdb queries
areafile <- gadget_areafile(
  mfdb_area_size(mdb, list(
    area = area_group))[[1]],
  mfdb_temperature(mdb, list(
    year = 1998:2000,
    timestep = mfdb_timestep_quarterly,
    area = area_group))[[1]]
)

# Write this to a gadget_directory
gadget_dir_write(gadget_directory(tempfile()), areafile)

## End(Not run)

### gadget_directory

**Gadget directory objects**

**Description**

Structures representing a directory of data files

**Usage**

```r
gadget_directory(dir, mainfile = "main")
gadget_dir_write(gd, obj)
gadget_dir_read(gd, file_name, missing_okay = TRUE, file_type = c())
```

**Arguments**

- **dir**
  Name of directory, will be created if it doesn’t exist.
- **mainfile**
  Name of the GADGET mainfile to use.
- **gd**
  A gadget_directory object.
- **obj**
  The gadget_file, or gadget_lielihood_component to write.
- **file_name**
  File to read out of the directory and turn into a gadget_file.
- **missing_okay**
  If true, return an empty file instead of complaining that the given file does not exist.
file_type  A character vector that alters how the file is parsed. Currently either NULL or "bare_component", which implies we write "something" instead of "[something]".

Details

These functions handle reading and writing of files to a directory containing GADGET model files.

First a gadget_directory object needs to be created with gadget_directory, this ensures the directory exists and stores the name of the mainfile to use.

Any portion of a gadget model can then be written out with gadget_dir_write. You do not need to tell it which files in the model to update, since this is worked out based on what you are writing out.

Value

gadget_directory returns a list of class 'gadget_directory', containing the location of the mainfile that the gadget configuration will use.

gadget_dir_write returns NULL

gadget_dir_read returns a gadget_file object from read.gadget_file

Examples

# Create a gadget directory
gd <- gadget_directory(tempfile())

# Read in the likelihood file
likelihood <- gadget_dir_read(gd, 'likelihood')

# Write out an area file to "(tempfile)/areas", replacing any existing file
# Write out an area file to "(tempfile)/areas", replacing any existing file
gadget_dir_write(gd, gadget_file("areas", components = list(list(north = 1:3, south = 4:7))))

# Replace a likelihood component if one already exists with
# the same name/type or append it to the bottom
# Replace a likelihood component if one already exists with
# the same name/type or append it to the bottom
gadget_dir_write(gd, gadget_likelihood_component("understocking", name = "frank"))

---

gadget_file  *Gadget file objects*

Description

Structures representing an individual GADGET data file.
gadget_file

Usage

```r
gadget_file(file_name, components = list(), data = NULL, file_type = c())
## S3 method for class 'gadget_file'
print(x, ...)
## S3 method for class 'gadget_file'
as.character(x, ...)
read.gadget_file(file_name, file_type = c(), fileEncoding = "UTF-8")
```

Arguments

- `file_name`: Filename the output should be written to / read from
- `components`: A list of lists, representing each component. See details.
- `data`: A data.frame representing the tabular data at the end of a file.
- `file_type`: A character vector that alters how the file is parsed. Currently either NULL or "bare_component", which implies we write "something" instead of "[something]".
- `x`: gadget_file object
- `fileEncoding`: File's characterset. Defaults to UTF-8
- `...`: Unused

Details

For our purposes, a gadget file is broken down into components, where the first component is any key/value data at the top of the file. Each section separated by "[something]" is considered a new component. Each component is a list of key/values, where values can be vectors of multiple values. Also components can have comments prepended by adding a "preamble" attribute.

In slight deviation to GADGET spec, we insist that tabular data begins with "; – data –", to avoid any ambiguity on when it starts.

Value

- `gadget_file`: Returns a gadget_file object, a list of components.
- `print.gadget_file`: Prints the gadget file as it would be written to the filesystem.
- `as.character.gadget_file`: Returns a character string of the gadget file as it would be written to the filesystem.
- `read.gadget_file`: Returns a gadget_file object, a list of components.

Examples

```r
# Simple key/values
gadget_file("age", components = list(
  list(length = 5, age = 1:5))

# Multiple components
gadget_file("likelihood", components = list(
  list(),
```
gadget_fleetfile

component = structure(list(type = "penalty"), preamble = list("comment")),
component = structure(list(type = "penalty"), preamble = list("", "another comment"))

# Data
gadget_file("agelen", components = list(
    list(stocknames = "cod"), data = data.frame(
        area = c(102, 103),
        number = c(2345, 5023))
)

gadget_fleetfile  

---

gadget_fleetfile  

**Gadget fleet files**

**Description**

Structures representing fleet file components

**Usage**

gadget_fleet_component(type,
    name = type,
    livesonareas = unique(data$area),
    multiplicative = 1,
    suitability = NULL,
    fleetfile = 'fleet',
    data = stop("data not provided"),
    ...)

**Arguments**

- **type**: Required. Type of fleet component to create, e.g. 'totalfleet'
- **name**: Optional. A descriptive name for the fleet component, defaults to the type.
- **livesonareas**: Optional. Vector of area names, defaults to all unique areas in data.
- **multiplicative**: Optional. Defaults to 1
- **suitability**: Optional. Defaults to empty string
- **fleetfile**: Optional. The fleet file to put the component in. Defaults to 'fleet'.
- **data**: Required. The data.frame to use for 'amountfile'. Areas are translated into integers before adding to amountfile.
- **...**: Extra parameters for the component, see details

**Details**

effortfleet requires the following extra parameters:

catchability A list of stock names to catchability constants

quotafleet requires the following extra parameters:
**quotafunction** Function name, e.g. 'simple'

**biomasslevel** Vector of biomass levels

**quotalevel** Vector of fishing levels

**Value**

A gadget_fleet_component object that can then be added to a fleetfile with `gadget_dir_write`

**Examples**

```r
## Not run: # NB: Requires a PostgreSQL installation, see README

mdb <- mfdb('examples')
gd <- gadget_directory(tempfile())

# Make a 'totalfleet' component
fc <- gadget_fleet_component(
  'totalfleet',
  name = 'research',
  data = mfdb_sample_count(mdb, c(), list(
    vessel = '1.RSH',
    area = mfdb_group(x = 'divA', y = 'divB'),
    year = 1998,
    step = mfdb_timestep_yearly))[[1]])

fc

# Write out to a directory
gadget_dir_write(gd, fc)

gadget_fleet_component(
  'effortfleet',
  name = 'commercial',
  suitability = "function constant 4;",
  catchability = list(stockA=4, stockB=5),
  quotafunction = 'simple',
  biomasslevel = c(1000, 2000),
  quotalevel = c(0.1, 0.4, 0.9),
  data = mfdb_sample_count(mdb, c(), list(
    vessel = '2.COM',
    area = mfdb_group(x = 'divA', y = 'divB'),
    year = 1998,
    step = mfdb_timestep_yearly))[[1]])

gadget_fleet_component(
  'quotafleet',
  name = 'commercial',
  suitability = "function constant 4;",
  catchability = list(stockA=4, stockB=5),
  quotafunction = 'simple',
  biomasslevel = c(1000, 2000),
  quotalevel = c(0.1, 0.4, 0.9),
  data = mfdb_sample_count(mdb, c(), list(
```
gadget_likelihood_component

Gadget likelihood components

Description

Structures representing a component of a GADGET likelihood file.

Usage

gadget_likelihood_component(type, weight = 0, name = type, likelihoodfile = 'likelihood', ...)

Arguments

type Type of group to create. One of penalty, understocking, catchstatistics, catchdistribution, stockdistribution.

name A descriptive name for the component

weight A numeric weighting

likelihoodfile The likelihood file this component should end up in

... Extra parameters for the group. See details.

Details

In addition, penalty understands:

data A data.frame with 2 columns, "switch" and "power"

catchstatistics understands:

data_function The function Gadget should use, by default guesses based on the function that generated data

data A data.frame probably generated by mfdb_sample_meanlength_stddev

area An list of areas, taken from attr(data,"area") if not supplied

age An list of ages, taken from attr(data,"age") if not supplied

fleetnames List of fleet names

stocknames List of stock names

catchdistribution understands:

vessel = '2.COM',
area = mfdb_group(x = 'divA', y = 'divB'),
year = 1998,
step = mfdb_timestep_yearly))[[1]])

## End(Not run)
The function Gadget should use, by default uses sumofsquares

Extra parameters to supply to gadget, based on the function

TRUE or FALSE, defaults to FALSE

TRUE or FALSE, defaults to FALSE

Numeric, defaults to 10

A data.frame probably generated by mfdb_sample_meanlength_stddev

An list of areas, taken from attr(data,"area") if not supplied

An list of ages, taken from attr(data,"age") if not supplied

An list of lengths, taken from attr(data,"length") if not supplied

List of fleet names

List of stock names

The function Gadget should use, by default uses sumofsquares

TRUE or FALSE, defaults to FALSE

Numeric, defaults to 10

A data.frame probably generated by mfdb_sample_meanlength_stddev

An list of areas, taken from attr(data,"area") if not supplied

An list of ages, taken from attr(data,"age") if not supplied

An list of lengths, taken from attr(data,"length") if not supplied

List of fleet names

List of stock names

Fit options, see GADGET manual

A vector of length 2

A single suitability function

Numeric, defaults to 10

String, see GADGET manual

What data the component is based on, see GADGET manual

0 or 1, defaults to 0

A data.frame probably generated by mfdb_sample_meanlength_stddev
area An list of areas, taken from `attr(data,"area")` if not supplied
age An list of ages, taken from `attr(data,"age")` if not supplied
length An list of lengths, taken from `attr(data,"length")` if not supplied
fleetnames List of fleet names
stocknames List of stock names
surveynames List of acoustic survey names
fittype, slope, intercept Fit options, see GADGET manual
stomachcontent understands:
data_function Function GADGET will use
epsilon To be used when calculated probability is low
prey_labels Either a vector of stock names to be used for all preys, or a list to match preys, see below
prey_digestion_coefficients Optional. Either a vector of coefficients fo be used for all preys, or a list to match preys, see below
predator_names Vector of predator stock names
data A data.frame probably generated by `mfdb_sample_meanlength_stddev`

Both `prey_labels` and `prey_digestion_coefficients` allow you to match parts of prey labels and use repetition. For instance, `list("cod.mat" = "mature_cod","cod" = "cod","other")` will give "cod.mat" the label "mature_cod", "cod.imm" the label "cod", and anything else will get "other". You can also use regular expression syntax, for example "cod[0-9]":
migrationpenalty understands:
stockname Stock name
powercoeffs 2 power coefficients

Value
A `gadget_likelihood_component` object that can then be written to a likelihood file with `gadget_dir_write`

Examples

```r
# Create a penalty component
component <- gadget_likelihood_component("penalty",
    name = "bounds",
    weight = "0.5",
    data = data.frame(
        switch = c("default"),
        power = c(2),
        stringsAsFactors = FALSE))
component

# Create an understocking component
component <- gadget_likelihood_component("understocking", name ="understock")
component
```

# Any example could be added to a file with the following:-
gd <- gadget_directory(tempfile())
gadget_dir_write(gd, component)

---

gadget_stockfile  Gadget stock files

**Description**

Structures representing a GADGET stock file

**Usage**

- `gadget_stockfile_extremes(stock_name, data)`
- `gadget_stockfile_refweight(stock_name, data)`
- `gadget_stockfile_initialconditions(stock_name, data)`
- `gadget_stockfile_recruitment(stock_name, data)`

**Arguments**

- `stock_name` A name, e.g. cod.imm, used as the name for the stockfile
- `data` A data.frame used to generate the data. See details.

**Details**

The columns required in the data varies depends on which function you are using.

- `gadget_stockfile_extremes` requires age and length columns and populates minlength, minage, maxlength, maxage. The values are obtained by grouping used, rather than the maximum values in the data. If you want the minimum and maximum from the data, query with `length = NULL`, `age = NULL`, so the table contains "all" and the grouping contains the actual minimum and maximum.

- `gadget_stockfile_refweight` requires a length column and a mean column representing mean weight for that length group. It populates the refweightfile and dl.

- `gadget_stockfile_initialconditions` requires area, age, length, number and mean (weight) columns. Populates initialconditions minlength, minage, maxlength, maxage, dl and the numberfile. As before, the min/max values are populated using the groupings you specify, not the min/max available data.

- `gadget_stockfile_recruitment` requires year, step, area, age, length, number and mean (weight) columns. Populates doesrenew, minlength, maxlength, dl, numberfile.

**Value**

The return value is a gadget_stockfile object that can be written to the filesystem with gadget_dir_write.
Examples

```r
## Not run:  # NB: Requires a PostgreSQL installation, see README

mdb <- mfdb('examples')

imm_data <- mfdb_sample_meanweight(mdb, c('age', 'length'), list(
    age = NULL,  # The age column will say 'all', but will know the min/max
    length = mfdb_step_interval('', 10, to = 100),
    species = 'COD'))

# Write both min/max and refweightfile into our gadget directory
component <- gadget_stockfile_extremes('cod.imm', imm_data[[1]])
component

component <- gadget_stockfile_refweight('cod.imm', imm_data[[1]])
component

gadget_dir_write(gadget_directory(tempfile()), component)

## End(Not run)
```

---

### mfdb

**MareFrame DB class**

#### Description

Create a class representing a connection to a MareFrame DB

#### Usage

```r
mfdb(case_study_name,
    db_params = list(),
    destroy_schema = FALSE,
    check_db_available = FALSE,
    save_temp_tables = FALSE)

mfdb_disconnect(mdb)
```

#### Arguments

- **case_study_name**
  
The name of your case study. Your Postgres database can be used to store any number of case studies, by storing them in separate schemas. This parameter defines the schema to connect to, and can contain any lower case characters or underscore.

- **db_params**
  
  Extra parameters to supply to DBI::dbConnect. By default it will search for a "mf" database locally, but you can override any of the parameters, in particular host, dbname, user, password. See methods?RPostgreSQL::dbConnect for more information.
**destroy_schema**

Optional boolean. If true, all mfdb tables will be destroyed when connecting. This allows you to start populating your case study from scratch if required. The function will return NULL, you need to call mfdb again to connect, at which point the mfdb tables will be recreated and you can populate with data again.

**save_temp_tables**

Optional boolean. If true, any temporary tables will be made permanent for later inspection.

**check_db_available**

Optional boolean. If true, we will only check that the database connection could be made. Mostly for use in examples.

**mdb**

Database connection created by mfdb().

---

**Value**

A ’mfdb’ object representing the DB connection

**Examples**

```r
## Not run: # NB: Requires a PostgreSQL installation, see README

# Connect to local DB, as the "examples" case study
mdb <- mfdb('examples')

# Disconnect from database
mfdb_disconnect(mdb)

# Connect to remote server, will prompt for username/password
if (interactive()) {
  mdb <- mfdb('examples', db_params = list(host = "mfdb.rhi.hi.is"))
}

## End(Not run)
```

---

**MareFrame DB Datasets**

**Description**

Data sets representing the content of taxonomies used in the database.

**Usage**

```
case_study
gear
institute
market_category
maturity_stage
```
Details

All of these tables represent acceptable values for use when importing data. You can see the content of an individual taxonomy at the R command line, e.g. `mfdb::gear`

Each of the datasets will have the following columns.

- **id** A numeric ID to be used internally
- **name** An alphanumeric ID to be used when importing and reporting data.
- **description** Some text describing the option.
- **t_group** Groups together several items to query all in one go. E.g. for institutes you can query `’NOR’` to get all institutes in Norway.

The taxonomies are used in the following locations:

- **case_study** Possible case studies, use when connecting with `mfdb()`
- **gear, institute, vessel_type** Used to describe the dataset being imported with `mfdb_import_survey()`
- **sex, maturity_stage, species** Used for individual sample points when using `mfdb_import_survey()`

---

**mfdb_aggregate_group**  
*MareFrame DB groups*

Description

Represent a grouping of data to be applied when summarising area, timestep, age or length.

Usage

```r
# Named grouping of discrete items
mfdb_group(...)

# Pre-baked mfdb_groups for timesteps
mfdb_timestep_yearly
mfdb_timestep_biannually
mfdb_timestep_quarterly

# Grouping of discrete items, names generated by prefix
mfdb_group_numbered(prefix, ...)

# make (count) mfdb_groups, by sampling (count) times from (group)
mfdb_bootstrap_group(count, group, seed = NULL)
```
Arguments

... For mfdb_group, all named arguments are expected to be a named list of members for that group. For mfdb_group_numbered, the members do not have to be named, a name will be generated based on the prefix.

prefix When generating numeric group names, the character prefix to append to the beginning.

group For mfdb_bootstrap_group, the mfdb_group to do sampling with replacement from.

count For mfdb_bootstrap_group, how many times to sample each member of the given group.

seed For mfdb_bootstrap_group, if you want your groups to remain consistent across sessions, then specify a random integer as per RNG.

Value

An mfdb_aggregate object that can then be used in querying functions such as mfdb_sample_count

Examples

```r
## Aggregate age into 2 groups. "young" (for ages 1--3) and
## "old" (for ages 4--6)
g1 <- mfdb_group(young = c(1,2,3), old = c(4,5,6))

## Aggregate areas into "area1" and "area2".
g2 <- mfdb_group_numbered("area", c(1011,1012,1013), c(1021,1022))

## Take 3 samples with replacement from each group in area
g3 <- mfdb_bootstrap_group(3, g2)
```

mfdb_aggregate_interval

* MareFrame DB intervals *

Description

Represent a uniform or non-uniform interval.

Usage

```r
mfdb_interval(prefix, vect, open_ended = FALSE)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>(required) A character prefix to prepend to minimum to create list names</td>
</tr>
<tr>
<td>vect</td>
<td>(required) A vector representing the minimum for each group, and the maximum</td>
</tr>
<tr>
<td>open_ended</td>
<td>If TRUE / c('upper'), the last group will ignore its upper bound and include</td>
</tr>
<tr>
<td></td>
<td>any value. If c('lower'), the first group will ignore its lower bound include</td>
</tr>
<tr>
<td></td>
<td>everything &lt; the first value in vect. If c('upper', 'lower'), both the above occur.</td>
</tr>
<tr>
<td></td>
<td>This is useful when creating plus groups for GADGET, as GADGET will still</td>
</tr>
<tr>
<td></td>
<td>be presented a bounded group, but will contain all remaining data.</td>
</tr>
</tbody>
</table>

Value

An mfdb_aggregate object that can then be used in querying functions such as mfdb_sample_count

Examples

```r
## Make groups of len40 (40--60), len60 (60--80)
g1 <- mfdb_interval("len", c(40, 60, 80))

## Use seq to make life easier
# g2 <- mfdb_interval("len", seq(40, 80, by = 20))

## Create groups len40: [40, 60), len60: [60, inf) (but [60, 80) in the GADGET model)
g1 <- mfdb_interval("len", c(40, 60, 80), open_ended = c("upper"))
```

mfdb_aggregate_na_group

MareFrame DB aggregate NAs

Description

A decorator for other MFDB attributes to file NAs into another group, either one created by the main function or not.

Usage

```r
mfdb_na_group(sub_aggregate, na_group)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sub_aggregate</td>
<td>An mfdb_aggregate produced by another function, e.g. mfdb_step_interval</td>
</tr>
<tr>
<td>na_group</td>
<td>The group to assign NAs to, e.g. &quot;len_unknown&quot;</td>
</tr>
</tbody>
</table>

Details

The NA group won’t be added to any aggregate files generated by MFDB, since the output would be invalid.
**Value**

An `mfdb_aggregate` object that can then be used in querying functions such as `mfdb_sample_count`.

**Examples**

```r
length <- mfdb_na_group(mfdb_step_interval("len", 10), 'len_unknown')
```

---

**Description**

Groups data into uniform intervals.

**Usage**

```r
mfdb_step_interval(prefix, by, from = 0, to = NULL, open_ended = FALSE)
```

**Arguments**

- `prefix` (required) A character prefix to prepend to minimum to create list names.
- `by` (required) Increment of the sequence. NB: Must be an integer.
- `from, to` Start / end of the sequence. Defaults to 0 / infinity respectively.
- `open_ended` If TRUE / c('upper'), the last group will ignore it's upper bound and include any value. If c('lower'), the first group will ignore it's lower bound include everything < the first value in vect. If c('upper', 'lower'), both the above occur. This is useful when creating plus groups for GADGET, as GADGET will still be presented a bounded group, but will contain all remaining data.

**Value**

An `mfdb_aggregate` object that can then be used in querying functions such as `mfdb_sample_count`.

**Examples**

```r
## Make groups of len0 (0--5), len5 (5--10), ... len45(45--50)
g1 <- mfdb_step_interval("len", 5, to = 50)
## Make groups of len0 (0--5), len5 (5--10), ... len45(45--50), len50(50--inf)
g2 <- mfdb_step_interval("len", 5, to = 50, open_ended = TRUE)
```
mfdb_aggregate_unaggregated

*MareFrame DB unaggregated data*

Description

Tell mfdb functions not to aggregate this column, just return all values.

Usage

```r
mfdb_unaggregated(omitNA = FALSE, like = c(), not_like = c())
```

Arguments

- `omitNA`: Skip over rows where column is NA
- `like`: Vector of SQL like expressions to check column against
- `not_like`: Vector of SQL like expressions to check column against

Details

SQL like expressions can use the wildcards "_" to match any character and "

Value

An mfdb_aggregate object that can then be used in querying functions such as mfdb_sample_count

Examples

```r
# All vessels with a name ending with 'e' or 'd'
mfdb_unaggregated(like = c("%e", "%d"))
```

mfdb_bulk

*MareFrame DB Dump / Restore*

Description

Dump / Restore entire case studies.

Usage

```r
mfdb_cs_dump(mdb, out_location)
mfdb_cs_restore(mdb, in_location)
```
Arguments

mdb (required) A database connection created by mfdb()
in_location, out_location (required) A filesystem directory or '.tar.gz' file to dump / restore database contents.

Details

Deprecated: These commands aren’t strictly necessary any more. In most situations it will be easier to use Postgres’ pg_dump and pg_restore. See mfdb-package for some examples of how to do it. These functions don’t offer much more functionality and much slower.
mfdb_cs_dump copies all data from the database/case-study that mdb is connected to, and writes it out to files in out_location. If this ends with 'tar.gz', then all files will be put into a tarball with the name
mfdb_cs_restore will remove any case-study data, and replaces it with the content of in_location, a directory or tarball.

Value

NULL

Examples

## Not run:  # NB: Requires a PostgreSQL installation, see README

# Copy case study data from one case study to another
mdb_out <- mfdb('examples')
mdb_in <- mfdb('examples-copy')

dump_path <- tempfile(fileext='.tar.gz')
mfdb_cs_dump(mdb_out, dump_path)
mfdb_cs_restore(mdb_in, dump_path)

## End(Not run)

mfdb_dplyr MareFrame DB dplyr interface

Description

Use mfdb tables with dplyr

Usage

mfdb_dplyr_survey_index(mdb, include_cols = all_cols)
mfdb_dplyr_division(mdb, include_cols = all_cols)
mfdb_dplyr_sample(mdb, include_cols = all_cols)
mfdb_dplyr_predator(mdb, include_cols = all_cols)
mfdb_dplyr_prey(mdb, include_cols = all_cols)
Arguments

mdb An object created by mfdb()
include_cols Any additional columns to include in output, see details.

Details

Warning: Whilst these might be handy for exploration, there is no guarantee that code using these will continue to work from one version of MFDB to the next.
There is one function for each measurement table. By default every possible taxonomy column is included. However this is somewhat inefficient if you do not require the data, in which case specify the columns required with include_cols. See mfdb::mfdb_taxonomy_tables for possible values.

Value

A dplyr table object, for you to do as you please.

Examples

## Not run: # NB: Requires a PostgreSQL installation, see README

```
mdb <- mfdb('examples')

# Include as many columns as possible
mfdb_dplyr_sample(mdb)

# Only include 'data_source' and 'species' columns, as well as measurements
mfdb_dplyr_sample(mdb, c('data_source', 'species'))

## End(Not run)
```

mfdb_helpers MareFrame tools & helpers

Description

Misc. functions to aid working with an MFDB database.

Usage

```
# Find species from abbreviated names
mfdb_find_species(partial_name, single_matches_only = FALSE)
```

Arguments

partial_name Vector of partial species names, e.g. "Gad Mor", "gad. Mor.", "Gadus Mor", will all match "Cod (Gadus Morhua)".
single_matches_only Logical, default FALSE. If true, return NA for partial_names with multiple or zero matches.
Value

A matrix of all potential id, name & descriptions for each item in partial_name.

Examples

```r
mfdb_find_species(c("gad mor","tube worms"))
#  id       name        description
#  1e+10 "COD"   "Cod (Gadus Morhua)"
#  1e+10 "TBX"   "Tube Worms (Tubeworms)"
```

# Can also generate a map to help insert a data.frame of foreign data
```r
stomachs <- read.csv(text =
  A,Palaemon Elegans,1,1,10,5
  A,Palaemon Elegans,1,4,40,1
  B,Palaemon Elegans,1,1,10,5
  B,Palaemon Elegans,4,1,10,5
  B,Palaemon Elegans,5,1,10,NA
  B,Palaemon Elegans,5,1,10,NA
  C,Crangon Crangon,2,3.5,9.5,3
  D,Palaemon Elegans,1,1.4,10,1
  D,Crangon Crangon,5,4,40,1
  E,Worms,1,1.4,10,1
', stringsAsFactors = TRUE)
```

# Work out a map from all Prey_Species_Name values to MFDB species codes
```r
species_map <- mfdb_find_species(levels(stomachs$species), single_matches_only = TRUE)['name',]
```

# Put the new levels back onto the species column
```r
levels(stomachs$species) <- unlist(species_map)
```

```
mfdb_helpers_mfdb_concatenate_results

MareFrame Query Utilities

Description

Aggregate data from the database in a variety of ways

Usage

```
mfdb_concatenate_results(...)
```

Arguments

... Any number of data.frames produced by mfdb query functions with identical columns, e.g. mfdb_sample_count
Value

Given any number of data.frames from mfdb query functions with identical columns, produces a combined data.frame, similar to rbind but preserving the attributes required to produce aggregation files.

Description

Functions to import data into MareFrame DB

Usage

mfdb_import_temperature(mdb, data_in)
mfdb_import_survey(mdb, data_in, data_source = 'default_sample')
mfdb_import_survey_index(mdb, data_in, data_source = 'default_index')
mfdb_import_stomach(mdb, predator_data, prey_data, data_source = "default_stomach")

Arguments

mdb Database connection created by mfdb().
data_in, predator_data, prey_data A data.frame of survey data to import, see details.
data_source A name for this data, e.g. the filename it came from. Used so you can replace it later without disturbing other data.

Details

All functions will replace existing data in the case study with new data, unless you specify a data_source, in which case then only existing data with the same data_source will be replaced.

If you want to remove the data, import empty data.frames with the same data_source.
mfdb_import_temperature imports temperature time-series data for areacells. The data_in should be a data.frame with the following columns:

id A numeric ID for this areacell (will be combined with the case study number internally)
year Required. Year each sample was taken, e.g. c(2000, 2001)
month Required. Month (1–12) each sample was taken, e.g. c(1, 12)
areacell Required. Areacell sample was taken within
temperature The temperature at given location/time

mfdb_import_survey imports institution surveys and commercial sampling for your case study. The data_in should be a data.frame with the following columns:

institute Optional. An institute name, see mfdb::institute for possible values
gear Optional. Gear name, see mfdb::gear for possible values
vessel Optional. Vessel defined previously with mfdb_import_vessel_taxonomy(...)
tow Optional. Tow defined previously with mfdb_import_tow_taxonomy(...)sampling_type Optional. A sampling_type, see mfdb::sampling_type for possible valuesyear Required. Year each sample was taken, e.g. c(2000, 2001)month Required. Month (1–12) each sample was taken, e.g. c(1, 12)areacell Required. Areacell sample was taken within
species Optional, default c(NA). Species of sample, see mfdb::species for possible valuesage Optional, default c(NA). Age of sample, or mean age sex Optional, default c(NA). Sex of sample, see mfdb::sex for possible valueslength Optional, default c(NA). Length of sample / mean length of all sampleslength_var Optional, default c(NA). Sample variance, if data is already aggregatedlength_min Optional, default c(NA). Minimum theoretical length, if data is already aggregatedweight Optional, default c(NA). Weight of sample / mean weight of all samplesweight_var Optional, default c(NA). Sample variance, if data is already aggregatedweight_total Optional, default c(NA). Total weight of all samples, can be used with count = NA to represent an unknown number of samplescount Optional, default c(1). Number of samples this row represents (i.e. if the data is aggregated)
mfdb_import_survey_index adds indices that can be used as abundance information, for example. Before using mfdb_import_survey_index, make sure that the index_type you intend to use exists by using mfdb_import_cs_taxonomy. The data_in should be a data.frame with the following columns:
index_type Required. the name of the index data you are storing, e.g. 'acoustic'
year Required. Year each sample was taken, e.g. c(2000, 2001)month Required. Month (1–12) each sample was taken, e.g. c(1, 12)areacell Required. Areacell sample was taken withinvalue Value of the index at this point in space/time
mfdb_import_stomach imports data on predators and prey. The predator and prey data are stored separately, however they should be linked by the stomach_name column. If a prey has a stomach name that doesn’t match a predator, then an error will be returned.
The predator_data should be a data.frame with the following columns:
stomach_name Required. An arbitrary name that provides a link between the predator and prey tables
institute Optional. An institute name, see mfdb::institute for possible valuesgear Optional. Gear name, see mfdb::gear for possible valuesvessel Optional. Vessel defined previously with mfdb_import_vessel_taxonomy(mdb,...)tow Optional. Tow defined previously with mfdb_import_tow_taxonomy(....)
### mfdb_import_data

**sampling_type**  Optional. A sampling_type, see mfdb::sampling_type for possible values

**year**  Required. Year each sample was taken, e.g. c(2000, 2001)

**month**  Required. Month (1–12) each sample was taken, e.g. c(1, 12)

**areacell**  Required. Areacell sample was taken within

**species**  Optional, default c(NA). Species of sample, see mfdb::species for possible values

**age**  Optional, default c(NA). Age of sample, or mean age

**sex**  Optional, default c(NA). Sex of sample, see mfdb::sex for possible values

**maturity_stage**  Optional, default c(NA). Maturity stage of sample, see mfdb::maturity_stage for possible values

**stomach_state**  Optional, default c(NA). Stomach state of sample, see mfdb::stomach_state for possible values

**length**  Optional, default c(NA). Length of sample

**weight**  Optional, default c(NA). Weight of sample

The `prey_data` should be a data.frame with the following columns:

**stomach_name**  Required. The stomach name of the predator this was found in

**species**  Optional, default c(NA). Species of sample, see mfdb::species for possible values

**digestion_stage**  Optional, default c(NA). Stage of digestion of the sample, see mfdb::digestion_stage for possible values

**length**  Optional, default c(NA). Length of sample / mean length of all samples

**weight**  Optional, default c(NA). Weight of sample / mean weight of all samples

**weight_total**  Optional, default c(NA). Total weight of all samples

**count**  Optional, default c(NA). Number of samples this row represents (i.e. if the data is aggregated), count = NA represents an unknown number of samples

### Value

NULL

### Examples

```r
# Not run:  # NB: Requires a PostgreSQL installation, see README

mdb <- mfdb('examples-import-data')

# We need to set-up vocabularies first
mfdb_import_area(mdb, data.frame(
    id = c(1,2,3),
    name = c('35F1', '35F2', '35F3'),
    size = c(5)))

mfdb_import_vessel_taxonomy(mdb, data.frame(
    name = c('1.RSH', '2.COM'),
    stringsAsFactors = FALSE))

mfdb_import_sampling_type(mdb, data.frame(
    name = c("RES", "LND"),
    ...))
```

---

The `prey_data` should be a data.frame with the following columns:

**stomach_name**  Required. The stomach name of the predator this was found in

**species**  Optional, default c(NA). Species of sample, see mfdb::species for possible values

**digestion_stage**  Optional, default c(NA). Stage of digestion of the sample, see mfdb::digestion_stage for possible values

**length**  Optional, default c(NA). Length of sample / mean length of all samples

**weight**  Optional, default c(NA). Weight of sample / mean weight of all samples

**weight_total**  Optional, default c(NA). Total weight of all samples

**count**  Optional, default c(NA). Number of samples this row represents (i.e. if the data is aggregated), count = NA represents an unknown number of samples

### Value

NULL

### Examples

```r
# Not run:  # NB: Requires a PostgreSQL installation, see README

mdb <- mfdb('examples-import-data')

# We need to set-up vocabularies first
mfdb_import_area(mdb, data.frame(
    id = c(1,2,3),
    name = c('35F1', '35F2', '35F3'),
    size = c(5)))

mfdb_import_vessel_taxonomy(mdb, data.frame(
    name = c('1.RSH', '2.COM'),
    stringsAsFactors = FALSE))

mfdb_import_sampling_type(mdb, data.frame(
    name = c("RES", "LND"),
    ...))
```
mfdb_import_taxonomy

MareFrame Taxonomy import functions

Description

Functions to import taxonomy data into MareFrame DB

Usage

mfdb_import_area(mdb, data_in)
mfdb_import_division(mdb, data_in)
mfdb_import_sampling_type(mdb, data_in)
mfdb_import_tow_taxonomy(mdb, data_in)
mfdb_import_vessel_taxonomy(mdb, data_in)

mfdb_empty_taxonomy(mdb, taxonomy_name)

mfdb_import_cs_taxonomy(mdb, taxonomy_name, data_in)

Arguments

mdb Database connection created by mfdb().
taxonomy_name The name of the taxonomy to import, if there isn’t a special function for it. See mfdb:::mfdb_taxonomy_tables for possible values.
data_in A data.frame of data to import, see details.
Details

MFDB taxonomies define the values you can use when importing / querying for data. They need to be populated with the values you need before data is imported. Most taxonomies are pre-populated by the MFDB package, so you should use the predefined values. Others however this does not make sense, so should be done separately. This is what these functions do. 

mfdb_import_division is a special case, which imports groupings of areacells into divisions, if you haven’t already done this with mfdb_import_area or your divisions are too complicated to represent this way. The data_in should be a list of areacell vectors, with division names. For example, `list(divA = c('45G01', '45G02', '45G03'))` 

Beyond this, all functions accept the following columns: 

id Optional. A numeric ID to use internally, defaults to 1..n 
name Required. A vector of short names to use in data, e.g. "SEA" 
t_group Optional. A vector of the that groups together a set of values

Note that the database doesn’t use your short names internally. This means you can rename items by changing what name is set to. t_group allows taxonomy values to be grouped together. For example, giving all vessels in a fleet the same t_group you can then query the entire fleet as well as individually.

mfdb_import_area imports areacell information for your case study. Beyond the above, you can also provide the following:

size The size of the areacell 
division The name of the division this areacell is part of

mfdb_import_vessel_taxonomy imports names of vessels into the taxonomy table, so they can be used when importing samples. As well as the above, you can also specify:

full_name Optional. The full name of this vessel 
length Optional. The length of the vessel in meters 
power Optional. The vessel’s engine power in KW 
tonnage Optional. The vessel’s gross tonnage

mfdb_import_vessel_taxonomy imports names of vessels into the taxonomy table, so they can be used when importing samples. As well as the above, you can also specify:

latitude Optional. 
longitude Optional. 
depth Optional. Depth in meters 
length Optional. Length in meters

mfdb_import_sampling_type imports sampling types so that you can then use these against records in the sample table. You can also provide:

description Optional. A vector of descriptive names, e.g. "sea sampling"
mfdb_empty_taxonomy allows you to empty out a taxonomy of previous data. The import functions insert or update values that already exist, based on the numeric ID for these values. They do not delete anything, as it may be impossible to remove rows without destroying existing data.

However, if e.g. you want to replace the species taxonomy with an entirely different one you will need to flush it first, before you import any data. Use this function, then mfdb_import_species_taxonomy to import the new taxonomy.

NB: This won’t be possible if there is some data already using any of the terms. It is best used before your database is populated.

Value

NULL

---

mfdb_queries  MareFrame DB queries

Description

Aggregate data from the database in a variety of ways

Usage

mfdb_area_size(mdb, params)
mfdb_temperature(mdb, params)
mfdb_survey_index_mean(mdb, cols, params)
mfdb_survey_index_total(mdb, cols, params)
mfdb_sample_count(mdb, cols, params, scale_index = NULL)
mfdb_sample_meanlength(mdb, cols, params, scale_index = NULL)
mfdb_sample_meanlength_stddev(mdb, cols, params, scale_index = NULL)
mfdb_sample_totalweight(mdb, cols, params)
mfdb_sample_meanweight(mdb, cols, params, scale_index = NULL)
mfdb_sample_meanweight_stddev(mdb, cols, params, scale_index = NULL)
mfdb_sample_rawdata(mdb, cols, params, scale_index = NULL)
mfdb_sample_scaled(mdb, cols, params, abundance_scale = NULL, scale = 'tow_length')
mfdb_stomach_preycount(mdb, cols, params)
mfdb_stomach_preymeanlength(mdb, cols, params)
mfdb_stomach_preymeanweight(mdb, cols, params)
mfdb_stomach_preyweightratio(mdb, cols, params)
mfdb_stomach_presenceratio(mdb, cols, params)

Arguments

mdb        An object created by mfdb()
cols       Any additional columns to group by, see details.
params     A list of parameters, see details.
scale_index  Optional. survey_index used to scale results before aggregation, either "tow_length" or from mfdb_import_survey_index
abundance_scale  Optional. Same as scale_index
scale  Optional. A scale to apply to the resulting values, e.g. 'tow_length'

Details

The items in the params list either restrict data that is returned, or groups data if they are also in the cols vector, or are 'year', 'timestep', or 'area'.

If you are grouping by the column, params should contain one of the following:

NULL  Don’t do any grouping, instead put ‘all’ in the resulting column. For example, age = NULL results in "all".
character / numeric vector  Aggregate all samples together where they match. For example, year = 1990:2000 results in 1990, ..., 2000.
mfdb_unaggregated()  Don’t do any aggregation for this column, return all possible values.
mfdb_group()  Group several discrete items together. For example, age = mfdb_group(young = 1:3, old = 4:5) results in "young" and "old".
mfdb_interval()  Group irregular ranges together. For example, length = mfdb_interval('len', c(0, 10, 100, 1000)) results in "len0", "len10", "len100" (1000 is the upper bound to len100).
mfdb_step_interval()  Group regular ranges together. For example, length = mfdb_step_interval('len', to = 100, by = 10) results in "len0", "len10", ..., "len90".

In addition, params can contain other arguments to purely restrict the data that is returned.

institute  A vector of institute names / countries, see mfdb::institute for possible values
gear  A vector of gear names, see mfdb::gear for possible values
vessel  A vector of vessel names, see mfdb::vessel for possible values
sampling_type  A vector of sampling_type names, see mfdb::sampling_type for possible values
species  A vector of species names, see mfdb::species for possible values
sex  A vector of sex names, see mfdb::sex for possible values

To save specifying the same items repeatedly, you can use list concatenation to keep some defaults, for example:

defaults <- list(year = 1998:2000)
mfdb_sample_meannlength(mdb, c('age'), c(list(), defaults))

scale_index allows you to scale samples before aggregation. If it contains the name of a survey index (see mfdb_import_survey_index), then any counts will be scaled by the value for that areacell before and used in aggregation / weighted averages. As a special case, you can use "tow_length" to to scale counts by the tow length.
**Value**

All will return a list of data.frame objects. If there was no bootstrapping requested, there will be only one. Otherwise, there will be one for each sample.

The columns of these data frames depends on the function called.

- **mfdb_area_size** Returns area, (total area) size
- **mfdb_temperature** Returns year, step, area, (mean) temperature
- **mfdb_survey_index_mean** Returns year, step, area, (group cols), (mean) survey index
- **mfdb_survey_index_total** Returns year, step, area, (group cols), (sum) survey index
- **mfdb_sample_count** Returns year, step, area, (group cols), number (i.e sum of count)
- **mfdb_sample_meanlength** Return year, step, area, (group cols), number (i.e sum of count), mean (length)
- **mfdb_sample_meanlength_stddev** As mfdb_sample_meanlength, but also returns std. deviation.
- **mfdb_sample_totalweight** Returns year, step, area, (group cols), total (weight of group)
- **mfdb_sample_meanweight** Returns year, step, area, (group cols), number (i.e sum of count), mean (weight)
- **mfdb_sample_meanweight_stddev** As mfdb_sample_meanweight, but also returns std. deviation.
- **mfdb_sample_rawdata** Returns year, step, area, (group cols), number of samples, raw_weight and raw_length.
  
  NB: No grouping of results is performed, instead all matching table entries are returned
- **mfdb_sample_scaled** Returns year, step, area, (group cols), number (i.e. sum of count, scaled by tow_length), mean_weight (scaled by tow_length)
- **mfdb_stomach_preycount** Returns year, step, area, (group cols), number (of prey found in stomach)
- **mfdb_stomach_preymeanlength** Returns year, step, area, (group cols), number (of prey found in stomach), mean_length (of prey found in stomach). NB: Entries where count is NA (i.e. totals) are ignored with this function.
- **mfdb_stomach_preymeanweight** Returns year, step, area, (group cols), number (of unique stomachs in group), mean_weight (per unique stomach).
- **mfdb_stomach_preyweightratio** Returns year, step, area, (group cols), ratio (of selected prey in stomach to all prey by weight)
- **mfdb_stomach_presenceratio** Returns year, step, area, (group cols), ratio (of selected prey in stomach to all prey by count)

**Examples**

```r
# Not run: # NB: Requires a PostgreSQL installation, see README

mdb <- mfdb('examples')

# Query numbers by age and length
agg_data <- mfdb_sample_count(mdb, c('age', 'length'), list(  
  length = mfdb_interval("len", seq(0, 500, by = 30)),
```

```
```r
age = mfdb_group('young' = c(1,2), old = 3),
year = c(1998))

agg_data

# Use in a catchdistribution likelihood component
gadget_dir_write(gadget_directory(tempfile()), gadget_likelihood_component("catchdistribution",
    name = "cdist",
    weight = 0.9,
    data = agg_data[[1]],
    area = attr(agg_data[[1]], "area"),
    age = attr(agg_data[[1]], "age"))

## End(Not run)
```

### mfdb_sharing  
* MareFrame DB sharing options*

**Description**

Alter database privileges

**Usage**

```r
mfdb_share_with(mdb, user_or_role, query = TRUE, import = FALSE)
```

**Arguments**

- **mdb** (required) A database connection created by `mfdb()`
- **user_or_role** (required) Another database user, or a role, or 'public' to share with all users
- **query** Should the user be able to query the current case study?
- **import** Should the user be able to import more data current case study?

**Details**

This allows you to share case study data between users. This is most useful when using a shared database. Only the owner of the schema (i.e. the user that created it) will be able to change table structure (i.e. upgrade MFDB versions).

By default nothing is shared between users.

**Value**

NULL
Examples

## Not run: # NB: Requires a PostgreSQL installation, and creation of extra users

```r
mdb <- mfdb('examples')

mfdb_share_with(mdb, 'gelda') # Allow DB user gelda to query the 'examples' case study data
```

## End(Not run)
Index

as.character.gadget_file (gadget_file), 8

case_study (mfdb-data), 17
digestion_stage (mfdb-data), 17
ewe_diet (ewe_model), 5
ewe_model, 5
ewe_pedigree (ewe_model), 5
ewe_stanza_group (ewe_model), 5
ewe_stanzas (ewe_model), 5
gadget_areafile, 4, 6
gadget_dir_read (gadget_directory), 7
gadget_dir_write (gadget_directory), 7
gadget_directory, 4, 7
gadget_fleet_component
gadget_fleetfile, 10
gadget_fleetfile (gadget_fleetfile), 10
gadget_file, 8
gadget_fleetfile, 10
gadget_likelihood_component, 4, 12
gadget_stockfile, 15
gadget_stockfile_extremes
gadget_stockfile_initialconditions
gadget_stockfile_recruitment
gadget_stockfile_refweight
gear (mfdb-data), 17

ingstitute (mfdb-data), 17
market_category (mfdb-data), 17
maturity_stage (mfdb-data), 17
mfdb, 2, 16, 18
mfdb-data, 2, 17
mfdb-package, 2, 23
mfdb_aggregate_group, 18

mfdb_aggregate_interval, 19
mfdb_aggregate_na_group, 20
mfdb_aggregate_step_interval, 21
mfdb_aggregate_unaggregated, 22
mfdb_area_size (mfdb_queries), 31
mfdb_bootstrap_group

mfdb_aggregate_group, 18

mfdb_bulk, 22
mfdb_concatenate_results

mfdb_helpers_mfdb_concatenate_results, 25

mfdb_cs_dump (mfdb_bulk), 22
mfdb_cs_restore (mfdb_bulk), 22
mfdb_disconnect (mfdb), 16

mfdb_dplyr, 23
mfdb_dplyr_division (mfdb_dplyr), 23
mfdb_dplyr_prey (mfdb_dplyr), 23
mfdb_dplyr_sample (mfdb_dplyr), 23
mfdb_dplyr_survey_index (mfdb_dplyr), 23
mfdb_empty_taxonomy

mfdb_import_taxonomy, 29
mfdb_find_species (mfdb_helpers), 24
mfdb_group (mfdb_aggregate_group), 18
mfdb_group_numbered

mfdb_aggregate_group, 18

mfdb_helpers, 24
mfdb_helpers_mfdb_concatenate_results, 25

mfdb_import_area, 3
mfdb_import_area

mfdb_import_cs_taxonomy, 27
mfdb_import_cs_taxonomy

mfdb_import_data, 26
mfdb_import_division, 3
mfdb_import_division

mfdb_import_taxonomy, 29
mfdb_import_gear_taxonomy
  (mfdb_import_taxonomy), 29
mfdb_import_sampling_type
  (mfdb_import_sampling_type), 3
mfdb_import_species_taxonomy
  (mfdb_import_species_taxonomy), 29
mfdb_import_stomach (mfdb_import_data), 26
mfdb_import_survey, 3, 18
mfdb_import_survey (mfdb_import_data), 26
mfdb_import_survey_index, 32
mfdb_import_survey_index
  (mfdb_import_data), 26
mfdb_import_taxonomy, 29
mfdb_import_temperature
  (mfdb_import_data), 26
mfdb_import_tow_taxonomy
  (mfdb_import_taxonomy), 29
mfdb_import_vessel_taxonomy
  (mfdb_import_taxonomy), 29
mfdb_interval
  (mfdb_aggregate_interval), 19
mfdb_na_group
  (mfdb_aggregate_na_group), 20
mfdb_queries, 31
mfdb_sample_count
  (mfdb_sample_count), 4
mfdb_sample_count (mfdb_queries), 31
mfdb_sample_meanlength (mfdb_queries), 31
mfdb_sample_meanlength_stddev
  (mfdb_queries), 31
mfdb_sample_meanweight (mfdb_queries), 31
mfdb_sample_meanweight_stddev
  (mfdb_queries), 31
mfdb_sample_rawdata (mfdb_queries), 31
mfdb_sample_scaled (mfdb_queries), 31
mfdb_sample_totalweight
  (mfdb_sample_totalweight), 6
mfdb_sample_totalweight (mfdb_queries), 31
mfdb_share_with (mfdb_sharing), 34
mfdb_sharing, 34
mfdb_step_interval
  (mfdb_aggregate_step_interval), 21
mfdb_stomach_presenceratio
  (mfdb_queries), 31
mfdb_stomach_preycount (mfdb_queries), 31
mfdb_stomach_preymeanlength
  (mfdb_queries), 31
mfdb_stomach_preymeanweight
  (mfdb_queries), 31
mfdb_stomach_preyweightratio
  (mfdb_queries), 31
mfdb_survey_index_mean (mfdb_queries), 31
mfdb_survey_index_total (mfdb_queries), 31
mfdb_temperature (mfdb_queries), 31
mfdb_timestep_biannually
  (mfdb_aggregate_group), 18
mfdb_timestep_quarterly
  (mfdb_aggregate_group), 18
mfdb_timestep_yearly
  (mfdb_aggregate_group), 18
mfdb_unaggregated
  (mfdb_aggregate_unaggregated), 22
print.gadget_file (gadget_file), 8
read.gadget_file (gadget_file), 8
sex (mfdb-data), 17
species (mfdb-data), 17
stomach_state (mfdb-data), 17
vessel_type (mfdb-data), 17