Package 'starschemar'

January 8, 2024

Title Obtaining Stars from Flat Tables

Version 1.2.4

Description Data in multidimensional systems is obtained from operational systems and is transformed to adapt it to the new structure. Frequently, the operations to be performed aim to transform a flat table into a star schema. Transformations can be carried out using professional extract, transform and load tools or tools intended for data transformation for end users. With the tools mentioned, this transformation can be carried out, but it requires a lot of work. The main objective of this package is to define transformations that allow obtaining stars from flat tables easily. In addition, it includes basic data cleaning, dimension enrichment, incremental data refresh and query operations, adapted to this context.

License MIT + file LICENSE

URL https://josesamos.github.io/starschemar/, https://github.com/josesamos/starschemar

BugReports https://github.com/josesamos/starschemar/issues

Depends R (>= 2.10)

Imports dplyr, generics, methods, purrr, rlang, snakecase, stats, tibble, tidyr

Suggests knitr, pander, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Jose Samos [aut, cre] (<https://orcid.org/0000-0002-4457-3439>), Universidad de Granada [cph]

Maintainer Jose Samos <jsamos@ugr.es>

Repository CRAN

Date/Publication 2024-01-08 15:30:02 UTC

R topics documented:

`character_dimensions` *Transform dimension numeric attributes to character*

Description

Transforms numeric type attributes of dimensions into character type. In a star_schema numerical data are measurements that are situated in the facts. Numerical data in dimensions are usually codes, day, week, month or year numbers. There are tools that consider any numerical data to be a measurement, for this reason it is appropriate to transform the numerical data of dimensions into character data.

Usage

```
character_dimensions(st, length_integers = list(), NA_replacement_value = NULL)

## S3 method for class 'star_schema'
character_dimensions(st, length_integers = list(), NA_replacement_value = NULL)
```

Arguments

st	A star_schema object.
length_integers	A list of pairs name = length, for each attribute name its length.
NA_replacement_value	A string, value to replace NA values.

Details

It allows indicating the amplitude for some fields, filling with zeros on the left. This is useful to make the alphabetical order of the result correspond to the numerical order.

It also allows indicating the literal to be used in case the numerical value is not defined.

If a role playing dimension has been defined, the transformation is performed on it.

Value

A star_schema object.

See Also

Other star schema and constellation definition functions: constellation(), role_playing_dimension(), snake_case(), star_schema()

Examples

```
st <- star_schema(mrs_age_test, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("date", "week", "year")
  ) |>
  character_dimensions(length_integers = list(week = 2),
                       NA_replacement_value = "Unknown")
```

constellation constellation *S3 class*

Description

Creates a constellation object from a list of star_schema objects. All dimensions with the same name in the star schemas have to be conformable.

Usage

```
constellation(lst, name = NULL)
```

Arguments

lst	A list of star_schema objects.
name	A string.

Value

A constellation object.

See Also

Other star schema and constellation definition functions: character_dimensions(), role_playing_dimension(), snake_case(), star_schema()

Examples

```
ct <- constellation(list(st_mrs_age, st_mrs_cause), name = "mrs")
```

constellation_as_multistar *Export a constellation as a* multistar

Description

Once we have refined the format or content of facts and dimensions, we can obtain a multistar. A multistar only distinguishes between general and conformed dimensions, each dimension has its own data. It can contain multiple fact tables.

Usage

```
constellation_as_multistar(ct)

## S3 method for class 'constellation'
constellation_as_multistar(ct)
```

Arguments

ct	A constellation object.

Value

A multistar object.

See Also

Other results export functions: constellation_as_tibble_list(), multistar_as_flat_table(), star_schema_as_flat_table(), star_schema_as_multistar(), star_schema_as_tibble_list()

Examples

```
ms <- ct_mrs |>
  constellation_as_multistar()
```

---

constellation_as_tibble_list

Export a constellation as a tibble *list*

Description

Once we have refined the format or content of facts and dimensions, we can obtain a tibble list with them. Role playing dimensions can be optionally included.

Usage

```
constellation_as_tibble_list(ct, include_role_playing = FALSE)

## S3 method for class 'constellation'
constellation_as_tibble_list(ct, include_role_playing = FALSE)
```

Arguments

ct	A constellation object.
include_role_playing	A boolean.

Value

A list of tibble objects.

See Also

Other results export functions: constellation_as_multistar(), multistar_as_flat_table(), star_schema_as_flat_table(), star_schema_as_multistar(), star_schema_as_tibble_list()

Examples

```
tl <- ct_mrs |>
  constellation_as_tibble_list()

tl <- ct_mrs |>
  constellation_as_tibble_list(include_role_playing = TRUE)
```

`ct_mrs` *Constellation for Mortality Reporting System*

Description

Constellation for the Mortality Reporting System considering age and cause classification.

Usage

```
ct_mrs
```

Format

A `constellation` object.

Examples

```
# Defined by:

ct_mrs <- constellation(list(st_mrs_age, st_mrs_cause), name = "mrs")
```

`ct_mrs_test` *Constellation for Mortality Reporting System Test*

Description

Constellation for the Mortality Reporting System considering age and cause classification data test.

Usage

```
ct_mrs_test
```

Format

A `constellation` object.

Examples

```
# Defined by:

ct_mrs_test <-
  constellation(list(st_mrs_age_test, st_mrs_cause_test), name = "mrs_test")
```

`define_dimension` *Define dimensions in a* `dimensional_model` *object*

Description

To define a dimension in a `dimensional_model` object, we have to define its name and the set of attributes that make it up.

Usage

```
define_dimension(st, name = NULL, attributes = NULL)

## S3 method for class 'dimensional_model'
define_dimension(st, name = NULL, attributes = NULL)
```

Arguments

`st`	A `dimensional_model` object.
`name`	A string, name of the dimension.
`attributes`	A vector of attribute names.

Details

To get a star schema (a `star_schema` object) we need a flat table (implemented through a `tibble`) and a `dimensional_model` object. The definition of dimensions in the `dimensional_model` object is made from the flat table column names. Using the `dput` function we can list the column names of the flat table so that we do not have to type their names.

Value

A `dimensional_model` object.

See Also

Other star definition functions: `define_fact()`, `dimensional_model()`

Examples

```
# dput(colnames(mrs_age))
#
# c(
#   "Reception Year",
#   "Reception Week",
#   "Reception Date",
#   "Data Availability Year",
#   "Data Availability Week",
#   "Data Availability Date",
#   "Year",
#   "WEEK",
#   "Week Ending Date",
#   "REGION",
#   "State",
#   "City",
#   "Age Range",
#   "Deaths"
# )

dm <- dimensional_model() |>
  define_dimension(name = "When",
                   attributes = c("Week Ending Date",
                                  "WEEK",
                                  "Year")) |>
  define_dimension(name = "When Available",
                   attributes = c("Data Availability Date",
                                  "Data Availability Week",
                                  "Data Availability Year")) |>
  define_dimension(name = "Where",
                   attributes = c("REGION",
                                  "State",
                                  "City")) |>
  define_dimension(name = "Who",
                   attributes = c("Age Range"))
```

`define_fact` *Define facts in a* `dimensional_model` *object*

Description

To define facts in a `dimensional_model` object, the essential data is a name and a set of measurements that can be empty (does not have explicit measurements). Associated with each measurement, an aggregation function is required, which by default is SUM.

Usage

```
define_fact(
  st,
  name = NULL,
  measures = NULL,
  agg_functions = NULL,
  nrow_agg = "nrow_agg"
)

## S3 method for class 'dimensional_model'
define_fact(
  st,
  name = NULL,
  measures = NULL,
  agg_functions = NULL,
  nrow_agg = "nrow_agg"
)
```

Arguments

st	A dimensional_model object.
name	A string, name of the fact.
measures	A vector of measure names.
agg_functions	A vector of aggregation function names. If none is indicated, the default is SUM. Additionally they can be MAX or MIN.
nrow_agg	A string, measurement name for the number of rows aggregated.

Details

To get a star schema (a star_schema object) we need a flat table (implemented through a tibble) and a dimensional_model object. The definition of facts in the dimensional_model object is made from the flat table column names. Using the dput function we can list the column names of the flat table so that we do not have to type their names.

Associated with each measurement there is an aggregation function that can be SUM, MAX or MIN. Mean is not considered among the possible aggregation functions: The reason is that calculating the mean by considering subsets of data does not necessarily yield the mean of the total data.

An additional measurement corresponding to the number of aggregated rows is always added which, together with SUM, allows us to obtain the mean if needed.

Value

A dimensional_model object.

See Also

Other star definition functions: define_dimension(), dimensional_model()

Examples

```
# dput(colnames(mrs_age))
#
# c(
#   "Reception Year",
#   "Reception Week",
#   "Reception Date",
#   "Data Availability Year",
#   "Data Availability Week",
#   "Data Availability Date",
#   "Year",
#   "WEEK",
#   "Week Ending Date",
#   "REGION",
#   "State",
#   "City",
#   "Age Range",
#   "Deaths"
# )

dm <- dimensional_model() |>
  define_fact(
    name = "mrs_age",
    measures = c("Deaths"),
    agg_functions = c("SUM"),
    nrow_agg = "nrow_agg"
  )

dm <- dimensional_model() |>
  define_fact(
    name = "mrs_age",
    measures = c("Deaths")
  )

dm <- dimensional_model() |>
  define_fact(name = "Factless fact")
```

`dimensional_model` `dimensional_model` *S3 class*

Description

An empty `dimensional_model` object is created in which definition of facts and dimensions can be added.

Usage

```
dimensional_model()
```

Details

To get a star schema (a `star_schema` object) we need a flat table (implemented through a `tibble`) and a `dimensional_model` object. The definition of facts and dimensions in the `dimensional_model` object is made from the flat table columns. Each attribute can only appear once in the definition.

Value

A `dimensional_model` object.

See Also

`star_schema`

Other star definition functions: `define_dimension()`, `define_fact()`

Examples

```
dm <- dimensional_model()
```

`dimensional_query` `dimensional_query` *S3 class*

Description

An empty `dimensional_query` object is created where you can select fact measures, dimension attributes and filter dimension rows.

Usage

```
dimensional_query(ms = NULL)
```

Arguments

`ms`	A `multistar` object.

Value

A `dimensional_query` object.

See Also

Other query functions: `filter_dimension()`, `run_query()`, `select_dimension()`, `select_fact()`

Examples

```
# ms_mrs <- ct_mrs |>
#  constellation_as_multistar()

# dq <- dimensional_query(ms_mrs)
```

---

dm_mrs_age *Star Definition for Mortality Reporting System by Age*

Description

Definition of facts and dimensions for the Mortality Reporting System considering the age classification.

Usage

```
dm_mrs_age
```

Format

A dimensional_model object.

Examples

```
# Defined by:

dm_mrs_age <- dimensional_model() |>
  define_fact(
    name = "mrs_age",
    measures = c(
      "Deaths"
    ),
    agg_functions = c(
      "SUM"
    ),
    nrow_agg = "nrow_agg"
  ) |>
  define_dimension(
    name = "when",
    attributes = c(
      "Week Ending Date",
      "WEEK",
      "Year"
    )
  ) |>
  define_dimension(
    name = "when_available",
```

```
    attributes = c(
      "Data Availability Date",
      "Data Availability Week",
      "Data Availability Year"
    )
  ) |>
  define_dimension(
    name = "where",
    attributes = c(
      "REGION",
      "State",
      "City"
    )
  ) |>
  define_dimension(
    name = "who",
    attributes = c(
      "Age Range"
    )
  )
```

---

dm_mrs_cause *Star Definition for Mortality Reporting System by Cause*

Description

Definition of facts and dimensions for the Mortality Reporting System considering the cause classification.

Usage

```
dm_mrs_cause
```

Format

A dimensional_model object.

Examples

```
# Defined by:

dm_mrs_cause <- dimensional_model() |>
  define_fact(
    name = "mrs_cause",
    measures = c(
      "Pneumonia and Influenza Deaths",
      "Other Deaths"
    ),
  ) |>
```

```
  define_dimension(
    name = "when",
    attributes = c(
      "Week Ending Date",
      "WEEK",
      "Year"
    )
  ) |>
  define_dimension(
    name = "when_received",
    attributes = c(
      "Reception Date",
      "Reception Week",
      "Reception Year"
    )
  ) |>
  define_dimension(
    name = "when_available",
    attributes = c(
      "Data Availability Date",
      "Data Availability Week",
      "Data Availability Year"
    )
  ) |>
  define_dimension(
    name = "where",
    attributes = c(
      "REGION",
      "State",
      "City"
    )
  )
```

`enrich_dimension_export` *Export selected attributes of a dimension*

Description

Export the selected attributes of a dimension, without repeated combinations, to enrich the dimension.

Usage

```
enrich_dimension_export(st, name = NULL, attributes = NULL)

## S3 method for class 'star_schema'
enrich_dimension_export(st, name = NULL, attributes = NULL)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.
attributes	A vector of attribute names.

Details

If it is a role dimension they cannot be exported, you have to work with the associated role playing dimension.

Value

A tibble object.

See Also

Other dimension enrichment functions: enrich_dimension_import_test(), enrich_dimension_import()

Examples

```
tb <-
  enrich_dimension_export(st_mrs_age,
                          name = "when_common",
                          attributes = c("week", "year"))
```

---

enrich_dimension_import

Import tibble *to enrich a dimension*

Description

For a dimension of a star schema a tibble is attached. This contains dimension attributes and new attributes. If values associated with all rows in the dimension are included in the tibble, the dimension is enriched with the new attributes.

Usage

```
enrich_dimension_import(st, name = NULL, tb)

## S3 method for class 'star_schema'
enrich_dimension_import(st, name = NULL, tb)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.
tb	A tibble object.

Details

Role dimensions cannot be directly enriched. If a role playing dimension is enriched, the new attributes are also added to the associated role dimensions.

Value

A star_schema object.

See Also

Other dimension enrichment functions: enrich_dimension_export(), enrich_dimension_import_test()

Examples

```
tb <-
  enrich_dimension_export(st_mrs_age,
                          name = "when_common",
                          attributes = c("week", "year"))

# Add new columns with meaningful data (these are not), possibly exporting
# data to a file, populating it and importing it.
tb <- tibble::add_column(tb, x = "x", y = "y", z = "z")

st <- enrich_dimension_import(st_mrs_age, name = "when_common", tb)
```

---

enrich_dimension_import_test

Import tibble *to test to enrich a dimension*

Description

For a dimension of a star schema a tibble is attached. This contains dimension attributes and new attributes. If values associated with all rows in the dimension are included in the tibble, the dimension is enriched with the new attributes. This function checks that there are values for all instances. Returns the dimension instances that do not match the imported data.

Usage

```
enrich_dimension_import_test(st, name = NULL, tb)

## S3 method for class 'star_schema'
enrich_dimension_import_test(st, name = NULL, tb)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.
tb	A tibble object.

Value

A dimension object.

See Also

Other dimension enrichment functions: enrich_dimension_export(), enrich_dimension_import()

Examples

```
tb <-
  enrich_dimension_export(st_mrs_age,
                          name = "when_common",
                          attributes = c("week", "year"))

# Add new columns with meaningful data (these are not), possibly exporting
# data to a file, populating it and importing it.
tb <- tibble::add_column(tb, x = "x", y = "y", z = "z")[-1, ]

tb2 <- enrich_dimension_import_test(st_mrs_age, name = "when_common", tb)
```

---

filter_dimension *Filter dimension*

Description

Allows you to define selection conditions for dimension rows.

Usage

```
filter_dimension(dq, name = NULL, ...)

## S3 method for class 'dimensional_query'
filter_dimension(dq, name = NULL, ...)
```

Arguments

dq	A dimensional_query object.
name	A string, name of the dimension.
...	Conditions, defined in exactly the same way as in dplyr::filter.

Details

Conditions can be defined on any attribute of the dimension (not only on attributes selected in the query for the dimension). The selection is made based on the function dplyr::filter. Conditions are defined in exactly the same way as in that function.

Value

A dimensional_query object.

See Also

Other query functions: dimensional_query(), run_query(), select_dimension(), select_fact()

Examples

```
dq <- dimensional_query(ms_mrs) |>
  filter_dimension(name = "when", when_happened_week <= "03") |>
  filter_dimension(name = "where", city == "Boston")
```

---

filter_fact_rows *Filter fact rows*

Description

Filter fact rows based on dimension conditions in a star schema. Dimensions remain unchanged.

Usage

```
filter_fact_rows(st, name = NULL, ...)

## S3 method for class 'star_schema'
filter_fact_rows(st, name = NULL, ...)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.
...	Conditions, defined in exactly the same way as in dplyr::filter.

Details

Filtered rows can be deleted using the `incremental_refresh_star_schema` function.

Value

A `star_schema` object.

See Also

Other incremental refresh functions: `get_star_schema_names()`, `get_star_schema()`, `incremental_refresh_constell`
`incremental_refresh_star_schema()`, `purge_dimensions_constellation()`, `purge_dimensions_star_schema()`

Examples

```
st <- st_mrs_age |>
  filter_fact_rows(name = "when", week <= "03") |>
  filter_fact_rows(name = "where", city == "Bridgeport")

st2 <- st_mrs_age |>
  incremental_refresh_star_schema(st, existing = "delete")
```

---

ft_datagov_uk *Modelling the long-term health impacts of air pollution in London*

Description

Estimation of the long-term health impacts of exposure to air pollution in London from 2016 to 2050.

Usage

```
ft_datagov_uk
```

Format

A `tibble`.

Details

The original dataset contains 68 files, corresponding to 34 London areas and 2 pollutants: pollutant and zone are indicated in the name of each file. Each file has several sheets with different variables. It has been transformed into a flat table considering a single variable and defining the area and the pollutant as columns.

Source

https://data.world/datagov-uk/fd864906-8456-46a8-9a01-0dcb2dbd87b9

ft_london_boroughs *London Boroughs*

Description

Classification of London's boroughs into zones and sub-regions.

Usage

```
ft_london_boroughs
```

Format

A tibble.

Source

https://en.wikipedia.org/wiki/List_of_sub-regions_used_in_the_London_Plan

ft_usa_city_county *USA City and County*

Description

City, state and county for US cities. It only includes those that appear in the Mortality Reporting System.

Usage

```
ft_usa_city_county
```

Format

A tibble.

Source

https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html

ft_usa_states *USA States*

Description

Name and abbreviation of US states.

Usage

```
ft_usa_states
```

Format

A tibble.

Source

https://www.census.gov/geographies/mapping-files/time-series/geo/carto-boundary-file.html

get_conformed_dimension *Get conformed dimension*

Description

Get a conformed dimension of a constellation given its name.

Usage

```
get_conformed_dimension(ct, name)

## S3 method for class 'constellation'
get_conformed_dimension(ct, name)
```

Arguments

ct	A constellation object.
name	A string, name of the dimension.

Value

A dimension_table object.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_dimension_names(), get_dimension(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(), record_update_set(), update_record(), update_selection_general(), update_selection()

Examples

```
d <- ct_mrs |>
  get_conformed_dimension("when")
```

---

get_conformed_dimension_names

Get conformed dimension names

Description

Get the names of the conformed dimensions of a constellation.

Usage

```
get_conformed_dimension_names(ct)

## S3 method for class 'constellation'
get_conformed_dimension_names(ct)
```

Arguments

ct	A constellation object.

Value

A vector of dimension names.

See Also

Other data cleaning functions: get_conformed_dimension(), get_dimension_names(), get_dimension(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(), record_update_set(), update_record(), update_selection_general(), update_selection()

Examples

```
d <- ct_mrs |>
  get_conformed_dimension_names()
```

get_dimension *Get dimension*

Description

Get a dimension of a star schema given its name.

Usage

```
get_dimension(st, name)

## S3 method for class 'star_schema'
get_dimension(st, name)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.

Details

Role dimensions can be obtained but not role playing dimensions. Role dimensions get their instances of role playing dimensions.

Value

A dimension_table object.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_conformed_dimension(), get_dimension_names(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(record_update_set(), update_record(), update_selection_general(), update_selection()

Examples

```
d <- st_mrs_age |>
  get_dimension("when")
```

get_dimension_attribute_names *Get dimension attribute names*

Description

Get the name of attributes in a dimension.

Usage

```
get_dimension_attribute_names(st, name)

## S3 method for class 'star_schema'
get_dimension_attribute_names(st, name)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.

Value

A vector of attribute names.

See Also

Other rename functions: get_measure_names(), rename_dimension_attributes(), rename_dimension(), rename_fact(), rename_measures()

Examples

```
attribute_names <-
  st_mrs_age |> get_dimension_attribute_names("when")
```

get_dimension_names *Get dimension names*

Description

Get the names of the dimensions of a star schema.

Usage

```
get_dimension_names(st)

## S3 method for class 'star_schema'
get_dimension_names(st)
```

Arguments

st	A star_schema object.

Details

Role playing dimensions are not considered.

Value

A vector of dimension names.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_conformed_dimension(), get_dimension(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(), record_update_set(), update_record(), update_selection_general(), update_selection()

Examples

```
dn <- st_mrs_age |>
  get_dimension_names()
```

---

get_measure_names *Get measure names*

Description

Get the name of measures in facts.

Usage

```
get_measure_names(st)

## S3 method for class 'star_schema'
get_measure_names(st)
```

Arguments

st	A star_schema object.

Value

A vector of measure names.

See Also

Other rename functions: `get_dimension_attribute_names()`, `rename_dimension_attributes()`, `rename_dimension()`, `rename_fact()`, `rename_measures()`

Examples

```
measure_names <-
  st_mrs_age |> get_measure_names()
```

`get_star_schema` *Get star schema*

Description

Get a star schema of a constellation given its name.

Usage

```
get_star_schema(ct, name)

## S3 method for class 'constellation'
get_star_schema(ct, name)
```

Arguments

`ct`	A `constellation` object.
`name`	A string, name of the star schema.

Value

A `dimension_table` object.

See Also

Other incremental refresh functions: `filter_fact_rows()`, `get_star_schema_names()`, `incremental_refresh_constel`
`incremental_refresh_star_schema()`, `purge_dimensions_constellation()`, `purge_dimensions_star_schema()`

Examples

```
d <- ct_mrs |>
  get_star_schema("mrs_age")
```

get_star_schema_names *Get star schema names*

Description

Get the names of the star schemas in a constellation.

Usage

```
get_star_schema_names(ct)

## S3 method for class 'constellation'
get_star_schema_names(ct)
```

Arguments

ct	A constellation object.

Value

A vector of star schema names.

See Also

Other incremental refresh functions: filter_fact_rows(), get_star_schema(), incremental_refresh_constellation(), incremental_refresh_star_schema(), purge_dimensions_constellation(), purge_dimensions_star_schema()

Examples

```
d <- ct_mrs |>
  get_star_schema_names()
```

incremental_refresh_constellation *Incrementally refresh a constellation with a star schema*

Description

Incrementally refresh a star schema in a constellation with the content of a new star schema that is integrated into the first.

Usage

```
incremental_refresh_constellation(ct, st, existing = "ignore")

## S3 method for class 'constellation'
incremental_refresh_constellation(ct, st, existing = "ignore")
```

Arguments

ct	A constellation object.
st	A star_schema object.
existing	A string, operation to be performed with records in the fact table whose keys match.

Details

Once the dimensions are integrated, if there are records in the fact table whose keys match the new ones, new ones can be ignored, they can be replaced by new ones, all of them can be grouped using the aggregation functions, or they can be deleted. Therefore, the possible values of the existing parameter are: "ignore", "replace", "group" or "delete".

Value

A constellation object.

See Also

Other incremental refresh functions: filter_fact_rows(), get_star_schema_names(), get_star_schema(), incremental_refresh_star_schema(), purge_dimensions_constellation(), purge_dimensions_star_schema()

Examples

```
ct <- ct_mrs |>
  incremental_refresh_constellation(st_mrs_age_w10, existing = "replace")

ct <- ct_mrs |>
  incremental_refresh_constellation(st_mrs_cause_w10, existing = "group")
```

---

match_records *Make a dimension record equal to another*

Description

For a dimension, given the primary key of two records, it adds an update to the set of updates that modifies the combination of values of the rest of attributes of the first record so that they become the same as those of the second.

Usage

```
match_records(updates, dimension, old, new)

## S3 method for class 'record_update_set'
match_records(updates, dimension, old, new)
```

Arguments

updates	A record_update_set object.
dimension	A dimension_table object, dimension to update.
old	A number, primary key of the record to update.
new	A number, primary key of the record from which the values are taken.

Details

Primary keys are only used to get the combination of values easily. The update is defined exclusively from the rest of values.

It is especially useful when it is detected that two records should be only one: Two have been generated due to some data error.

Value

A record_update_set object.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_conformed_dimension(), get_dimension_names(), get_dimension(), modify_conformed_dimension_records(), modify_dimension_records(record_update_set(), update_record(), update_selection_general(), update_selection()

Examples

```
dim_names <- st_mrs_age |>
    get_dimension_names()

where <- st_mrs_age |>
  get_dimension("where")

# head(where, 2)

updates <- record_update_set() |>
  match_records(dimension = where,
                old = 1,
                new = 2)
```

`modify_conformed_dimension_records` *Apply dimension record update operations to conformed dimensions*

Description

Given a list of dimension record update operations, they are applied on the conformed dimensions of the `constellation` object. Update operations must be defined with the set of functions available for that purpose.

Usage

```
modify_conformed_dimension_records(ct, updates = record_update_set())

## S3 method for class 'constellation'
modify_conformed_dimension_records(ct, updates = record_update_set())
```

Arguments

`ct`	A `constellation` object.
`updates`	A `record_update_set` object.

Details

When dimensions are defined, records can be detected that must be modified as part of the data cleaning process: frequently to unify two or more records due to data errors or missing data. This is not immediate because facts must be adapted to the new set of dimension instances.

This operation allows us to unify records and automatically propagate modifications to facts in star schemas.

Value

A `constellation` object.

See Also

Other data cleaning functions: `get_conformed_dimension_names()`, `get_conformed_dimension()`, `get_dimension_names()`, `get_dimension()`, `match_records()`, `modify_dimension_records()`, `record_update_set()`, `update_record()`, `update_selection_general()`, `update_selection()`

Examples

```
ct <- ct_mrs |>
  modify_conformed_dimension_records(updates_st_mrs_age)
```

`modify_dimension_records` *Apply dimension record update operations*

Description

Given a list of dimension record update operations, they are applied on the dimensions of the `star_schema` object. Update operations must be defined with the set of functions available for that purpose.

Usage

```
modify_dimension_records(st, updates = record_update_set())

## S3 method for class 'star_schema'
modify_dimension_records(st, updates = record_update_set())
```

Arguments

`st`	A `star_schema` object.
`updates`	A `record_update_set` object.

Details

When dimensions are defined, records can be detected that must be modified as part of the data cleaning process: frequently to unify two or more records due to data errors or missing data. This is not immediate because facts must be adapted to the new set of dimension instances.

This operation allows us to unify records and automatically propagate modifications to facts.

The list of update operations can be applied repeatedly to new data received to be incorporated into the `star_schema` object.

Value

A `star_schema` object.

See Also

Other data cleaning functions: `get_conformed_dimension_names()`, `get_conformed_dimension()`, `get_dimension_names()`, `get_dimension()`, `match_records()`, `modify_conformed_dimension_records()`, `record_update_set()`, `update_record()`, `update_selection_general()`, `update_selection()`

Examples

```
st <- st_mrs_age |>
  modify_dimension_records(updates_st_mrs_age)
```

mrs *Mortality Reporting System*

Description

Selection of data from the 122 Cities Mortality Reporting System, for the first 11 weeks of 1962.

Usage

```
mrs
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

mrs_age *Mortality Reporting System by Age*

Description

Selection of data from the 122 Cities Mortality Reporting System by age group, for the first 9 weeks of 1962.

Usage

```
mrs_age
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

---

`mrs_age_test` *Mortality Reporting System by Age Test*

Description

Selection of data from the 2 Cities Mortality Reporting System by age group, for the first 3 weeks of 1962.

Usage

```
mrs_age_test
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

---

`mrs_age_w10` *Mortality Reporting System by Age for Week 10*

Description

Selection of data from the 122 Cities Mortality Reporting System by age group, for week 10 of 1962. It also includes some isolated data from previous weeks that is supposed to be corrections for data errors.

Usage

```
mrs_age_w10
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

---

`mrs_age_w11` *Mortality Reporting System by Age for Week 11*

Description

Selection of data from the 122 Cities Mortality Reporting System by age group, for week 11 of 1962. It also includes some isolated data from previous weeks that is supposed to be corrections for data errors.

Usage

```
mrs_age_w11
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

mrs_age_w_test *Mortality Reporting System by Age for Week Test*

Description

Selection of data from the 3 Cities Mortality Reporting System by age group, for week 4 of 1962. It also includes some isolated data from previous weeks that is supposed to be corrections for data errors.

Usage

```
mrs_age_w_test
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

mrs_cause *Mortality Reporting System by Cause*

Description

Selection of data from the 122 Cities Mortality Reporting System by cause, for the first 9 weeks of 1962.

Usage

```
mrs_cause
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

---

mrs_cause_test *Mortality Reporting System by Cause Test*

Description

Selection of data from the 2 Cities Mortality Reporting System by cause, for the first 3 weeks of 1962.

Usage

```
mrs_cause_test
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

mrs_cause_w10 *Mortality Reporting System by Cause for Week 10*

Description

Selection of data from the 122 Cities Mortality Reporting System by cause, for week 10 of 1962. It also includes some isolated data from previous weeks that is supposed to be additional data not considered before.

Usage

```
mrs_cause_w10
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

mrs_cause_w11 *Mortality Reporting System by Cause for Week 11*

Description

Selection of data from the 122 Cities Mortality Reporting System by cause, for week 11 of 1962. It also includes some isolated data from previous weeks that is supposed to be additional data not considered before.

Usage

```
mrs_cause_w11
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

---

`mrs_cause_w_test` *Mortality Reporting System by Cause for Week Test*

Description

Selection of data from the 3 Cities Mortality Reporting System by cause, for week 4 of 1962. It also includes some isolated data from previous weeks that is supposed to be additional data not considered before.

Usage

```
mrs_cause_w_test
```

Format

A `tibble`.

Details

The original dataset begins in 1962. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included (i.e., the combination of age group and cause is not included). In the cause, only a distinction is made between pneumonia or influenza and others.

Two additional dates have been generated, which were not present in the original dataset.

Source

https://catalog.data.gov/dataset/deaths-in-122-u-s-cities-1962-2016-122-cities-mortality-reporting

ms_mrs *Multistar for Mortality Reporting System*

Description

Multistar for the Mortality Reporting System considering age and cause classification. It is the result obtained in the vignette.

Usage

```
ms_mrs
```

Format

A `multistar` object.

Examples

```
# Defined by:

ms_mrs <- ct_mrs |>
  constellation_as_multistar()
```

ms_mrs_test *Multistar for Mortality Reporting System Test*

Description

Multistar for the Mortality Reporting System considering age and cause classification data test.

Usage

```
ms_mrs_test
```

Format

A `multistar` object.

Examples

```
# Defined by:

ms_mrs_test <- ct_mrs_test |>
  constellation_as_multistar()
```

multistar_as_flat_table *Export a* multistar *as a flat table*

Description

We can obtain a flat table, implemented using a tibble, from a multistar (which can be the result of a query). If it only has one fact table, it is not necessary to provide its name.

Usage

```
multistar_as_flat_table(ms, fact = NULL)

## S3 method for class 'multistar'
multistar_as_flat_table(ms, fact = NULL)
```

Arguments

ms	A multistar object.
fact	A string, name of the fact.

Value

A tibble.

See Also

Other results export functions: constellation_as_multistar(), constellation_as_tibble_list(), star_schema_as_flat_table(), star_schema_as_multistar(), star_schema_as_tibble_list()

Examples

```
ft <- ms_mrs |>
  multistar_as_flat_table(fact = "mrs_age")

ms <- dimensional_query(ms_mrs) |>
  select_dimension(name = "where",
                   attributes = c("city", "state")) |>
  select_dimension(name = "when",
                   attributes = c("when_happened_year")) |>
  select_fact(name = "mrs_age",
              measures = c("n_deaths")) |>
  select_fact(
    name = "mrs_cause",
    measures = c("pneumonia_and_influenza_deaths", "other_deaths")
  ) |>
  filter_dimension(name = "when", when_happened_week <= "03") |>
```

```
  filter_dimension(name = "where", city == "Boston") |>
  run_query()

ft <- ms |>
  multistar_as_flat_table()
```

`purge_dimensions_constellation` *Purge dimensions in a constellation*

Description

Delete instances of dimensions not related to facts in a constellation.

Usage

```
purge_dimensions_constellation(ct)

## S3 method for class 'constellation'
purge_dimensions_constellation(ct)
```

Arguments

`ct`	A `constellation` object.

Value

A `constellation` object.

See Also

Other incremental refresh functions: `filter_fact_rows()`, `get_star_schema_names()`, `get_star_schema()`, `incremental_refresh_constellation()`, `incremental_refresh_star_schema()`, `purge_dimensions_star_schema()`

Examples

```
ct <- ct_mrs |>
  purge_dimensions_constellation()
```

purge_dimensions_star_schema *Purge dimensions*

Description

Delete instances of dimensions not related to facts in a star schema.

Usage

```
purge_dimensions_star_schema(st)

## S3 method for class 'star_schema'
purge_dimensions_star_schema(st)
```

Arguments

st	A star_schema object.

Value

A star_schema object.

See Also

Other incremental refresh functions: filter_fact_rows(), get_star_schema_names(), get_star_schema(), incremental_refresh_constellation(), incremental_refresh_star_schema(), purge_dimensions_constellatio

Examples

```
st <- st_mrs_age |>
  purge_dimensions_star_schema()
```

record_update_set record_update_set *S3 class*

Description

A record_update_set object is created. Stores updates on dimension records.

Usage

```
record_update_set()
```

Details

Each update is made up of a dimension name, an old value set, and a new value set.

When the update is applied, all the dimension records that have the combination of old values are modified with the new values provided.

Value

A `record_update_set` object.

See Also

Other data cleaning functions: `get_conformed_dimension_names()`, `get_conformed_dimension()`, `get_dimension_names()`, `get_dimension()`, `match_records()`, `modify_conformed_dimension_records()`, `modify_dimension_records()`, `update_record()`, `update_selection_general()`, `update_selection()`

Examples

```
updates <- record_update_set()
```

---

rename_dimension *Rename dimension*

Description

Set new name for a dimension.

Usage

```
rename_dimension(st, name, new_name)

## S3 method for class 'star_schema'
rename_dimension(st, name, new_name)
```

Arguments

`st`	A `star_schema` object.
`name`	A string, name of the dimension.
`new_name`	A string, new name of the dimension.

Value

A `star_schema` object.

See Also

Other rename functions: get_dimension_attribute_names(), get_measure_names(), rename_dimension_attributes(
rename_fact(), rename_measures()

Examples

```
st <- st_mrs_age |>
  rename_dimension(name = "when", new_name = "when_happened")
```

`rename_dimension_attributes`
Rename dimension attributes

Description

Set new names of some attributes in a dimension.

Usage

```
rename_dimension_attributes(st, name, attributes, new_names)

## S3 method for class 'star_schema'
rename_dimension_attributes(st, name, attributes, new_names)
```

Arguments

st	A star_schema object.
name	A string, name of the dimension.
attributes	A vector of attribute names.
new_names	A vector of new attribute names.

Value

A star_schema object.

See Also

Other rename functions: get_dimension_attribute_names(), get_measure_names(), rename_dimension(),
rename_fact(), rename_measures()

Examples

```
st <-
  st_mrs_age |> rename_dimension_attributes(
    name = "when",
    attributes = c("week", "year"),
    new_names = c("w", "y")
  )
```

`rename_fact` *Rename fact*

Description

Set new name for facts.

Usage

```
rename_fact(st, name)

## S3 method for class 'star_schema'
rename_fact(st, name)
```

Arguments

`st`	A `star_schema` object.
`name`	A string, new name of the fact.

Value

A `star_schema` object.

See Also

Other rename functions: `get_dimension_attribute_names()`, `get_measure_names()`, `rename_dimension_attributes()`, `rename_dimension()`, `rename_measures()`

Examples

```
st <- st_mrs_age |> rename_fact("age")
```

rename_measures *Rename measures*

Description

Set new names of some measures in facts.

Usage

```
rename_measures(st, measures, new_names)

## S3 method for class 'star_schema'
rename_measures(st, measures, new_names)
```

Arguments

st	A star_schema object.
measures	A vector of measure names.
new_names	A vector of new measure names.

Value

A star_schema object.

See Also

Other rename functions: get_dimension_attribute_names(), get_measure_names(), rename_dimension_attributes(
rename_dimension(), rename_fact()

Examples

```
st <-
  st_mrs_age |> rename_measures(measures = c("deaths"),
                                new_names = c("n_deaths"))
```

role_playing_dimension *Define a role playing dimension in a* star_schema *object*

Description

Given a list of star_schema dimension names, all with the same structure, a role playing dimension with the indicated name and attributes is generated. The original dimensions become role dimensions defined from the new role playing dimension.

Usage

```
role_playing_dimension(st, dim_names, name = NULL, attributes = NULL)

## S3 method for class 'star_schema'
role_playing_dimension(st, dim_names, name = NULL, attributes = NULL)
```

Arguments

st	A star_schema object.
dim_names	A vector of dimension names.
name	A string, name of the role playing dimension.
attributes	A vector of attribute names of the role playing dimension.

Details

After definition, all role dimensions have the same virtual instances (those of the role playing dimension). The foreign keys in facts are adapted to this new situation.

Value

A star_schema object.

See Also

Other star schema and constellation definition functions: character_dimensions(), constellation(), snake_case(), star_schema()

Examples

```
st <- star_schema(mrs_age, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("Date", "Week", "Year")
  )
```

```
st <- star_schema(mrs_cause, dm_mrs_cause) |>
  role_playing_dimension(
    dim_names = c("when", "when_received", "when_available"),
    name = "when_common",
    attributes = c("date", "week", "year")
  )
```

`run_query` *Run query*

Description

Once we have selected the facts, dimensions and defined the conditions on the instances, we can execute the query to obtain the result.

Usage

```
run_query(dq, unify_by_grain = TRUE)

## S3 method for class 'dimensional_query'
run_query(dq, unify_by_grain = TRUE)
```

Arguments

`dq`	A `dimensional_query` object.
`unify_by_grain`	A boolean, unify facts with the same grain.

Details

As an option, we can indicate if we do not want to unify the facts in the case of having the same grain.

Value

A `dimensional_query` object.

See Also

Other query functions: `dimensional_query()`, `filter_dimension()`, `select_dimension()`, `select_fact()`

Examples

```
ms <- dimensional_query(ms_mrs) |>
  select_dimension(name = "where",
                   attributes = c("city", "state")) |>
  select_dimension(name = "when",
                   attributes = c("when_happened_year")) |>
  select_fact(
    name = "mrs_age",
    measures = c("n_deaths"),
    agg_functions = c("MAX")
  ) |>
  select_fact(
    name = "mrs_cause",
    measures = c("pneumonia_and_influenza_deaths", "other_deaths")
  ) |>
  filter_dimension(name = "when", when_happened_week <= "03") |>
  filter_dimension(name = "where", city == "Boston") |>
  run_query()
```

`select_dimension` *Select dimension*

Description

To add a dimension in a `dimensional_query` object, we have to define its name and a subset of the dimension attributes. If only the name of the dimension is indicated, it is considered that all its attributes should be added.

Usage

```
select_dimension(dq, name = NULL, attributes = NULL)

## S3 method for class 'dimensional_query'
select_dimension(dq, name = NULL, attributes = NULL)
```

Arguments

`dq`	A `dimensional_query` object.
`name`	A string, name of the dimension.
`attributes`	A vector of attribute names.

Value

A `dimensional_query` object.

See Also

Other query functions: dimensional_query(), filter_dimension(), run_query(), select_fact()

Examples

```
dq <- dimensional_query(ms_mrs) |>
  select_dimension(name = "where",
                   attributes = c("city", "state")) |>
  select_dimension(name = "when")
```

`select_fact` *Select fact*

Description

To define the fact to be consulted, its name is indicated, optionally, a vector of names of selected measures and another of aggregation functions are also indicated.

Usage

```
select_fact(dq, name = NULL, measures = NULL, agg_functions = NULL)

## S3 method for class 'dimensional_query'
select_fact(dq, name = NULL, measures = NULL, agg_functions = NULL)
```

Arguments

`dq`	A `dimensional_query` object.
`name`	A string, name of the fact.
`measures`	A vector of measure names.
`agg_functions`	A vector of aggregation function names. If none is indicated, those defined in the fact table are considered.

Details

If the name of any measure is not indicated, only the one corresponding to the number of aggregated rows is included, which is always included.

If no aggregation function is included, those defined for the measures are considered.

Value

A `dimensional_query` object.

See Also

Other query functions: dimensional_query(), filter_dimension(), run_query(), select_dimension()

Examples

```
dq <- dimensional_query(ms_mrs) |>
  select_fact(
    name = "mrs_age",
    measures = c("n_deaths"),
    agg_functions = c("MAX")
  )

dq <- dimensional_query(ms_mrs) |>
  select_fact(name = "mrs_age",
              measures = c("n_deaths"))

dq <- dimensional_query(ms_mrs) |>
  select_fact(name = "mrs_age")
```

---

snake_case *Transform names according to the snake case style*

Description

Transform fact, dimension, measurement, and attribute names according to the snake case style.

Usage

```
snake_case(st)

## S3 method for class 'star_schema'
snake_case(st)
```

Arguments

st	A star_schema object.

Details

This style is suitable if we are going to work with databases.

Value

A star_schema object.

See Also

Other star schema and constellation definition functions: `character_dimensions()`, `constellation()`, `role_playing_dimension()`, `star_schema()`

Examples

```
st <- star_schema(mrs_age, dm_mrs_age) |>
  snake_case()

st <- star_schema(mrs_age, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("Date", "Week", "Year")
  ) |>
  snake_case()
```

`starschemar` *Obtaining Star Schemas from Flat Tables*

Description

Transformations that allow obtaining star schemas from flat tables.

Details

From flat tables star schemas can be defined that can form constellations (*star schema and constellation definition functions*). Dimensions contain data without duplicates, operations to do data cleaning can be applied on them (*data cleaning functions*). Dimensions can be enriched by adding additional columns, sometimes using functions, others explicitly defined by the user (*dimension enrichment functions*). When new data is obtained, it is necessary to refresh the existing data with them by means of incremental refresh operations or delete data that is no longer necessary (*incremental refresh functions*). Finally, the results obtained can be exported to be consulted with other tools (*results export functions*) or through the defined query functions (*query functions*).

Star schema and constellation definition

Starting from a flat table, a dimensional model is defined specifying the attributes that make up each of the dimensions and the measurements in the facts. The result is a `dimensional_model` object. It is carried out through the following *dimensional model definition functions*:

- `dimensional_model()`
- `define_dimension()`
- `define_fact()`

A star schema is defined from a flat table and a dimensional model definition. Once defined, a star schema can be transformed by defining role playing dimensions, changing the writing style of element names or the type of dimension attributes. These operations are carried out through the following *star schema definition and transformation functions*:

- `star_schema()`
- `role_playing_dimension()`
- `snake_case()`
- `character_dimensions()`

Once a star schema is defined, we can rename its elements. It is necessary to be able to rename attributes of dimensions and measures of facts because the definition operations only allowed us to select columns of a flat table. For completeness also dimensions and facts can be renamed. To carry out these operations, the following *star schema rename functions* are available:

- `rename_dimension()`
- `get_dimension_attribute_names()`
- `rename_dimension_attributes()`
- `rename_fact()`
- `get_measure_names()`
- `rename_measures()`

Based on various star schemas, a constellation can be defined in which star schemas share common dimensions. Dimensions with the same name must be shared. It is defined by the following *constellation definition function*:

- `constellation()`

Data cleaning

Once the star schemas and constellations are defined, data cleaning operations can be carried out on dimensions. There are three groups of functions: one to obtain dimensions of star schemas and constellations; another to define data cleaning operations over dimensions; and one more to apply operations to star schemas or constellations.

Obtaining dimensions:

- `get_dimension_names()`
- `get_dimension()`
- `get_conformed_dimension_names()`
- `get_conformed_dimension()`

Update definition functions:

- `record_update_set()`
- `match_records()`
- `update_record()`
- `update_selection()`

- update_selection_general()

Modification application functions:

- modify_dimension_records()
- modify_conformed_dimension_records()

Dimension enrichment

To enrich a dimension with new attributes related to others already included in it, first, we export the attributes on which the new ones depend, then we define the new attributes, and import the table with all the attributes to be added to the dimension.

- enrich_dimension_export()
- enrich_dimension_import()

Incremental refresh

When new data is obtained, an incremental refresh of the data can be carried out, both of the dimensions and of the facts. Incremental refresh can be applied to both star schema and constellation, using the following functions:

- incremental_refresh_star_schema()
- incremental_refresh_constellation()

Sometimes the data refresh consists of eliminating data that is no longer necessary, generally because it corresponds to a period that has stopped being analysed but it can also be for other reasons. This data can be selected using the following function:

- filter_fact_rows()

Once the fact data is removed (using the other incremental refresh functions), we can remove the data for the dimensions that are no longer needed using the following functions:

- purge_dimensions_star_schema()
- purge_dimensions_constellation()

Results export

Once the data has been properly structured and transformed, it can be exported to be consulted with other tools or with R. Various export formats have been defined, both for star schemas and for constellations, using the following functions:

- star_schema_as_flat_table()
- star_schema_as_multistar()
- star_schema_as_tibble_list()
- constellation_as_multistar()
- constellation_as_tibble_list()
- multistar_as_flat_table()

Query functions

There are many multidimensional query tools available. The exported data, once stored in files, can be used directly from them. Using the following functions, you can also perform basic queries from R on data in the `multistar` format:

- `dimensional_query()`
- `select_fact()`
- `select_dimension()`
- `filter_dimension()`
- `run_query()`

`star_schema` `star_schema` *S3 class*

Description

Creates a `star_schema` object from a flat table (implemented by a `tibble`) and a `dimensional_model` object.

Usage

```
star_schema(ft, sd)
```

Arguments

`ft`	A `tibble`, implements a flat table.
`sd`	A `dimensional_model` object.

Details

Transforms the flat table data according to the facts and dimension definitions of the `dimensional_model` object. Each dimension is generated with a surrogate key which is a foreign key in facts.

Facts only contain measurements and foreign keys.

Value

A `star_schema` object.

See Also

`dimensional_model`

Other star schema and constellation definition functions: `character_dimensions()`, `constellation()`, `role_playing_dimension()`, `snake_case()`

Examples

```
st <- star_schema(mrs_age, dm_mrs_age)
```

`star_schema_as_flat_table` *Export a star schema as a flat table*

Description

Once we have refined the format or content of facts and dimensions, we can again obtain a flat table, implemented using a `tibble`, from a star schema.

Usage

```
star_schema_as_flat_table(st)

## S3 method for class 'star_schema'
star_schema_as_flat_table(st)
```

Arguments

`st`	A `star_schema` object.

Value

A `tibble`.

See Also

Other results export functions: `constellation_as_multistar()`, `constellation_as_tibble_list()`, `multistar_as_flat_table()`, `star_schema_as_multistar()`, `star_schema_as_tibble_list()`

Examples

```
ft <- st_mrs_age |>
  star_schema_as_flat_table()
```

star_schema_as_multistar

Export a star schema as a `multistar`

Description

Once we have refined the format or content of facts and dimensions, we can obtain a `multistar`. A `multistar` only distinguishes between general and conformed dimensions, each dimension has its own data. It can contain multiple fact tables.

Usage

```
star_schema_as_multistar(st)

## S3 method for class 'star_schema'
star_schema_as_multistar(st)
```

Arguments

`st`	A `star_schema` object.

Value

A `multistar` object.

See Also

Other results export functions: `constellation_as_multistar()`, `constellation_as_tibble_list()`, `multistar_as_flat_table()`, `star_schema_as_flat_table()`, `star_schema_as_tibble_list()`

Examples

```
ms <- st_mrs_age |>
  star_schema_as_multistar()
```

star_schema_as_tibble_list

Export a star schema as a `tibble` *list*

Description

Once we have refined the format or content of facts and dimensions, we can obtain a `tibble` list with them. Role playing dimensions can be optionally included.

Usage

```
star_schema_as_tibble_list(st, include_role_playing = FALSE)

## S3 method for class 'star_schema'
star_schema_as_tibble_list(st, include_role_playing = FALSE)
```

Arguments

st	A star_schema object.
include_role_playing	A boolean.

Value

A list of tibble objects.

See Also

Other results export functions: constellation_as_multistar(), constellation_as_tibble_list(), multistar_as_flat_table(), star_schema_as_flat_table(), star_schema_as_multistar()

Examples

```
tl <- st_mrs_age |>
  star_schema_as_tibble_list()

tl <- st_mrs_age |>
  star_schema_as_tibble_list(include_role_playing = TRUE)
```

---

st_mrs_age *Star Schema for Mortality Reporting System by Age*

Description

Star Schema for the Mortality Reporting System considering the age classification.

Usage

```
st_mrs_age
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_age <- star_schema(mrs_age, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("date", "week", "year")
  ) |>
  snake_case() |>
  character_dimensions(NA_replacement_value = "Unknown",
                       length_integers = list(week = 2))
```

`st_mrs_age_test` *Star Schema for Mortality Reporting System by Age Test*

Description

Star Schema for the Mortality Reporting System considering the age classification data test.

Usage

```
st_mrs_age_test
```

Format

A `star_schema` object.

Examples

```
# Defined by:

st_mrs_age_test <- star_schema(mrs_age_test, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("date", "week", "year")
  ) |>
  snake_case() |>
  character_dimensions(NA_replacement_value = "Unknown",
                       length_integers = list(week = 2))
```

st_mrs_age_w10 *Star Schema for Mortality Reporting System by Age for Week 10*

Description

Star Schema for the Mortality Reporting System considering the age classification data, for week 10 of 1962. It also includes some isolated data from previous weeks that is supposed to be corrections for data errors.

Usage

```
st_mrs_age_w10
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_age_w10 <- star_schema(mrs_age_w10, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("date", "week", "year")
  ) |>
  snake_case() |>
  character_dimensions(NA_replacement_value = "Unknown",
                       length_integers = list(week = 2))
```

st_mrs_age_w11 *Star Schema for Mortality Reporting System by Age for Week 11*

Description

Star Schema for the Mortality Reporting System considering the age classification data, for week 11 of 1962. It also includes some isolated data from previous weeks that is supposed to be corrections for data errors.

Usage

```
st_mrs_age_w11
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_age_w11 <- star_schema(mrs_age_w11, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("date", "week", "year")
  ) |>
  snake_case() |>
  character_dimensions(NA_replacement_value = "Unknown",
                       length_integers = list(week = 2))
```

---

st_mrs_age_w_test *Star Schema for Mortality Reporting System by Age for Week Test*

Description

Star Schema for the Mortality Reporting System considering the age classification data test, for week 4 of 1962. It also includes some isolated data from previous weeks that is supposed to be corrections for data errors.

Usage

```
st_mrs_age_w_test
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_age_w_test <- star_schema(mrs_age_w_test, dm_mrs_age) |>
  role_playing_dimension(
    dim_names = c("when", "when_available"),
    name = "When Common",
    attributes = c("date", "week", "year")
  ) |>
  snake_case() |>
  character_dimensions(NA_replacement_value = "Unknown",
                       length_integers = list(week = 2))
```

st_mrs_cause *Star Schema for Mortality Reporting System by Cause*

Description

Star Schema for the Mortality Reporting System considering the cause classification.

Usage

```
st_mrs_cause
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_cause <- star_schema(mrs_cause, dm_mrs_cause) |>
  snake_case() |>
  character_dimensions(
    NA_replacement_value = "Unknown",
    length_integers = list(
      week = 2,
      data_availability_week = 2,
      reception_week = 2
    )
  ) |>
  role_playing_dimension(
    dim_names = c("when", "when_received", "when_available"),
    name = "when_common",
    attributes = c("date", "week", "year")
  )
```

st_mrs_cause_test *Star Schema for Mortality Reporting System by Cause Test*

Description

Star Schema for the Mortality Reporting System considering the cause classification data test.

Usage

```
st_mrs_cause_test
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_cause_test <- star_schema(mrs_cause_test, dm_mrs_cause) |>
  snake_case() |>
  character_dimensions(
    NA_replacement_value = "Unknown",
    length_integers = list(
      week = 2,
      data_availability_week = 2,
      reception_week = 2
    )
  ) |>
  role_playing_dimension(
    dim_names = c("when", "when_received", "when_available"),
    name = "when_common",
    attributes = c("date", "week", "year")
  )
```

---

st_mrs_cause_w10 *Star Schema for Mortality Reporting System by Cause for Week 10*

Description

Star Schema for the Mortality Reporting System considering the cause classification data, for week 10 of 1962. It also includes some isolated data from previous weeks that is supposed to be additional data not considered before.

Usage

```
st_mrs_cause_w10
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_cause_w10 <- star_schema(mrs_cause_w10, dm_mrs_cause) |>
  snake_case() |>
  character_dimensions(
    NA_replacement_value = "Unknown",
```

```
    length_integers = list(
      week = 2,
      data_availability_week = 2,
      reception_week = 2
    )
  ) |>
  role_playing_dimension(
    dim_names = c("when", "when_received", "when_available"),
    name = "when_common",
    attributes = c("date", "week", "year")
  )
```

---

st_mrs_cause_w11 *Star Schema for Mortality Reporting System by Cause for Week 11*

Description

Star Schema for the Mortality Reporting System considering the cause classification data, for week 11 of 1962. It also includes some isolated data from previous weeks that is supposed to be additional data not considered before.

Usage

```
st_mrs_cause_w11
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_cause_w11 <- star_schema(mrs_cause_w11, dm_mrs_cause) |>
  snake_case() |>
  character_dimensions(
    NA_replacement_value = "Unknown",
    length_integers = list(
      week = 2,
      data_availability_week = 2,
      reception_week = 2
    )
  ) |>
  role_playing_dimension(
    dim_names = c("when", "when_received", "when_available"),
    name = "when_common",
    attributes = c("date", "week", "year")
  )
```

st_mrs_cause_w_test *Star Schema for Mortality Reporting System by Cause for Week Test*

Description

Star Schema for the Mortality Reporting System considering the cause classification data test, for week 4 of 1962. It also includes some isolated data from previous weeks that is supposed to be additional data not considered before.

Usage

```
st_mrs_cause_w_test
```

Format

A star_schema object.

Examples

```
# Defined by:

st_mrs_cause_w_test <- star_schema(mrs_cause_w_test, dm_mrs_cause) |>
  snake_case() |>
  character_dimensions(
    NA_replacement_value = "Unknown",
    length_integers = list(
      week = 2,
      data_availability_week = 2,
      reception_week = 2
    )
  ) |>
  role_playing_dimension(
    dim_names = c("when", "when_received", "when_available"),
    name = "when_common",
    attributes = c("date", "week", "year")
  )
```

updates_st_mrs_age *Updates for the Star Schema for Mortality Reporting System by Age*

Description

Example of updates on some dimensions of the star schema for Mortality Reporting System by age.

Usage

```
updates_st_mrs_age
```

Format

A record_update_set object.

Examples

```
# Defined by:

(dim_names <- st_mrs_age |>
    get_dimension_names())

where <- st_mrs_age |>
  get_dimension("where")

when <- st_mrs_age |>
  get_dimension("when")

who <- st_mrs_age |>
  get_dimension("who")

updates_st_mrs_age <- record_update_set() |>
  update_selection_general(
    dimension = where,
    columns_old = c("state", "city"),
    old_values = c("CT", "Bridgepor"),
    columns_new = c("city"),
    new_values = c("Bridgeport")
  ) |>
  match_records(dimension = when,
                old = 37,
                new = 36) |>
  update_record(
    dimension = when,
    old = 73,
    values = c("1962-02-17", "07", "1962")
  ) |>
  update_selection(
    dimension = who,
    columns = c("age_range"),
    old_values = c("<1 year"),
    new_values = c("1: <1 year")
  ) |>
  update_selection(
    dimension = who,
    columns = c("age_range"),
    old_values = c("1-24 years"),
    new_values = c("2: 1-24 years")
  ) |>
  update_selection(
    dimension = who,
    columns = c("age_range"),
    old_values = c("25-44 years"),
    new_values = c("3: 25-44 years")
```

```
  ) |>
  update_selection(
    dimension = who,
    columns = c("age_range"),
    old_values = c("45-64 years"),
    new_values = c("4: 45-64 years")
  ) |>
  update_selection(
    dimension = who,
    columns = c("age_range"),
    old_values = c("65+ years"),
    new_values = c("5: 65+ years")
  )
```

`updates_st_mrs_age_test` *Updates for the Star Schema for Mortality Reporting System by Age Test*

Description

Example of updates on some dimensions of the star schema for Mortality Reporting System by age test.

Usage

```
updates_st_mrs_age_test
```

Format

A `record_update_set` object.

Examples

```
# Defined by:

(dim_names <- st_mrs_age_test |>
    get_dimension_names())

where <- st_mrs_age_test |>
  get_dimension("where")

when <- st_mrs_age_test |>
  get_dimension("when")

who <- st_mrs_age_test |>
  get_dimension("who")

updates_st_mrs_age_test <- record_update_set() |>
```

```
update_selection_general(
  dimension = where,
  columns_old = c("state", "city"),
  old_values = c("CT", "Bridgepor"),
  columns_new = c("city"),
  new_values = c("Bridgeport")
) |>
match_records(dimension = when,
              old = 4,
              new = 3) |>
update_record(
  dimension = when,
  old = 9,
  values = c("1962-01-20", "03", "1962")
) |>
update_selection(
  dimension = who,
  columns = c("age_range"),
  old_values = c("<1 year"),
  new_values = c("1: <1 year")
) |>
update_selection(
  dimension = who,
  columns = c("age_range"),
  old_values = c("1-24 years"),
  new_values = c("2: 1-24 years")
) |>
update_selection(
  dimension = who,
  columns = c("age_range"),
  old_values = c("25-44 years"),
  new_values = c("3: 25-44 years")
) |>
update_selection(
  dimension = who,
  columns = c("age_range"),
  old_values = c("45-64 years"),
  new_values = c("4: 45-64 years")
) |>
update_selection(
  dimension = who,
  columns = c("age_range"),
  old_values = c("65+ years"),
  new_values = c("5: 65+ years")
)
```

---

update_record *Update a dimension record with a set of values*

Description

For a dimension, given the primary key of one record, it adds an update to the set of updates that modifies the combination of values of the rest of attributes of the selected record so that they become those given.

Usage

```
update_record(updates = NULL, dimension, old, values = vector())

## S3 method for class 'record_update_set'
update_record(updates = NULL, dimension, old, values = vector())
```

Arguments

updates	A record_update_set object.
dimension	A dimension_table object, dimension to update.
old	A number, primary key of the record to modify.
values	A vector of character values.

Details

Primary key is only used to get the combination of values easily. The update is defined exclusively from the rest of values.

Value

A record_update_set object.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_conformed_dimension(), get_dimension_names(), get_dimension(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(), record_update_set(), update_selection_general(), update_selection()

Examples

```
dim_names <- st_mrs_age |>
    get_dimension_names()

where <- st_mrs_age |>
  get_dimension("where")

# head(where, 2)

updates <- record_update_set() |>
  update_record(
    dimension = where,
    old = 1,
```

```
    values = c("1", "CT", "Bridgeport")
  )
```

`update_selection` *Update dimension records with a set of values*

Description

For a dimension, given a vector of column names, a vector of old values and a vector of new values, it adds an update to the set of updates that modifies all the records that have the combination of old values in the columns with the new values in those same columns.

Usage

```
update_selection(
  updates = NULL,
  dimension,
  columns = vector(),
  old_values = vector(),
  new_values = vector()
)

## S3 method for class 'record_update_set'
update_selection(
  updates = NULL,
  dimension,
  columns = vector(),
  old_values = vector(),
  new_values = vector()
)
```

Arguments

`updates`	A `record_update_set` object.
`dimension`	A `dimension_table` object, dimension to update.
`columns`	A vector of column names.
`old_values`	A vector of character values.
`new_values`	A vector of character values.

Value

A `record_update_set` object.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_conformed_dimension(), get_dimension_names(), get_dimension(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(), record_update_set(), update_record(), update_selection_general()

Examples

```
dim_names <- st_mrs_age |>
    get_dimension_names()

where <- st_mrs_age |>
  get_dimension("where")

# head(where, 2)

updates <- record_update_set() |>
  update_selection(
    dimension = where,
    columns = c("city"),
    old_values = c("Bridgepor"),
    new_values = c("Bridgeport")
  )
```

---

update_selection_general

Update dimension records with a set of values in given columns

---

Description

For a dimension, given a vector of column names, a vector of old values for those columns, another vector column names, and a vector of new values for those columns, it adds an update to the set of updates that modifies all the records that have the combination of old values in the first column vector with the new values in the second column vector.

Usage

```
update_selection_general(
  updates = NULL,
  dimension,
  columns_old = vector(),
  old_values = vector(),
  columns_new = vector(),
  new_values = vector()
)

## S3 method for class 'record_update_set'
```

```
update_selection_general(
  updates = NULL,
  dimension,
  columns_old = vector(),
  old_values = vector(),
  columns_new = vector(),
  new_values = vector()
)
```

Arguments

updates	A record_update_set object.
dimension	A dimension_table object, dimension to update.
columns_old	A vector of column names.
old_values	A vector of character values.
columns_new	A vector of column names.
new_values	A vector of character values.

Value

A record_update_set object.

See Also

Other data cleaning functions: get_conformed_dimension_names(), get_conformed_dimension(), get_dimension_names(), get_dimension(), match_records(), modify_conformed_dimension_records(), modify_dimension_records(), record_update_set(), update_record(), update_selection()

Examples

```
dim_names <- st_mrs_age |>
    get_dimension_names()

where <- st_mrs_age |>
  get_dimension("where")

# head(where, 2)

updates <- record_update_set() |>
  update_selection_general(
    dimension = where,
    columns_old = c("state", "city"),
    old_values = c("CT", "Bridgepor"),
    columns_new = c("city"),
    new_values = c("Bridgeport")
  )
```

Index