Package: framrsquared (via r-universe)

March 3, 2025

Type Package

Title FRAM Database Interface

Version 0.5.0

Description A convenient tool for interfacing with FRAM access databases in R environments.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports cli, DBI, dplyr, framrosetta (>= 0.1.0), ggplot2, glue, here, janitor, lifecycle, lubridate, odbc, purrr, readr, renv, rlang, RSQLite, scales, stats, stringr, tibble, tidyr, tools

RoxygenNote 7.3.2

Depends R (>= 4.1)

Suggests knitr, testthat (>= 3.0.0), rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Remotes FRAMverse/framrosetta

URL https://framverse.github.io/framrsquared/

Config/pak/sysreqs make libicu-dev unixodbc-dev libx11-dev

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

RemoteUrl https://github.com/FRAMverse/framrsquared

RemoteRef HEAD

RemoteSha 42a490eb18132f14b58e8c40187f5eb9cce11b09

Contents

`addstock_check` *Check FRAM database after adding new stock*

Description

Either provides the step by step process of adding new stock to a FRAM database, or walks through fram database run and checks the tables for potential errors associated with adding new stock.

Usage

```
addstock_check(
  file_name = NULL,
  run_id = NULL,
  old_stockcount = 78,
  override_db_checks = FALSE
)
```

Arguments

file_name	filepath to database. If NULL, provide summary of process instead. Default = NULL.
run_id	RunID associated with the new stock in the FRAM database. IF left NULL, provide summary of process instead. Default = NULL.
old_stockcount	The number of stocks previously present to treat as the "baseline" – several checking steps will focus solely on newly added stocks. Defaults to 78.
override_db_checks	Ignore species, database type. When FALSE, function will stop if the database is not Chinook or if it's a transfer file. Defaults to FALSE.

Examples

```
## Not run:
## review process
addstock_check()
## check database for additional stock
addstock_check("2024 Pre-Season Chinook DB - first test.mdb",
run_id = 138)

## End(Not run)
```

---

add_flag_text *Adds a column with a text version of flags for either non-retention or fishery scalers*

Description

Adds a column with a text version of flags for either non-retention or fishery scalers

Usage

```
add_flag_text(.data)
```

Arguments

.data	fetched FisheryScalers or NonRetentions

Examples

```
## Not run:  mortality_table |> add_flag_text()
```

`aeq_mortality` *Extract AEQ mortality from Chinook FRAM database. Refactored and stripped down from the framr package written by Dan Auerbach. https://github.com/FRAMverse/framr/*

Description

Extract AEQ mortality from Chinook FRAM database. Refactored and stripped down from the framr package written by Dan Auerbach. https://github.com/FRAMverse/framr/

Usage

```
aeq_mortality(fram_db, run_id = NULL, msp = TRUE)
```

Arguments

`fram_db`	Fram database object
`run_id`	numeric, RunID(s) as ID or ID:ID
`msp`	Do we use MSP expansion? Logical, default true.

Examples

```
## Not run:
fram_db |> aeq_mortality(run_id = 132)

## End(Not run)
```

`bkfram_checks_coho` *Performs error checks of a backwards FRAM run Returns nested tibble with diagnostics*

Description

Performs error checks of a backwards FRAM run Returns nested tibble with diagnostics

Usage

```
bkfram_checks_coho(fram_db, backward_run_id = NULL, forward_run_id = NULL)
```

Arguments

`fram_db`	fram database object, supplied through connect_fram_db
`backward_run_id`	numeric, RunID
`forward_run_id`	numeric, RunID

Examples

```
## Not run:
fram_db |> bkfram_checks_coho(backward_run_id = 132, forward_run_id = 133)

## End(Not run)
```

`change_run_id` *Changes a run's ID number in a FRAM database*

Description

Changes a run's ID number in a FRAM database

Usage

```
change_run_id(fram_db, old_run_id, new_run_id)
```

Arguments

`fram_db`	FRAM database object
`old_run_id`	FRAM run ID to be changed
`new_run_id`	New FRAM run ID

Examples

```
## Not run: fram_db |> change_run_id(old_run_id = 132, new_run_id = 300)
```

`coho_mark_rates` **[Experimental]** *Returns a tibble displaying predicted FRAMencounter mark rates by fishery, fishery type, and time-step.*

Description

[Experimental] Returns a tibble displaying predicted FRAMencounter mark rates by fishery, fishery type, and time-step.

Usage

```
coho_mark_rates(fram_db, run_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`run_id`	Run ID (optional)

Examples

```
## Not run:
fram_db |> coho_mark_rates(run_id)

## End(Not run)
```

`compare_databases` *Compare tables in two equivalent FRAM databases*

Description

Function supports QAQC practices by comparing the tables of two FRAM databases and identifying (and quantifying) differences.

Usage

```
compare_databases(
  file1,
  file2,
  runid_use = NULL,
  tables_use = NULL,
  slim = FALSE,
  quiet = TRUE
)
```

Arguments

`file1`	Character string. FRAM database that contains the results from running baseline FRAM runs (e.g., our "production" version).
`file2`	Character string. FRAM database that contains the results from running modified FRAM runs (e.g., running a forked version of FRAM or using modified input values)
`runid_use`	Numeric vector of the run_ids to compare. Optional. If not provided, compare all run ids in the databases.
`tables_use`	Vector of character strings. Optional. If provided, only compare the listed tables.
`slim`	Logical. Optional, defaults to FALSE. If TRUE, do not include `$tabs_file1` and `$tabs_file2` in output list.
`quiet`	Logical, defaults to TRUE. When TRUE, suppress messages showing individual steps.

Details

The key output is the `$ratios` tibble, which contains every comparison of relevant table entries in long-form. These comparisons are implemented by first aligning corresponding table rows using appropriate key columns (e.g. run_id, fishery_id, stock_id, age, time_step, etc).

In `$ratios`, the `table` and `variable` columns specify the table column being compared, respectively. `prop.err`, `abs.err`, and `scale.err` provide measures of the changes between the "original" value (from `file1`) and the "comparison" value (from `file2`). More on those below. The `original` and `comparison` columns give the actual values being compared. `run_id` through `time_step` specify the rows being compared. `bkfram.off.by.fish` and `bkfram.off.by.prop` provide the context for the comparison (more on that below).

Quantifying error

Because FRAM involves numerical solvers, we expect some small differences in table entries even when comparing two effectively equivalent databases. `compare_databases()` provides three metrics for these changes. In each case, it is assumed that `file1` is the reference file; the "error" measures all show how much the value in `file2` changed relative to the corresponding value in `file1`. The simplest measure of error is the `abs.err`. This is the absolute value of the difference between the two values. If we're looking at an entry with table = "Mortality" and variable = "landed_catch", then an abs.err of 5 means that the `file2` entry was five fish more or less than the `file1` entry. You can confirm this by looking at the `original` and `comparison` columns. While `abs.err` is the most easily interpreted, it is often not very meaningful when looking across tables and variables. After all, an `abs.err` value of 5 could mean a a relatively meaningless change of five fish for a landed catch entry that was originally thousands of fish, but the same value of 5 would be a huge change in fishing effort if it were for a fishery scaler entry.

One way to make error comparable across tables and variables is to calculate the proportional error. If an entry changed by 0.01%, that's not meaningful, while if it changed by 10%, that is. `$prop.err` provides this proportional error, where -0.5 means the entry in file2 was 50% less than the corresponding value in file1, and a value of 2 means the entry in file2 was 200% more than the corresponding value in file2. ' This gives error in context of the original value, and is often a good a way to look for problems. However, we sometimes find very large `$prop.err` values for changes that aren't concerning. For example, we may have an entry for landed catch in the mortality table that was 0.00001 fish in file1, and 0 fish in file2. In all practicality these two values are identical, and the 0.00001 fish difference is likely one of random jitter in the numerical solver or rounding differences. However, our `$prop.err` value for this cell is `-1`, the most extreme negative change we can get. We can jointly look at `$abs.err` and `$prop.err` to address the potential for misleadingly large errors `$prop.err`, but it would be nice to have a single error metric that provides error in context without being sensitive to very small entries in file1.

`scale.err` is an elaboration on `$prop.err` that provides broader context. `$prop.err` takes the absolute error and scales by the original value in file1. `$scale.err` generalizes this idea, first calculating the average error for each table-variable combination, and then scaling the absolute error by the corresponding table-variable average. That is, if an entry for landed_catch in the Mortality table was 0.001 in file1, and was then 0.002 in file2, and the average of all landed_catch entries in file 1 was 1000, then the `prop.err` would be 1 (since file2 had double the value of file1, or `(0.002-0.001)/0.001`), and the `scale.err` would be `0.000001` (`(0.002-0.001)/1000`). This better captures our intuition that a difference of 0.001 fish in the landed catch isn't a big deal, since those values are typically huge. `scale.err` is thus a measure of error that is comparable across variables and tables, essentially answer the question "Has this entry changed a lot for this kind of variable and table?".

While scale.err is frequently the most useful error metric, compare_databases() provides all three. There may be contexts in which it's important to focus on the proportional error. For example, large proportional errors landed catch for the catch rare stocks can be important, but the much larger catch from other stock could water down the scale_err metric.

Addressing the backwardsFRAM wiggle

For post-season runs, the backwards FRAM algorithm is employed; its solver stops when the estimated fish numbers are within 1 fish of the target size. This means that there is the potential for substantial "wiggle" in bkFRAm values when comparing two databases. This wiggle can propagate to other tables, especially for stock-age-timesteps in which the target values were quite small (so a wiggle of +/- 1 fish would be a proportionally large amount). For this reason, it can be useful to see how our measures of error correspond to the errors in the corresponding bk fram table. For every table entry for which this makes sense (e.g., has a stock id, age, and timestep), $bkfram.off.by.fish gives the absolute error in the corresponding row of the BackwardsFram table, and bkfram.off.by.prop give the relative error (as a proportion) in the corresponding row of the BackwardsFram table. If this bkfram wiggle were the cause of observed errors, we would expect the largest errors to correspond to the largest $bkfram.off.by.fish or $bkfram.off.by.prop values.

Suggestions

For simple plotting to see if the original and comparison values fall on the 1:1 line, ggplot2::geom_point() can be used, with $ratios$original and $ratios$comparison for x and y, and a facet_wrap by table (and perhaps variable) to make plots readable. For identifying meaningful change, scale.err is likely the best measure of error. It can be helpful to plot scale.err against bkfram.off.by.fish or bkfram.off.by.prop to see if the table entries with the largest error correspond to the stock-fishery-age-timestep in which there's the largest wiggle in the backwards fram solutions.

When digging into individual tables, it can sometimes be helpful to look at the comparisons in $ratios_detailed, which contains additional columns which did not fit into the standardized formatting of $ratios.

Value

List of lists and tibbles containing comparison information:

- $ratios tibble comparing every entry of every relevant column of every table. See "Details" for column descriptions.
- $ratios_detailed list of tibbles showing the contents of $ratios broken into tables, with additional non-compared columns present (e.g., stock_name in $ratios_detailed$Stock)
- $nrow_tracker dataframe providing the number of rows in each table of file1 ($nrow.prod), file2 ($nrow.fork), and the joined comparison ($nrow.comp). Useful to track down cause of many-to-many join warnings that can result from duplicated table entries; unless there are duplicate entries, $nrow.comp should be less than or equal to the minimum of $nrow.prod and $nrow.fork.
- $tabs_file1 List containing the original fetched tables from file1. Not returned if argument slim is TRUE
- $tabs_file2 List containing the original fetched tables from file2. Not returned if argument slim is TRUE.

Examples

```
## Not run:
```

```
file1 = "Valid2022_Round_7_1_1_11142023_REFERENCE_fixed - fork rebuild.mdb"
file2 = "Valid2022_Round_7.1.1_11142023 - green river split.mdb"
out = tables_compare(file1, file2)

## End(Not run)
```

`compare_fishery_inputs`

Compares the fishery inputs of two runs

Description

Compares the fishery inputs of two runs

Usage

```
compare_fishery_inputs(fram_db, run_ids, tolerance = 0.01, verbose = TRUE)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Two run ids
`tolerance`	Tolerance for detecting changes
`verbose`	If `TRUE`, print an update to screen when there are no differences in recruits.

Examples

```
## Not run: fram_db |> compare_fishery_inputs(c(55, 56))
```

`compare_fishery_input_flags`

Compares the fishery flags of two runs

Description

Compares the fishery flags of two runs

Usage

```
compare_fishery_input_flags(fram_db, run_ids, verbose = TRUE)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Two run ids
`verbose`	If `TRUE`, print an update to screen when there are no differences in recruits.

Examples

```
## Not run: fram_db |> compare_fishery_input_flags(c(55, 56))
```

`compare_inputs` *Generates a dataframe that compares fishery scalers table for two runs identified by run_id's*

Description

Generates a dataframe that compares fishery scalers table for two runs identified by run_id's

Usage

```
compare_inputs(fram_db, run_ids)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Vector of two run_ids

Details

Comparisons assume the first run provided is the baseline, and provide relative changes from that. This includes percent changes ($percent.diff)include percent changes (infinite when)

Value

Data frame of differences. `$percentdiff` = change in quota (comparing the appropriate quotas based on fishery flags), `$regulation_comparison` = change in regulation (NS, MSF, NS + MSF). Columns present in the FisheriesScalers database are included, with `_original` and `_comparison` suffixes identifying entries associated with the first and second entries of `run_ids`, respectively.

Examples

```
## Not run: fram_db |> compare_inputs(c(100,101))
```

`compare_inputs_chart` *Generates a heat map of changed between two run inputs. Can be a very busy chart if not filtered down. Consider using a filter.*

Description

Generates a heat map of changed between two run inputs. Can be a very busy chart if not filtered down. Consider using a filter.

Usage

```
compare_inputs_chart(.data)
```

Arguments

`.data`	Dataframe origination from the compare_inputs() function

Examples

```
## Not run: fram_db |> compare_inputs(c(100, 101)) |> compare_inputs_chart()
```

`compare_non_retention_inputs`
Compares the non retention inputs of two runs

Description

Compares the non retention inputs of two runs

Usage

```
compare_non_retention_inputs(fram_db, run_ids, verbose = TRUE)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Two run ids
`verbose`	If `TRUE`, print an update to screen when there are no differences in recruits.

Examples

```
## Not run: fram_db |> compare_non_retention_inputs(c(55, 56))
```

compare_non_retention_input_flags *Compares the non retention flags of two runs*

Description

Compares the non retention flags of two runs

Usage

```
compare_non_retention_input_flags(fram_db, run_ids, verbose = TRUE)
```

Arguments

fram_db	FRAM database object
run_ids	Two run ids
verbose	If TRUE, print an update to screen when there are no differences in recruits.

Examples

```
## Not run: fram_db |> compare_non_retention_inputs(c(55, 56))
```

compare_recruits *Compares the recruit scalers of two runs*

Description

Compares the recruit scalers of two runs

Usage

```
compare_recruits(fram_db, run_ids, tolerance = 0.01, verbose = TRUE)
```

Arguments

fram_db	FRAM database object
run_ids	Two run ids
tolerance	Tolerance for detecting changes
verbose	If TRUE, print an update to screen when there are no differences in recruits.

Examples

```
## Not run: fram_db |> compare_recruits()
```

`compare_runs` *Generates a report to the console of changes to inputs between two runs*

Description

Generates a report to the console of changes to inputs between two runs

Usage

```
compare_runs(fram_db, run_ids, tolerance = 0.01)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Two run ids
`tolerance`	Tolerance of detection, 1 percent default

Examples

```
## Not run: fram_db |> compare_runs(c(55, 56))
```

`compare_stock_fishery_rate_scalers` *Compares the stock fishery rate scalers of two runs*

Description

Compares the stock fishery rate scalers of two runs

Usage

```
compare_stock_fishery_rate_scalers(fram_db, run_ids)
```

Arguments

`fram_db`	FRAM database object
`run_ids`	Two run ids

Examples

```
## Not run: fram_db |> compare_stock_fishery_rate_scalers(c(55, 56))
```

connect_fram_db *This is a connection object to a FRAM database. Returns a list used throughout the rest of this package which carries meta data.*

Description

This is a connection object to a FRAM database. Returns a list used throughout the rest of this package which carries meta data.

Usage

```
connect_fram_db(
  db_path,
  enforce_type = c("full", "transfer"),
  read_only = FALSE,
  quiet = FALSE
)
```

Arguments

db_path	Path to a FRAM database.
enforce_type	Not used
read_only	Optional argument to flag this connection as read-only (if set to TRUE). If TRUE, framrsquared functions that modify the database will abort rather than run. Use as a safety feature when working with a database that *must not* be modified.
quiet	Logical. Optitional argument; when TRUE, silences success message and database summary.

Details

Additional details...

Examples

```
## Not run: fram_db <- connect_fram_db('<path>')
```

copy_fishery_scalers *Experimental copying scaler inputs from one run to another DANGEROUS*

Description

Experimental copying scaler inputs from one run to another DANGEROUS

Usage

```
copy_fishery_scalers(fram_db, from_run, to_run, fishery_id = NULL)
```

Arguments

fram_db	FRAM database object
from_run	Run ID to be copied from
to_run	Run ID to be copied to
fishery_id	ID or IDs for specific fishery(s) to copy inputs to/from. If not provided, interactive option to copy inputs for all fisheries.

Examples

```
## Not run: framdb |> copy_fishery_scalers(132, 133, 87)
```

disconnect_fram_db *Safely disconnect from FRAM database*

Description

Safely disconnect from FRAM database

Usage

```
disconnect_fram_db(fram_db, quiet = TRUE)
```

Arguments

fram_db	FRAM database R object
quiet	Logical. Optional; when true, silences success message.

Examples

```
## Not run: disconnect_fram_db(fram_db)
```

error_check_code *Check code for common errors*

Description

Tool to streamline development. Currently this checks for use of filter() without dplyr::. This would call stats::filter(), which is usually not what I intend. As possible, add additional checks for issues that cause problems but do not give an informative error message (or any error message).

Usage

```
error_check_code(filepath, n = Inf)
```

Arguments

filepath	Path to R file to be checked
n	Number of rows to print. Default is to print all rows, but set to smaller values if output is overwhelming.

Examples

```
## Not run:
error_check_code("R/copy.R")

## End(Not run)
```

fetch_quarto_templates *Creates quarto template files*

Description

Creates template files from specified organization in the specified path. Generally `initialize_project()` will be more useful for new R projects, while `fetch_quarto_templates()` can be helpful when working with existing projects. See `initialize_project()` for details on adding template files for new organizations.

Usage

```
fetch_quarto_templates(
  to.path,
  organization = "WDFW",
  color = "coffee",
  overwrite = FALSE
)
```

Arguments

to.path	Character string. Destination file path for template files. Typically, root of Rproject directory.
organization	Character, defaults to "WDFW". Specifies the set of quarto templates to use. Currently only supports "WDFW".
color	Character string, defaults to "coffee". Specifies quarto template to use; organizations may have several.
overwrite	Boolean. Overwrite _quarto.yml and style.css files if they already exist? Defaults to FALSE.

Value

Nothing.

---

fetch_table *Fetches a complete table from a FRAM database. Returns a cleaned tibble.*

Description

Fetches a complete table from a FRAM database. Returns a cleaned tibble.

Usage

```
fetch_table(fram_db, table_name = NULL, warn = TRUE)
```

Arguments

fram_db	FRAM database object
table_name	Table to be fetched. If not given, a list of options will be printed
warn	Print a warning when fetching BackwardsFRAM table from a Chinook database? Defaults to TRUE.

Examples

```
## Not run: fram_db |> fetch_table('Mortality')
```

`fetch_table_bkchin` *Safely fetch Chinook BackwardsFRAM table*

Description

The BackwardsFRAM table uses a stock_id different numbering system from all other tables, and includes sums of joint stocks (e.g. for a marked and unmarked pair of stocks, BackwardsFRAM will typically have an additional stock which represents the sum of those two). Because the numbering is different but the column name is the same, joining the Chinook BackwardsFRAM table with other tables can cause problems.

Usage

```
fetch_table_bkchin(fram_db)
```

Arguments

`fram_db`	FRAM database object

Details

This function augments fetch_table by renaming the `stock_id` column to `bk_stock_id`, and then adding on the associated stock_id (with NAs when the bkfram stock is one of these new "sum" stocks and the associated bkfram stock names). This function only works for Chinook databases.

The resulting dataframe will necessarily NOT be an exact match to the BackwardsFRAM table in the FRAM database. The stock_id column will differ (containing normal stock ID values instead of bk stock ID values), and there will be two additional columns.

Examples

```
#' @examples
## Not run:
##Potentially problematic stock_id won't align with other tables
fram_db |> fetch_table('BackwardsFRAM')
## "safe" version of the table; stock_id WILL align with other tables
fram_db |> fetch_table_bkchin()

## End(Not run)
```

filter_ak *Filters a dataframe to Alaska fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* fishery_id *column name.*

Description

Filters a dataframe to Alaska fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id column name.

Usage

```
filter_ak(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_ak()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_ak(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_ak(species = "COHO")
```

filter_bc *Filters a dataframe to Canadian (BC) fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* fishery_id *column name.*

Description

Filters a dataframe to Canadian (BC) fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id column name.

Usage

```
filter_bc(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_bc()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_bc(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_bc(species = "COHO")
```

---

filter_ca

Filters a dataframe to California fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id *column name.*

Description

Filters a dataframe to California fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id column name.

Usage

```
filter_ca(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_ca()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_ca(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_ca(species = "COHO")
```

filter_coast *Filters a dataframe to Coastal fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* fishery_id *column name.*

Description

Filters a dataframe to Coastal fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id column name.

Usage

```
filter_coast(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_coast()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_coast(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_coast(species = "COHO")
```

filter_flag *NA's all the information in the FisheryScalers that's not being used e.g Flag 1 only NS Scalers will be returned*

Description

NA's all the information in the FisheryScalers that's not being used e.g Flag 1 only NS Scalers will be returned

Usage

```
filter_flag(.data)
```

Arguments

.data	Fishery Scalers table

Examples

```
## Not run:  fishery_scalers_table |> filter_flag()
```

`filter_marine` *Filters a dataframe to marine fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* `fishery_id` *column name.*

Description

Filters a dataframe to marine fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a `fishery_id` column name.

Usage

```
filter_marine(.data, species = NULL)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `framrsquared::fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to `NULL`

Examples

```
## Not run:
fram_dataframe |> filter_marine()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_marine(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_marine(species = "COHO")
```

`filter_net` *Filters a dataframe to net fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* `fishery_id` *column name.*

Description

Filters a dataframe to net fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a `fishery_id` column name.

Usage

```
filter_net(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_net()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_net(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_net(species = "COHO")
```

---

filter_or *Filters a dataframe to Oregon fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* fishery_id *column name.*

Description

Filters a dataframe to Oregon fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id column name.

Usage

```
filter_or(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_or()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_or(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_or(species = "COHO")
```

filter_puget_sound *Filters a dataframe to Puget Sound fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* `fishery_id` *column name.*

Description

Filters a dataframe to Puget Sound fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a `fishery_id` column name.

Usage

```
filter_puget_sound(.data, species = NULL)
```

Arguments

`.data`	Dataframe containing `fishery_id` column. Commonly, output from `framrsquared::fetch_table()`.
`species`	Optional argument to identify species if `.data` doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to `NULL`

Examples

```
## Not run:
fram_dataframe |> filter_puget_sound()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_puget_sound(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_puget_sound(species = "COHO")
```

filter_sport *Filters a dataframe to sport fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a* `fishery_id` *column name.*

Description

Filters a dataframe to sport fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a `fishery_id` column name.

Usage

```
filter_sport(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_sport()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_sport(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_sport(species = "COHO")
```

---

filter_wa

Filters a dataframe to Washington State fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id *column name.*

Description

Filters a dataframe to Washington State fisheries. Will automatically detect whether it's working with a Chinook or Coho dataset if the tables were generated within this package. Requires a fishery_id column name.

Usage

```
filter_wa(.data, species = NULL)
```

Arguments

.data	Dataframe containing fishery_id column. Commonly, output from framrsquared::fetch_table().
species	Optional argument to identify species if .data doesn't already. If provided, must be "COHO" or "CHINOOK". Defaults to NULL

Examples

```
## Not run:
fram_dataframe |> filter_wa()

## End(Not run)
framrosetta::fishery_chinook_fram |> filter_wa(species = "CHINOOK")
framrosetta::fishery_coho_fram |> filter_wa(species = "COHO")
```

find_tables_by_column_ *Finds tables that contain a specific column name*

Description

Finds tables that contain a specific column name

Usage

```
find_tables_by_column_(fram_db, column_name)
```

Arguments

fram_db	FRAM database object
column_name	Name of a column

Examples

```
## Not run: fram_db |> find_tables_by_column_('RunID')
```

fishery_mortality *Returns a tibble matching the Fishery Mortality screen.*

Description

Returns a tibble matching the Fishery Mortality screen.

Usage

```
fishery_mortality(fram_db, run_id = NULL, msp = TRUE)
```

Arguments

fram_db	FRAM database object
run_id	Run ID
msp	Model Stock Proportion, default TRUE

Examples

```
## Not run:
fram_db |> fishery_mortality(run_id = 101)

## End(Not run)
```

fram_clean_tables *Cleans the names of FRAM tables and coverts to a tibble*

Description

Cleans the names of FRAM tables and coverts to a tibble

Usage

```
fram_clean_tables(.data)
```

Arguments

.data	Dataframe

Examples

```
## Not run: fram_dataframe |> fram_clean_tables()
```

fram_database_species *Identifies the FRAM database species focus - Chinook or Coho*

Description

Identifies the FRAM database species focus - Chinook or Coho

Usage

```
fram_database_species(con)
```

Arguments

con	Connection to FRAM database

Examples

```
## Not run: fram_database_species(con)
```

fram_database_type *Identifies the FRAM database type - Full or Transfer*

Description

Identifies the FRAM database type - Full or Transfer

Usage

```
fram_database_type(con)
```

Arguments

con	Connection to FRAM database

Examples

```
## Not run: fram_database_type(con)
```

get_run_ids *Gets all run_ids of FRAM database*

Description

Gets all run_ids of FRAM database

Usage

```
get_run_ids(fram_db)
```

Arguments

fram_db	Fram database object

Examples

```
## Not run: fram_dataframe |> get_run_ids()
```

`initialize_project` **[Experimental]** *Initializes a FRAM project*

Description

By default, creates suggested folder structure from best coding practices, and adds WDFW-style yaml and style.css files to give quarto files consistent appearance. If you belong to another organization and want this function to support your own organization-specific quarto styling, reach out to the developers with a `_quarto.yml` and (optionally) `style.css` file.

Usage

```
initialize_project(
  folders = c("scripts", "original_data", "cleaned_data", "figures", "results",
    "results/quarto_output"),
  quarto = TRUE,
  organization = "WDFW",
  renv = FALSE,
  template.overwrite = TRUE,
  color = "coffee",
  quiet = TRUE
)
```

Arguments

`folders`	Vector of folders to create
`quarto`	Boolean. If TRUE, add quarto yaml file and style.css
`organization`	Character, defaults to "WDFW". Specifies the set of quarto templates to use. Currently only supports "WDFW".
`renv`	Boolean to initialize renv. Defaults to FALSE.
`template.overwrite`	Boolean. Overwrite _quarto.yml and style.css files if they already exist? Defaults to TRUE
`color`	Character string, defaults to "coffee". Specifies quarto template to use; organizations may have several.
`quiet`	Boolean, defaults to FALSE. If TRUE, suppresses informational messages.

Details

Dev note: new template files for additional organizations should be added to `inst/` in a subfolder matching an R-friendly organization name, and the same name should be added to the `organization` parameter description here and the `supported_organizations` in `fetch_quarto_templates()`.

Examples

```
## Not run:
framrsquared::initialize_project()

## End(Not run)
```

`input_summary_` *Generates an input summary based on a FisheryScalers dataframe. Probably end up streamlining / revising this.*

Description

Generates an input summary based on a FisheryScalers dataframe. Probably end up streamlining / revising this.

Usage

```
input_summary_(.data, run_id)
```

Arguments

`.data`	FisheryFishery scalers dataframe
`run_id`	Run ID number

Examples

```
## Not run: fishery_scalers_dataframe |> input_summary()
```

`make_impacts_per_catch_heatmap` *Make plots to show the amount of landed catch_per_impact*

Description

Identify how much reduction in landed catch at each fishery that would be needed to reduce the impacts on a focal stock by 1 fish.

Usage

```
make_impacts_per_catch_heatmap(fram_db, run_id, stock_id)
```

Arguments

`fram_db`	fram database connection
`run_id`	run_id of interest
`stock_id`	stock_id of interest.

Value

ggplot object

Examples

```
## Not run:
path = "FRAM compilations - readonly/2024-Pre-Season-Chinook-DB/2024 Pre-Season Chinook DB.mdb"
run_id = 132
stock_id = 3
make_impacts_per_catch_heatmap(path,
                               run_id = 132,
                               stock_id = 5)

## End(Not run)
```

---

management_week *Vectorized approach to calculating the management week, returns an integer*

Description

Vectorized approach to calculating the management week, returns an integer

Usage

```
management_week(date)
```

Arguments

date	A column with dates

Examples

```
## Not run:
data_fram |>
  mutate(mngmt_week = management_week(date_field))

## End(Not run)
```

mortality_scalers *Guestimate the impact on a particular stock by multiplying its mortalities by the* stock_mortality_ratio *produced by these functions.*

Description

Guestimate the impact on a particular stock by multiplying its mortalities by the stock_mortality_ratio produced by these functions.

Usage

```
mortality_scalers(fram_db, run_id, stock_id)
```

Arguments

fram_db	FRAM database object
run_id	Run ID
stock_id	A focal stock or stocks

Examples

```
## Not run: fram_db |> mortality_scalers(run_id = 101, stock_id = c(17:18))
```

msf_encounters *Produces the MSF screen report numbers for encounters. Returns different format depending database.*

Description

Produces the MSF screen report numbers for encounters. Returns different format depending database.

Usage

```
msf_encounters(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID

Examples

```
## Not run: fram_db |> msf_encounters(run_id = 101)
```

`msf_encounters_chinook_`

Returns a tibble matching the MSF screen report encounters for Chinook. This is specific for Chinook and in most cases msf_encounters() is preferable.

Description

Returns a tibble matching the MSF screen report encounters for Chinook. This is specific for Chinook and in most cases msf_encounters() is preferable.

Usage

```
msf_encounters_chinook_(fram_db)
```

Arguments

`fram_db`	FRAM database object

Examples

```
## Not run: fram_db |> msf_encounters_chinook_(run_id = 101)
```

`msf_encounters_coho_`

Returns a tibble matching the MSF screen report encounters for Coho This is specific for Coho and in most cases msf_encounters() is preferable.

Description

Returns a tibble matching the MSF screen report encounters for Coho This is specific for Coho and in most cases msf_encounters() is preferable.

Usage

```
msf_encounters_coho_(fram_db)
```

Arguments

`fram_db`	FRAM database object

Examples

```
## Not run: fram_db |> msf_encounters_coho_()
```

`msf_landed_catch` *Produces the MSF screen report numbers for landed catch. Returns different format depending database.*

Description

Produces the MSF screen report numbers for landed catch. Returns different format depending database.

Usage

```
msf_landed_catch(fram_db, run_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`run_id`	Run ID

Examples

```
## Not run: fram_db |> msf_landed_catch(run_id = 101)
```

`msf_landed_catch_chinook_` *Returns a tibble matching the MSF screen report landed catch for Chinook. This is specific for Chinook and in most cases msf_landed_catch() is preferable.*

Description

Returns a tibble matching the MSF screen report landed catch for Chinook. This is specific for Chinook and in most cases msf_landed_catch() is preferable.

Usage

```
msf_landed_catch_chinook_(fram_db)
```

Arguments

`fram_db`	FRAM database object

Examples

```
## Not run: fram_db |> msf_landed_catch_chinook_(run_id = 101)
```

`msf_landed_catch_coho_` *Returns a tibble matching the MSF screen report landed catch for Coho This is specific for Coho and in most cases msf_landed_catch() is preferable.*

Description

Returns a tibble matching the MSF screen report landed catch for Coho This is specific for Coho and in most cases msf_landed_catch() is preferable.

Usage

```
msf_landed_catch_coho_(fram_db)
```

Arguments

`fram_db`	FRAM database object

Examples

```
## Not run: fram_db |> msf_landed_catch_coho_()
```

`msf_mortalities` *Produces the MSF screen report numbers for mortalities. Returns different format depending database.*

Description

Produces the MSF screen report numbers for mortalities. Returns different format depending database.

Usage

```
msf_mortalities(fram_db, run_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`run_id`	Run ID

Examples

```
## Not run: fram_db |> msf_mortalities_coho_(run_id = 101)
```

msf_mortalities_chinook_ *Returns a tibble matching the MSF screen report mortalities for Chinook. This is specific for Chinook and in most cases msf_mortalities() is preferable.*

Description

Returns a tibble matching the MSF screen report mortalities for Chinook. This is specific for Chinook and in most cases msf_mortalities() is preferable.

Usage

```
msf_mortalities_chinook_(fram_db)
```

Arguments

fram_db	FRAM database object

Examples

```
## Not run: fram_db |> msf_mortalities_chinook_(run_id = 101)
```

msf_mortalities_coho_ *Returns a tibble matching the MSF screen report mortalities for Coho This is specific for Coho and in most cases msf_mortalities() is preferable.*

Description

Returns a tibble matching the MSF screen report mortalities for Coho This is specific for Coho and in most cases msf_mortalities() is preferable.

Usage

```
msf_mortalities_coho_(fram_db)
```

Arguments

fram_db	FRAM database object

Examples

```
## Not run: fram_db |> msf_mortalities_coho_()
```

msp_mortality *Expand Chinook mortality table using Model-Stock Proportion*

Description

See https://framverse.github.io/fram_doc/calcs_data_chin.html#46_Model-Stock_Proportion.

Usage

```
msp_mortality(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID (optional)

Value

Mortality table with mortality values expanded by msp

Examples

```
## Not run:
fram_db |> msp_mortality(run_id = 132)

## End(Not run)
```

NR_flag_translate *Provides English translation of numeric non-retention flags*

Description

Provides English translation of numeric non-retention flags

Usage

```
NR_flag_translate(vec)
```

Arguments

vec	vector of flags

Examples

```
## Not run: NR_flag_translate(sample(1:4, 10, replace = T))
```

`plot_stock_mortality` *Creates an ordered bar chart with the top number of mortalities per fishery.*

Description

Creates an ordered bar chart with the top number of mortalities per fishery.

Usage

```
plot_stock_mortality(
  fram_db,
  run_id,
  stock_id,
  top_n = 10,
  filters_list = NULL,
  msp = TRUE
)
```

Arguments

`fram_db`	fram database object, supplied through connect_fram_db
`run_id`	numeric, RunID
`stock_id`	numeric, ID of focal stock
`top_n`	numeric, Number of fisheries to display
`filters_list`	list of framrsquared filter functions to apply before plotting.
`msp`	Use Model Stock Proportion expansion? Default is true

Examples

```
## Not run:
fram_db |> plot_stock_mortality(run_id = 132, stock_id = 17)
fram_db |> plot_stock_mortality(run_id = 132, stock_id = 17,
        filters_list = list(filter_wa, filter_marine))

## End(Not run)
```

plot_stock_mortality_time_step *Creates an ordered bar chart with the top number of mortalities per fishery and time step.*

Description

Creates an ordered bar chart with the top number of mortalities per fishery and time step.

Usage

```
plot_stock_mortality_time_step(
  fram_db,
  run_id,
  stock_id,
  top_n = 10,
  filters_list = NULL,
  msp = TRUE
)
```

Arguments

fram_db	fram database object, supplied through connect_fram_db
run_id	numeric, RunID
stock_id	numeric, ID of focal stock
top_n	numeric, Number of fisheries to display
filters_list	list of framrsquared filter functions to apply before plotting.
msp	Use Model Stock Proportion expansion? Default is true

Examples

```
## Not run:
fram_db |> stock_mortality_time_step(run_id = 132, stock_id = 17)

## End(Not run)
```

population_statistics *Returns a tibble matching the Population Statistics screen.*

Description

Returns a tibble matching the Population Statistics screen.

Usage

```
population_statistics(fram_db, run_id = NULL)
```

Arguments

fram_db	FRAM database object
run_id	Run ID

Examples

```
## Not run: fram_db |> population_statistics(run_id = 101)
```

post_season_abundance *Generates post-season January age 3 abundances by stock from post-season databases. Used for forecasting.*

Description

Generates post-season January age 3 abundances by stock from post-season databases. Used for forecasting.

Usage

```
post_season_abundance(fram_db, units = c("ja3", "oa3"))
```

Arguments

fram_db	FRAM database object
units	Default January Age 3 'ja3', optional ocean age 3 'oa3'

Examples

```
## Not run: framdb |> post_season_abundance()
```

provide_table_names *List names of FRAM table*

Description

Provides list of FRAm database names, typically useful for internal functions.

Usage

```
provide_table_names(is_full = TRUE)
```

Arguments

is_full	Logical. Provide names for a full FRAM database (TRUE) or a model transfer (FALSE)?

Value

Character string of the names of FRAM tables

Examples

```
provide_table_names(is_full = FALSE)
```

remove_run *Removes a run in a FRAM database*

Description

Removes a run in a FRAM database

Usage

```
remove_run(fram_db, run_id)
```

Arguments

fram_db	FRAM database object
run_id	FRAM run ID to be deleted

Examples

```
## Not run: fram_db |> delete_run(run_id = 132)
```

run_info *Provides a print out of Run ID information*

Description

Provides a print out of Run ID information

Usage

```
run_info(fram_db, run_id)
```

Arguments

fram_db	FRAM database object
run_id	FRAM run ID

Examples

```
## Not run: fram_db |> run_info(run_id = 132)
```

scalers_flag_translate *Provides English translation of numeric scalers flags*

Description

Provides English translation of numeric scalers flags

Usage

```
scalers_flag_translate(vec)
```

Arguments

vec	vector of flags

Examples

```
## Not run: scaler_flags_translate(sample(c(1, 2, 7, 8, 17, 18, 27, 28), 10, replace = T))
```

statistical_week — *Vectorized approach to calculating the statistical week, returns an integer*

Description

Statistical weeks start on mondays, so the first statistical week of the year starts on the first monday of the year. (Contrast with management weeks which start on Sundays).

Usage

```
statistical_week(date)
```

Arguments

date	A vector of dates

Examples

```
## Not run:
data_fram |>
  mutate(mngmt_week = statistical_week(date_field))

## End(Not run)
```

stock_age_checker — *Helper function to check that all stock x age combinations are present*

Description

#' Intended for internal use, makes some assumptions about inputs.

Usage

```
stock_age_checker(table_name, NumStk, old_stockcount, df, min_age, max_age)
```

Arguments

table_name	Character of table name, for informative messages.
NumStk	Maximum number of stock, pulled from BaseID table
old_stockcount	Number of stock in previous FRAM baseperiod. Only looks for problems for StockID > this number.
df	Dataframe to check. Must have columns "stock_id" and "age" (which are the names for relevant columns of framrsquared::fetch_table).
min_age	Minimum age modeled. Should be the min_age from the baseid_df.
max_age	Maximum age modeled. Should be the max_age from the baseid_df.

Value

numeri; 0 if no warning, 1 if warning.

`stock_check_helper` *Helper function to check that stock id make sense*

Description

More thorough checking than stock_id_comp. Checks that the number of stock IDs makes sense given `NumStk`, that Stock IDs are sequential (in the sense that if NumStk = n, every integer up to n is represented). Optionally, can check that each stock ID is unique.

Usage

```
stock_check_helper(table_name, NumStk, stock_vec, uniques_only = FALSE)
```

Arguments

`table_name`	Character of table name, for informative messages.
`NumStk`	Maximum number of stock, pulled from BaseID table
`stock_vec`	vector of stock ids to check. Presumably column of fetched table.
`uniques_only`	Do we want warnings if there are duplicats of StockIDs? Useful for tables like Stock and Growth that should have only one entry per stock. Logical, default = `FALSE`.

Value

Numeric, returning number of warnings detected.

`stock_id_comp` *Helper function to check that stock id exist in the Stock database*

Description

Intended for internal use, makes some assumptions about inputs.

Usage

```
stock_id_comp(table_name, df, stock_ref)
```

Arguments

`table_name`	Character of table name, for informative messages
`df`	Dataframe
`stock_ref`	numeric vector of all stock IDs. Should be stock_df$stock_id.

Value

numeric; 0 if no warning, 1 if warning.

`stock_mortality` *Returns a tibble matching the Fishery Mortality screen.*

Description

Returns a tibble matching the Fishery Mortality screen.

Usage

```
stock_mortality(fram_db, run_id = NULL)
```

Arguments

`fram_db`	FRAM database object
`run_id`	Run ID

Examples

```
## Not run:
fram_db |>
 stock_mortality(run_id=132) |>
 filter(stock_id == 17, fishery_id == 36)


## End(Not run)
```

`styleguide` *Framrsquared style guide*

Description

For easy readability, we want to use consistent coding style when developing code for framrsquared. Presently this includes (a) using <- for assignment, and (b) using snakecase for variable and function names. The functions here streamlining checking R code for consistency with this style.

Usage

```
frs_stylecheck_assignment(filepath, n = Inf)

frs_stylecheck_snakecase(filepath, n = Inf)
```

Arguments

filepath	Path to R file to be checked
n	Number of rows to print. Default is to print all rows, but set to smaller values if output is overwhelming.

Details

frs_stylecheck_assignment() takes the path to an R file, and prints (and returns) any rows that may be mis-using the = for assignment. Note that it will give false positives for arguments defined in function calls if the call spans multiple lines, as well as = signs included in character strings.

frs_stylecheck_snakecase takes the path to an R file and prints (and returns) the names of any variables assigned using <- that do not include underscores. This will identify variables that do not use snakecase, but will also give false positive matches for variables that are single words and thus do not need snakecase.

Examples

```
## Not run:
frs_stylecheck_assignment("R/copy.R")
frs_stylecheck_snakecase("R/copy.R")

## End(Not run)
```

---

truns_fisheries *Returns a dataframe with fisheries defined by the TRuns report driver*

Description

Returns a dataframe with fisheries defined by the TRuns report driver

Usage

```
truns_fisheries(fram_db)
```

Arguments

fram_db	FRAM database object

Examples

```
## Not run: truns <- truns_fisheries(fram_db)
```

truns_stocks *Returns a dataframe with stocks defined by the TRuns report driver*

Description

Returns a dataframe with stocks defined by the TRuns report driver

Usage

```
truns_stocks(fram_db)
```

Arguments

fram_db	FRAM database object

Examples

```
## Not run: truns <- truns_stocks(fram_db)
```

validate_fram_db *Convenience function to check fram_db input*

Description

Convenience function to check fram_db input

Usage

```
validate_fram_db(
  fram_db,
  db_type = NULL,
  db_species = NULL,
  call = rlang::caller_env()
)
```

Arguments

fram_db	FRAM database object
db_type	Enforcement of a database type 'full' or 'transfer'
db_species	Enforcement of a species 'COHO' or 'CHINOOK'
call	internal use: identify name of function that called this function (for informative error message)

validate_run_id *Convenience function to check run_id input*

Description

Convenience function to check run_id input

Usage

```
validate_run_id(fram_db, run_id, call = rlang::caller_env())
```

Arguments

fram_db	FRAM database object
run_id	one or more run_ids
call	internal use: identify name of function that called this function (for informative error message)

welcome *Welcome message, summarizing database information*

Description

Welcome message, summarizing database information

Usage

```
welcome(con)
```

Arguments

con	FRAM database connection

Examples

```
## Not run: welcome(con)
```

Index