

Package: framqaqc (via r-universe)

September 20, 2026

Title What the Package Does (One Line, Title Case)
Version 0.0.0.9000
Description QAQC tools for comparing FRAM databases.
License MIT + file LICENSE
Encoding UTF-8
Roxygen list(markdown = TRUE)
URL <https://github.com/FRAMverse/framqaqc>,
<https://framverse.github.io/framqaqc/>
BugReports <https://github.com/FRAMverse/framqaqc/issues>
Imports cli, DBI, dplyr, framrosetta, framrsquared, ggplot2, gt,
janitor, purrr, rlang, stats, stringr, tibble, tidyr,
tidyselect
Suggests knitr, rmarkdown
Remotes FRAMverse/framrsquared
VignetteBuilder knitr
Config/roxygen2/version 8.1.0
Config/pak/sysreqs
cmake make libicu-dev libuv1-dev libxml2-dev unixodbc-dev libssl-dev libnode-dev libx11-dev
Repository <https://framverse.r-universe.dev>
Date/Publication 2026-08-21 18:04:00 UTC
RemoteUrl <https://github.com/FRAMverse/framqaqc>
RemoteRef HEAD
RemoteSha 3cc915a7c9ab321c8e4c1e42fe1dcd6a22080ecd

Contents

compare_databases	2
compare_table_across_dbs	5
plot_comparisons_exact	6

plot_comparisons_ratio	6
run_info	7
summarize_exact	7
summarize_ratio	8
Index	9

compare_databases	[Experimental] <i>Compare tables in two equivalent FRAM databases</i>
-------------------	--

Description

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

Usage

```
compare_databases(  
  fram_db_1,  
  fram_db_2,  
  runid_use = NULL,  
  tables_use = NULL,  
  slim = TRUE,  
  quiet = TRUE  
)
```

Arguments

fram_db_1	connection to FRAM database that contains the results from running baseline FRAM runs (e.g., our "original" version).
fram_db_2	connection to FRAM database that contains the results from running modified FRAM runs (e.g., running a new 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 TRUE. 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 `fram_db_1`) and the "comparison" value (from `fram_db_2`). More on those below. The original and new 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 `fram_db_1` is the reference file; the "error" measures all show how much the value in `fram_db_2` changed relative to the corresponding value in `fram_db_1`. 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 `fram_db_2` entry was five fish more or less than the `fram_db_1` entry. You can confirm this by looking at the original and new 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 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. `$prop_err` provides this proportional error, where -0.5 means the entry in `fram_db_2` was 50% less than the corresponding value in `fram_db_1`, and a value of 2 means the entry in `fram_db_2` was 200% more than the corresponding value in `fram_db_1`. This gives error in context of the original value, and is often a good way to look for problems. However, we sometimes find very large `$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 `fram_db_1`, and 0 fish in `fram_db_2`. 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 `fram_db_1`.

`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 `fram_db_1`. `$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 `fram_db_1`, and was then 0.002 in `fram_db_2`, and the average of all landed catch entries in `fram_db_1` was 1000, then the `prop_err` would be 1 (since `fram_db_2` had double the value of `fram_db_1`, or $(0.002 - 0.001) / 0.001$), and the `scale_err` would be 0.00001 ($(0.002 - 0.001) / 1000$). 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 new values fall on the 1:1 line, `ggplot2::geom_point()` can be used, with `$ratios$original` and `$ratios$new` 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$Stock`). Not shown if `slim` is TRUE.
- `$nrow_tracker` dataframe providing the number of rows in each table of `file1` (`$nrow_original`), `file2` (`$nrow_new`), and the joined comparison (`$nrow_comparison`). Useful to track down cause of many-to-many join warnings that can result from duplicated table entries; unless there are duplicate entries, `$nrow_comparison` should be less than or equal to the minimum of `$nrow_original` and `$nrow_new`.
- `$tabs_file1` List containing the original fetched tables from `file1`. Not returned if argument `slim` is TRUE
- `$tabs_file2` List containing the original fetched tables from `file2`. Not returned if argument `slim` is TRUE.

Examples

```
## Not run:
fram_db_1 = connect_fram_db("Valid2022_Round_7.1.1_11142023_REFERENCE_fixed - fork rebuild.mdb")
fram_db_2 = connect_fram_db("Valid2022_Round_7.1.1_11142023 - green river split.mdb")
out = tables_compare(fram_db_1, fram_db_2)
```

```
## End(Not run)
```

```
compare_table_across_dbs
```

Compare tables between two databases

Description

Compare tables between two databases

Usage

```
compare_table_across_dbs(fram_db_1, fram_db_2, table_name)
```

Arguments

fram_db_1	fram database connection to first database (used as reference for ratio calculations)
fram_db_2	fram database connection to second database
table_name	Name of table to compare

Value

tibble of comparisons, including run_id, as appropriate stock_id, age, fishery_id, time_step. \$db1 and \$db2 give the values in the database for column \$metric; \$diff_exact has db1-db2, and \$diff_ratio has db2/db1.

Examples

```
## Not run:
library(framrsquared)
fram_a = connect_fram_db(
  "C:/Repos/fram multirun testing/Formal testing/coho_preseason_notamm_a.mdb",
  read_only = TRUE,
  quiet = TRUE)
fram_c = connect_fram_db(
  "C:/Repos/fram multirun testing/Formal testing/coho_preseason_notamm_c.mdb",
  read_only = TRUE,
  quiet = TRUE)
compare_table_across_dbs(fram_a, fram_c, "Mortality")
disconnect_fram_db(fram_a)
disconnect_fram_db(fram_b)

## End(Not run)
```

plot_comparisons_exact

Plot the values of the two databases against each other

Description

Creates faceted plots based on the output of `compare_table_across_dbs()`, with the values from one database plotted against the corresponding values from the other. If two databases are exactly the same, all points should fall on the 1:1 line; dashed 1:1 line included as reference. To improve performance, avoids plotting duplicate and near-duplicate entries. "near-duplicate" is controlled by argument `round_digits`.

Usage

```
plot_comparisons_exact(tab, round_digits = 0)
```

Arguments

<code>tab</code>	tibble produced from <code>compare_table_across_dbs()</code> .
<code>round_digits</code>	Digits to round to before identifying entries as "near-duplicate" entries. Defaults to 0 (e.g., round to the nearest fish). Appropriate to reduce when comparing tables in which values are much smaller.

Value

ggplot object

plot_comparisons_ratio

Plot the ratios of database comparisons

Description

Creates faceted plots based on the output of `compare_table_across_dbs()`, with the ratio of `db2/db1` plotted against the values of `db1`. If two databases are exactly the same, all points should have a ratio of exactly 1; dashed horizontal line at 1 is included as reference. To improve performance, avoids plotting duplicate and near-duplicate entries. "near-duplicate" values of `db1` is controlled by argument `round_digits`; ratio values within 0.001 are treated as near-duplicate.

Usage

```
plot_comparisons_ratio(tab, round_digits = 0)
```

Arguments

tab tibble produced from `compare_table_across_dbs()`.
round_digits digits to round db1 to in order to avoid plotting near-duplicate points.

Value

ggplot object

run_info	<i>Provides a print out of Run ID information</i>
----------	---

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)
```

summarize_exact	<i>Summarize exact differences from <code>compare_table_across_dbs()</code></i>
-----------------	---

Description

Makes a quick summary of the outputs of `compare_table_across_dbs()`, giving min, max, 0.01 and 0.99 quantiles of the absolute differences of values by run id. A perfect match would have values that are all 0.

Usage

```
summarize_exact(tab)
```

Arguments

tab tibble produced from `compare_table_across_dbs()`.

Value

gt with summary information.

summarize_ratio	<i>Summarize ratio differences from</i> compare_table_across_dbs()
-----------------	--

Description

Makes a quick summary of the outputs of `compare_table_across_dbs()`, giving min, max, 0.01 and 0.99 quantiles of the ratio of values from db2 / db1, by run id. Ignore entries with values ≤ 1 to db1 to avoid divide-by-zero issues and inflation of small differences.

Usage

```
summarize_ratio(tab)
```

Arguments

tab tibble produced from `compare_table_across_dbs()`.

Value

gt with summary information.

Index

`compare_databases`, [2](#)
`compare_table_across_dbs`, [5](#)

`plot_comparisons_exact`, [6](#)
`plot_comparisons_ratio`, [6](#)

`run_info`, [7](#)

`summarize_exact`, [7](#)
`summarize_ratio`, [8](#)