Package 'rolap'

July 23, 2025

Title Obtaining Star Databases from Flat Tables

Version 2.5.2

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 ROLAP (Relational On-Line Analytical Processing) star database. The main objective of the package is to allow the definition of these transformations easily. The implementation of the multidimensional database obtained can be exported to work with multidimensional analysis tools on spreadsheets or relational databases.

License MIT + file LICENSE

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

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

Depends R (>= 4.1.0)

Imports dm, dplyr, methods, purrr, readr, rlang, sf, snakecase, tibble, tidyr, tidyselect, tools, utils, when, xlsx

Suggests DBI, dbplyr, DiagrammeR, DiagrammeRsvg, knitr, lubridate, magrittr, maps, pander, pivottabler, RMariaDB, rmarkdown, RSQLite, stringr, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.2

NeedsCompilation no

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

Maintainer Jose Samos <jsamos@ugr.es>

Repository CRAN

Date/Publication 2025-05-22 05:10:02 UTC

Contents

`add_custom_column` *Add custom column*

Description

Add a column returned by a function that takes the data of the flat table as a parameter.

Usage

```
add_custom_column(ft, name, definition)

## S3 method for class 'flat_table'
add_custom_column(ft, name = NULL, definition)
```

Arguments

`ft`	A `flat_table` object.
`name`	A string, new column name.
`definition`	A function that returns a table column.

Value

A `flat_table` object.

See Also

flat_table

Other flat table transformation functions: remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
f <- function(table) {
  paste0(table$City, ' - ', table$State)
}

ft <- flat_table('ft_num', ft_num) |>
  add_custom_column(name = 'city_state', definition = f)
```

---

as_csv_files *Generate csv files with fact and dimension tables*

Description

To port databases to other work environments it is useful to be able to export them as csv files, as this function does.

Usage

```
as_csv_files(db, dir, type)

## S3 method for class 'star_database'
as_csv_files(db, dir = NULL, type = 1)
```

Arguments

db	A star_database object.
dir	A string, name of a dir.
type	An integer, 1: uses "." for the decimal point and a comma for the separator; 2: uses a comma for the decimal point and a semicolon for the separator.

Value

A string, name of a dir.

See Also

star_database

Other star database exportation functions: as_dm_class(), as_multistar(), as_rdb(), as_single_tibble_list(), as_tibble_list(), as_xlsx_file(), draw_tables()

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
tl1 <- db1 |>
  as_csv_files()

db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()

ct <- constellation("MRS", db1, db2)
d <- ct |>
  as_csv_files(dir = tempdir())
```

---

as_dm_class *Generate a* dm *class with fact and dimension tables*

Description

To port databases to other work environments it is useful to be able to export them as a dm class, as this function does, in this way it can be saved directly in a DBMS.

Usage

```
as_dm_class(db, pk_facts, fk)

## S3 method for class 'star_database'
as_dm_class(db, pk_facts = TRUE, fk = TRUE)
```

Arguments

db	A star_database object.
pk_facts	A boolean, include primary key in fact tables.
fk	A boolean, include foreign key in fact tables.

Value

A dm object.

See Also

star_database

Other star database exportation functions: as_csv_files(), as_multistar(), as_rdb(), as_single_tibble_list(), as_tibble_list(), as_xlsx_file(), draw_tables()

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
dm1 <- db1 |>
  as_dm_class()

db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()

ct <- constellation("MRS", db1, db2)
dm <- ct |>
  as_dm_class()
```

---

as_geolayer *Get a* geolayer *object*

Description

From a star_database with at least one geoattribute, we obtain a geolayer object that allows us to select the data to obtain a vector layer with geographic information.

Usage

```
as_geolayer(db, dimension, attribute, geometry, include_nrow_agg)

## S3 method for class 'star_database'
as_geolayer(
  db,
  dimension = NULL,
  attribute = NULL,
  geometry = NULL,
  include_nrow_agg = FALSE
)
```

Arguments

db	An star_database object.
dimension	A string, dimension name.
attribute	A vector, attribute names.
geometry	A string, geometry name.
include_nrow_agg	A boolean, include default measure.

Details

If only one geographic attribute is defined, it is not necessary to indicate the dimension or the attribute. By default, polygon geometry is considered.

Value

A geolayer object.

See Also

Other query functions: as_GeoPackage(), filter_dimension(), get_layer(), get_variable_description(), get_variables(), run_query(), select_dimension(), select_fact(), set_layer(), set_variables(), star_query()

Examples

```
gl_polygon <- mrs_db_geo |>
  as_geolayer()

gl_point <- mrs_db_geo |>
  as_geolayer(geometry = "point")
```

---

as_GeoPackage *Save as* GeoPackage

Description

Save the geolayer (geographic information layer) and the variables layer in a file in GeoPackage format to be able to work with other tools.

Usage

```
as_GeoPackage(gl, dir, name, keep_all_variables_na)

## S3 method for class 'geolayer'
as_GeoPackage(gl, dir = NULL, name = NULL, keep_all_variables_na = FALSE)
```

Arguments

gl	A geolayer object.
dir	A string.
name	A string, file name.
keep_all_variables_na	A boolean, keep rows with all variables NA.

Details

If the file name is not indicated, it defaults to the name of the geovariable.

By default, rows that are NA for all variables are eliminated.

The GeoPackage format only allows defining a maximum of 1998 columns. If the number of variables and columns in the geographic layer exceeds this number, it cannot be saved in this format.

Value

A string, file name.

See Also

Other query functions: as_geolayer(), filter_dimension(), get_layer(), get_variable_description(), get_variables(), run_query(), select_dimension(), select_fact(), set_layer(), set_variables(), star_query()

Examples

```
gl <- mrs_db_geo |>
  as_geolayer()

f <- gl |>
  as_GeoPackage(dir = tempdir())
```

---

as_multistar *Generate a* geomultistar::multistar *object*

Description

In order to be able to use the query and integration functions with geographic information offered by the geomultistar package, we can obtain a multistar object from a star database or a constellation.

Usage

```
as_multistar(db)

## S3 method for class 'star_database'
as_multistar(db)
```

Arguments

db	A star_database object.

Value

A geomultistar::multistar object.

See Also

star_database

Other star database exportation functions: as_csv_files(), as_dm_class(), as_rdb(), as_single_tibble_list(), as_tibble_list(), as_xlsx_file(), draw_tables()

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
ms1 <- db1 |>
  as_multistar()

db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()

ct <- constellation("MRS", db1, db2)
ms <- ct |>
  as_multistar()
```

---

as_rdb *Generate tables in a relational database*

Description

Given a connection to a relational database, it stores the facts and dimensions in the form of tables. Tables can be overwritten.

Usage

```
as_rdb(db, con, overwrite)

## S3 method for class 'star_database'
as_rdb(db, con, overwrite = FALSE)
```

Arguments

db	A star_database object.
con	A DBI::DBIConnection object.
overwrite	A boolean, allow overwriting tables in the database.

Value

Invisible NULL.

See Also

star_database

Other star database exportation functions: as_csv_files(), as_dm_class(), as_multistar(), as_single_tibble_list(), as_tibble_list(), as_xlsx_file(), draw_tables()

Examples

```
my_db <- DBI::dbConnect(RSQLite::SQLite())

db <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
db |>
  as_rdb(my_db)

DBI::dbDisconnect(my_db)
```

---

as_single_tibble_list *Generate a list of tibbles of flat tables*

Description

Allows you to transform a star database into a flat table. If we have a constellation, it returns a list of flat tables.

Usage

```
as_single_tibble_list(db)

## S3 method for class 'star_database'
as_single_tibble_list(db)
```

Arguments

db	A star_database object.

Value

A list of tibble

See Also

star_database

Other star database exportation functions: as_csv_files(), as_dm_class(), as_multistar(), as_rdb(), as_tibble_list(), as_xlsx_file(), draw_tables()

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
tl1 <- db1 |>
  as_single_tibble_list()

db2 <- star_database(mrs_age_schema, ft_age) |>
```

```
  snake_case()

ct <- constellation("MRS", db1, db2)
tl <- ct |>
  as_single_tibble_list()
```

---

as_star_database *Get a star database from a flat table*

Description

Obtain a star database from the flat table and a star schema.

Usage

```
as_star_database(ft, schema)

## S3 method for class 'flat_table'
as_star_database(ft, schema)
```

Arguments

ft	A flat_table object.
schema	A star_schema object.

Value

A star_database object.

See Also

star_database

Other flat table definition functions: flat_table(), get_table(), get_unknown_value_defined(), get_unknown_values(), read_flat_table_file(), read_flat_table_folder()

Examples

```
db <- flat_table('ft_num', ft_num) |>
  as_star_database(mrs_cause_schema)
```

`as_tibble_list` *Generate a list of tibbles with fact and dimension tables*

Description

To port databases to other work environments it is useful to be able to export them as a list of tibbles, as this function does.

Usage

```
as_tibble_list(db)

## S3 method for class 'star_database'
as_tibble_list(db)
```

Arguments

db	A `star_database` object.

Value

A list of `tibble`

See Also

`star_database`

Other star database exportation functions: `as_csv_files()`, `as_dm_class()`, `as_multistar()`, `as_rdb()`, `as_single_tibble_list()`, `as_xlsx_file()`, `draw_tables()`

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
tl1 <- db1 |>
  as_tibble_list()

db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()

ct <- constellation("MRS", db1, db2)
tl <- ct |>
  as_tibble_list()
```

as_xlsx_file *Generate a xlsx file with fact and dimension tables*

Description

To port databases to other work environments it is useful to be able to export them as a xlsx file, as this function does.

Usage

```
as_xlsx_file(db, file)

## S3 method for class 'star_database'
as_xlsx_file(db, file = NULL)
```

Arguments

db	A star_database object.
file	A string, name of a file.

Value

A string, name of a file.

See Also

star_database

Other star database exportation functions: as_csv_files(), as_dm_class(), as_multistar(), as_rdb(), as_single_tibble_list(), as_tibble_list(), draw_tables()

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
tl1 <- db1 |>
  as_xlsx_file()

db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()

ct <- constellation("MRS", db1, db2)
f <- ct |>
  as_xlsx_file(file = tempfile())
```

`cancel_deployment` *Cancel deployment*

Description

Cancel deployment

Usage

```
cancel_deployment(db, name)

## S3 method for class 'star_database'
cancel_deployment(db, name)
```

Arguments

db	A `star_database` object.
name	A string, name of the deployment.

Value

A `star_database` object.

See Also

`star_database`

Other star database deployment functions: `deploy()`, `get_deployment_names()`, `load_star_database()`

Examples

```
mrs_rdb_file <- tempfile("mrs", fileext = ".rdb")
mrs_sqlite_file <- tempfile("mrs", fileext = ".sqlite")

mrs_sqlite_connect <- function() {
  DBI::dbConnect(RSQLite::SQLite(),
                 dbname = mrs_sqlite_file)
}

mrs_db <- mrs_db |>
  deploy(
    name = "mrs",
    connect = mrs_sqlite_connect,
    file = mrs_rdb_file
  )

mrs_db <- mrs_db |>
  cancel_deployment(name = "mrs")
```

check_geoattribute_geometry *Check a* geoattribute *geometry instances.*

Description

Get unrelated instances of a geoattribute for a geometry.

Usage

```
check_geoattribute_geometry(db, dimension, attribute, geometry)

## S3 method for class 'star_database'
check_geoattribute_geometry(
  db,
  dimension = NULL,
  attribute = NULL,
  geometry = "polygon"
)
```

Arguments

db	A star_database object.
dimension	A string, dimension name.
attribute	A vector, attribute names.
geometry	A string, geometry name ('point' or 'polygon').

Details

We obtain the values of the dimension attribute that do not have an associated geographic element of the indicated geometry.

If there is only one geoattribute defined, neither the dimension nor the attribute must be indicated.

Value

A tibble.

See Also

Other star database geographic attributes: define_geoattribute(), get_geoattribute_geometries(), get_geoattributes(), get_layer_geometry(), get_point_geometry(), summarize_layer()

Examples

```
db <- mrs_db |>
  define_geoattribute(
    dimension = "where",
    attribute = "state",
    from_layer = us_layer_state,
    by = "STUSPS"
  )

instances <- check_geoattribute_geometry(db,
                                         dimension = "where",
                                         attribute = "state")
```

---

check_lookup_table *Check the result of joining a flat table with a lookup table*

Description

Before joining a flat table with a lookup table we can check the result to determine if we need to adapt the values of some instances or add new elements to the lookup table. This function returns the values of the foreign key of the flat table that do not correspond to the primary key of the lookup table.

Usage

```
check_lookup_table(ft, fk_attributes, lookup)

## S3 method for class 'flat_table'
check_lookup_table(ft, fk_attributes = NULL, lookup)
```

Arguments

`ft`	A `flat_table` object.
`fk_attributes`	A vector of strings, attribute names.
`lookup`	A `flat_table` object.

Details

If no attributes are indicated, those that form the primary key of the lookup table are considered in the flat table.

Value

A `tibble` with attribute values.

See Also

flat_table

Other flat table join functions: get_pk_attribute_names(), join_lookup_table(), lookup_table()

Examples

```
lookup <- flat_table('iris', iris) |>
  lookup_table(
    measures = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"),
    measure_agg = c('MAX', 'MIN', 'SUM', 'MEAN')
  )
values <- flat_table('iris', iris) |>
  check_lookup_table(lookup = lookup)
```

---

constellation *Create constellation*

Description

Creates a constellation from a list of star_database objects. A constellation is also represented by a star_database object. All dimensions with the same name in the star schemas have to be conformable (share the same structure, even though they have different instances).

Usage

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

Arguments

name	A string.
...	star_database objects.

Value

A star_database object.

See Also

as_tibble_list, as_dm_class

Examples

```
db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()
ct1 <- constellation("MRS", db1, db2)


db3 <- star_database(mrs_cause_schema_rpd, ft_cause_rpd) |>
  role_playing_dimension(
    rpd = "When",
    roles = c("When Available", "When Received")
  )

db4 <- star_database(mrs_age_schema_rpd, ft_age_rpd) |>
  role_playing_dimension(
    rpd = "When Arrived",
    roles = c("When Available")
  )
ct2 <- constellation("MRS", db3, db4)
```

---

coordinates_to_point *Transform coordinates to point geometry*

Description

From the coordinates defined in fields such as latitude and longitude, it returns a layer of points.

Usage

```
coordinates_to_point(table, lon_lat = c("intptlon", "intptlat"), crs = NULL)
```

Arguments

table	A tibble object.
lon_lat	A vector, name of longitude and latitude attributes.
crs	A coordinate reference system: integer with the EPSG code, or character with proj4string.

Details

If we start from a geographic layer, it initially transforms it into a table.

The CRS of the new layer is indicated. If a CRS is not indicated, it considers the layer's CRS by default and, if it is not a layer, it considers 4326 CRS (WGS84).

Value

A sf object.

Examples

```
us_state_point <-
  coordinates_to_point(us_layer_state,
                       lon_lat = c("INTPTLON", "INTPTLAT"))
```

`define_dimension` *Define dimension in a* `star_schema` *object.*

Description

Dimensions are part of a `star_schema` object. They can be defined directly as a `dimension_schema` object or giving the name and a set of attributes.

Usage

```
define_dimension(
  schema,
  dimension,
  name,
  attributes,
  scd_nk,
  scd_t0,
  scd_t1,
  scd_t2,
  scd_t3,
  scd_t6,
  is_when,
  ...
)

## S3 method for class 'star_schema'
define_dimension(
  schema,
  dimension = NULL,
  name = NULL,
  attributes = NULL,
  scd_nk = NULL,
  scd_t0 = NULL,
  scd_t1 = NULL,
  scd_t2 = NULL,
  scd_t3 = NULL,
  scd_t6 = NULL,
  is_when = FALSE,
  ...
)
```

Arguments

schema	A star_schema object.
dimension	A dimension_schema object.
name	A string, name of the dimension.
attributes	A vector of attribute names.
scd_nk	A vector of attribute names, scd natural key.
scd_t0	A vector of attribute names, scd T0 attributes.
scd_t1	A vector of attribute names, scd T1 attributes.
scd_t2	A vector of attribute names, scd T2 attributes.
scd_t3	A vector of attribute names, scd T3 attributes.
scd_t6	A vector of attribute names, scd T6 attributes.
is_when	A boolean, is when dimension.
...	When dimension configuration parameters.

Value

A star_schema object.

See Also

star_database

Other star schema definition functions: define_facts(), dimension_schema(), fact_schema(), star_schema()

Examples

```
s <- star_schema() |>
  define_dimension(
    name = "when",
    attributes = c(
      "Week Ending Date",
      "WEEK",
      "Year"
    )
  )

s <- star_schema()
d <- dimension_schema(
  name = "when",
  attributes = c(
    "Week Ending Date",
    "WEEK",
    "Year"
  )
)
s <- s |>
  define_dimension(d)
```

define_facts *Define facts in a* star_schema *object.*

Description

Facts are part of a star_schema object. They can be defined directly as a fact_schema object or giving the name and a set of measures that can be empty (does not have explicit measures).

Usage

```
define_facts(schema, facts, name, measures, agg_functions, nrow_agg)

## S3 method for class 'star_schema'
define_facts(
  schema,
  facts = NULL,
  name = NULL,
  measures = NULL,
  agg_functions = NULL,
  nrow_agg = NULL
)
```

Arguments

schema	A star_schema object.
facts	A fact_schema object.
name	A string, name of the fact.
measures	A vector of measure names.
agg_functions	A vector of aggregation function names, each one for its corresponding measure. If none is indicated, the default is SUM. Additionally they can be MAX or MIN.
nrow_agg	A string, name of a new measure that represents the COUNT of rows aggregated for each resulting row.

Details

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

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

Value

A star_schema object.

See Also

star_database

Other star schema definition functions: define_dimension(), dimension_schema(), fact_schema(), star_schema()

Examples

```
s <- star_schema() |>
  define_facts(
    name = "mrs_cause",
    measures = c(
      "Pneumonia and Influenza Deaths",
      "Other Deaths"
    )
  )

s <- star_schema()
f <- fact_schema(
  name = "mrs_cause",
  measures = c(
    "Pneumonia and Influenza Deaths",
    "Other Deaths"
  )
)
s <- s |>
  define_facts(f)
```

---

define_geoattribute *Define* geoattribute *of a dimension*

Description

Define a set of attributes as a dimension's geoattribute. The set of attribute values must uniquely designate the instances of the given geographic layer.

Usage

```
define_geoattribute(db, dimension, attribute, from_layer, by, from_attribute)

## S3 method for class 'star_database'
define_geoattribute(
  db,
  dimension = NULL,
  attribute = NULL,
  from_layer = NULL,
  by = NULL,
  from_attribute = NULL
)
```

Arguments

db	A star_database object.
dimension	A string, dimension name.
attribute	A vector, attribute names.
from_layer	A sf object.
by	a vector of correspondence of attributes of the dimension with the sf layer structure.
from_attribute	A vector, attribute names.

Details

The definition can be done in two ways: Associates the instances of the attributes with the instances of a geographic layer or defines it from the geometry of previously defined geographic attributes.

Multiple attributes can be specified in the attribute parameter, the geographical attribute is the combination of all of them.

If defined from a layer (from_layer parameter), additionally the attributes used for the join between the tables (dimension and layer tables) must be indicated (by parameter).

If defined from another attribute, it should have the same or finer granularity, to obtain the result by grouping its instances. The considered attribute can be the pair that defines longitude and latitude.

If other geographic information has previously been associated with that attribute, the new information is considered and previous instances for which no new information is provided are also added.

If the geometry provided is polygons, a point layer is also generated.

Value

A star_database object.

See Also

Other star database geographic attributes: check_geoattribute_geometry(), get_geoattribute_geometries(), get_geoattributes(), get_layer_geometry(), get_point_geometry(), summarize_layer()

Examples

```
db <- mrs_db |>
  define_geoattribute(
    dimension = "where",
    attribute = "state",
    from_layer = us_layer_state,
    by = "STUSPS"
  ) |>
  define_geoattribute(
    dimension = "where",
    attribute = "region",
    from_attribute = "state"
  )  |>
```

```
define_geoattribute(
  dimension = "where",
  attribute = "city",
  from_attribute = c("long", "lat")
)
```

---

deploy *Deploy a star database in a relational database*

Description

To deploy the star database, we must indicate a name for the deployment, a connection function and a disconnection function from the database. If it is the first deployment, we must also indicate the name of a local file where the star database will be stored.

Usage

```
deploy(db, name, connect, disconnect, file)

## S3 method for class 'star_database'
deploy(db, name, connect, disconnect = NULL, file = NULL)
```

Arguments

db	A star_database object.
name	A string, name of the deployment.
connect	A function that returns a DBI::DBIConnection object.
disconnect	A function that receives a DBI::DBIConnection object as a parameter and close the connection.
file	A string, name of the file to store the object.

Details

If the disconnection function consists only of calling DBI::dbDisconnect(con), there is no need to indicate it, it is taken by default.

As a result, it exports the tables from the star database to the connection database and from now on will keep them updated with each periodic refresh. Additionally, it will also keep a copy of the star database updated on file, which can be used when needed.

Value

A star_database object.

See Also

star_database

Other star database deployment functions: cancel_deployment(), get_deployment_names(), load_star_database()

Examples

```
mrs_rdb_file <- tempfile("mrs", fileext = ".rdb")
mrs_sqlite_file <- tempfile("mrs", fileext = ".sqlite")

mrs_sqlite_connect <- function() {
  DBI::dbConnect(RSQLite::SQLite(),
                 dbname = mrs_sqlite_file)
}

mrs_db <- mrs_db |>
  deploy(
    name = "mrs",
    connect = mrs_sqlite_connect,
    file = mrs_rdb_file
  )
```

---

dimension_schema dimension_schema *S3 class*

Description

A dimension_schema object is created, we have to define its name and the set of attributes that make it up.

Usage

```
dimension_schema(
  name = NULL,
  attributes = NULL,
  scd_nk = NULL,
  scd_t0 = NULL,
  scd_t1 = NULL,
  scd_t2 = NULL,
  scd_t3 = NULL,
  scd_t6 = NULL,
  is_when = FALSE,
  ...
)
```

Arguments

name	A string, name of the dimension.
attributes	A vector of attribute names.
scd_nk	A vector of attribute names, scd natural key.
scd_t0	A vector of attribute names, scd T0 attributes.
scd_t1	A vector of attribute names, scd T1 attributes.
scd_t2	A vector of attribute names, scd T2 attributes.
scd_t3	A vector of attribute names, scd T3 attributes.
scd_t6	A vector of attribute names, scd T6 attributes.
is_when	A boolean, is when dimension.
...	When dimension configuration parameters.

Details

A dimension_schema object is part of a star_schema object, defines a dimension of the star schema.

Value

A dimension_schema object.

See Also

star_database

Other star schema definition functions: define_dimension(), define_facts(), fact_schema(), star_schema()

Examples

```
d <- dimension_schema(
  name = "when",
  attributes = c(
    "Week Ending Date",
    "WEEK",
    "Year"
  )
)
```

draw_tables *Draw tables*

Description

Draw the tables of the ROLAP star diagrams.

Usage

```
draw_tables(db)

## S3 method for class 'star_database'
draw_tables(db)
```

Arguments

db	A star_database object.

Value

An object with a print() method.

See Also

star_database

Other star database exportation functions: as_csv_files(), as_dm_class(), as_multistar(), as_rdb(), as_single_tibble_list(), as_tibble_list(), as_xlsx_file()

Examples

```
db <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()

db |>
  draw_tables()
```

fact_schema fact_schema *S3 class*

Description

A fact_schema object is created, the essential data is a name and a set of measures that can be empty (does not have explicit measures). It is part of a star_schema object, defines the facts of the star schema.

Usage

```
fact_schema(
  name = NULL,
  measures = NULL,
  agg_functions = NULL,
  nrow_agg = NULL
)
```

Arguments

name	A string, name of the fact.
measures	A vector of measure names.
agg_functions	A vector of aggregation function names, each one for its corresponding measure. If none is indicated, the default is SUM. Additionally they can be MAX or MIN.
nrow_agg	A string, name of a new measure that represents the COUNT of rows aggregated for each resulting row.

Details

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

An additional measure corresponding to the COUNT of aggregated rows is added which, together with SUM, allows us to obtain the AVG if needed.

Value

A fact_schema object.

See Also

star_database

Other star schema definition functions: define_dimension(), define_facts(), dimension_schema(), star_schema()

Examples

```
f <- fact_schema(
  name = "mrs_cause",
  measures = c(
    "Pneumonia and Influenza Deaths",
    "Other Deaths"
  )
)

f <- fact_schema(
  name = "mrs_cause",
  measures = c(
    "Pneumonia and Influenza Deaths",
    "Other Deaths"
  ),
  agg_functions = c(
    "MAX",
    "SUM"
  ),
  nrow_agg = "Nrow"
)
```

---

filter_dimension *Filter dimension*

Description

Allows you to define selection conditions for dimension rows.

Usage

```
filter_dimension(sq, name, ...)

## S3 method for class 'star_query'
filter_dimension(sq, name = NULL, ...)
```

Arguments

sq	A star_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 star_query object.

See Also

Other query functions: as_GeoPackage(), as_geolayer(), get_layer(), get_variable_description(), get_variables(), run_query(), select_dimension(), select_fact(), set_layer(), set_variables(), star_query()

Examples

```
sq <- mrs_db |>
  star_query() |>
  filter_dimension(name = "when", week <= " 3") |>
  filter_dimension(name = "where", city == "Cambridge")
```

---

flat_table flat_table *S3 class*

Description

Creates a flat_table object.

Usage

```
flat_table(name = NULL, instances, unknown_value = NULL)
```

Arguments

name	A string.
instances	A tibble, table of instances.
unknown_value	A string, value used to replace empty and NA values in attributes.

Details

The objective is to allow the transformation of flat tables.

We indicate the name of the flat table and we can also give the value that will be used to replace NA or empty values.

Value

A flat_table object.

See Also

star_database

Other flat table definition functions: as_star_database(), get_table(), get_unknown_value_defined(), get_unknown_values(), read_flat_table_file(), read_flat_table_folder()

Examples

```
ft <- flat_table('iris', iris)

ft <- flat_table('ft_num', ft_num)
```

---

ft *Mortality Reporting System*

Description

Selection of 20 rows from the 122 Cities Mortality Reporting System.

Usage

```
ft
```

Format

A tibble.

Details

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are 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

See Also

mrs_cause_schema

Other mrs example data: ft_age, ft_age_rpd, ft_cause_rpd, ft_num, mrs_db, mrs_db_geo, mrs_ft, mrs_ft_new

ft_age *Mortality Reporting System by Age Group*

Description

Selection data from the 122 Cities Mortality Reporting System by age group.

Usage

```
ft_age
```

Format

A tibble.

Details

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are included.

Source

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

See Also

mrs_age_schema

Other mrs example data: ft, ft_age_rpd, ft_cause_rpd, ft_num, mrs_db, mrs_db_geo, mrs_ft, mrs_ft_new

Examples

```
# The operations to obtain it from the `ft` data set are:

if (rlang::is_installed("stringr")) {
  ft_age <- ft |>
    dplyr::select(-`Pneumonia and Influenza Deaths`, -`All Deaths`) |>
    tidyr::gather("Age", "All Deaths", 7:11) |>
    dplyr::mutate(`All Deaths` = as.integer(`All Deaths`)) |>
    dplyr::mutate(Age = stringr::str_replace(Age, " \\(all cause deaths\\)", ""))
}
```

ft_age_rpd *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 and 4 cities.

Usage

```
ft_age_rpd
```

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

See Also

`mrs_age_schema`

Other mrs example data: `ft`, `ft_age`, `ft_cause_rpd`, `ft_num`, `mrs_db`, `mrs_db_geo`, `mrs_ft`, `mrs_ft_new`

ft_cause_rpd *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 and 4 cities.

Usage

```
ft_cause_rpd
```

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

See Also

mrs_cause_schema

Other mrs example data: ft, ft_age, ft_age_rpd, ft_num, mrs_db, mrs_db_geo, mrs_ft, mrs_ft_new

---

ft_num *Mortality Reporting System with numerical measures*

Description

Selection of 20 rows from the 122 Cities Mortality Reporting System. Measures have been defined as integer values.

Usage

```
ft_num
```

Format

A tibble.

Details

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, mortality figures by age group and cause, considered separately, are 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

See Also

mrs_cause_schema

Other mrs example data: ft, ft_age, ft_age_rpd, ft_cause_rpd, mrs_db, mrs_db_geo, mrs_ft, mrs_ft_new

Examples

```
# The operations to obtain it from the `ft` data set are:

ft_num <- ft |>
 dplyr::mutate(`Pneumonia and Influenza Deaths` = as.integer(`Pneumonia and Influenza Deaths`)) |>
  dplyr::mutate(`All Deaths` = as.integer(`All Deaths`))
```

---

get_attribute_names.flat_table

Get the names of the attributes

Description

Obtain the names of the attributes in a flat table or a dimension in a star database.

Usage

```
## S3 method for class 'flat_table'
get_attribute_names(db, name = NULL, ordered = FALSE, as_definition = FALSE)

get_attribute_names(db, name, ordered, as_definition)

## S3 method for class 'star_database'
get_attribute_names(db, name, ordered = FALSE, as_definition = FALSE)
```

Arguments

db	A flat_table or star_database object.
name	A string, dimension name.
ordered	A boolean, sort names alphabetically.
as_definition	A boolean, get the names as a vector definition in R.

Details

If indicated, names can be obtained in alphabetical order or as a vector definition in R

Value

A vector of strings or a string, attribute names.

See Also

star_database, flat_table

Other star database and flat table functions: get_measure_names.flat_table(), get_similar_attribute_values.flat_
get_similar_attribute_values_individually.flat_table(), get_unique_attribute_values.flat_table(),
replace_attribute_values.flat_table(), set_attribute_names.flat_table(), set_measure_names.flat_table
snake_case.flat_table()

Examples

```
names <- star_database(mrs_cause_schema, ft_num) |>
  get_attribute_names(name = "where")

names <- flat_table('iris', iris) |>
  get_attribute_names()
```

---

get_deployment_names *Get the names of the facts of a star database*

Description

Obtain the names of the facts of a star database.

Usage

```
get_deployment_names(db)

## S3 method for class 'star_database'
get_deployment_names(db)
```

Arguments

db	A star_database object.

Value

A vector of strings, fact names.

See Also

star_database

Other star database deployment functions: cancel_deployment(), deploy(), load_star_database()

Examples

```
mrs_rdb_file <- tempfile("mrs", fileext = ".rdb")
mrs_sqlite_file <- tempfile("mrs", fileext = ".sqlite")

mrs_sqlite_connect <- function() {
  DBI::dbConnect(RSQLite::SQLite(),
                 dbname = mrs_sqlite_file)
}

mrs_db <- mrs_db |>
  deploy(
    name = "mrs",
    connect = mrs_sqlite_connect,
    file = mrs_rdb_file
  )

names <- mrs_db |>
  get_deployment_names()
```

---

get_dimension_names *Get the names of the dimensions of a star database*

Description

Obtain the names of the dimensions of a star database.

Usage

```
get_dimension_names(db, star)

## S3 method for class 'star_database'
get_dimension_names(db, star = NULL)
```

Arguments

db	A star_database object.
star	A string or integer, star database name or index in constellation.

Value

A vector of strings, dimension names.

See Also

star_schema, flat_table

Other star database definition functions: get_dimension_table(), get_fact_names(), get_role_playing_dimension_n
get_table_names(), group_dimension_instances(), role_playing_dimension(), star_database()

Examples

```
names <- star_database(mrs_cause_schema, ft_num) |>
  get_dimension_names()
```

`get_dimension_table` *Get dimension table*

Description

Get the table for the dimension indicated by its name.

Usage

```
get_dimension_table(db, name)

## S3 method for class 'star_database'
get_dimension_table(db, name = NULL)
```

Arguments

db	A `star_database` object.
name	A string, dimension name.

Value

A `tibble`, dimension table.

See Also

`star_schema`, `flat_table`

Other star database definition functions: `get_dimension_names()`, `get_fact_names()`, `get_role_playing_dimension_n`
`get_table_names()`, `group_dimension_instances()`, `role_playing_dimension()`, `star_database()`

Examples

```
table <- star_database(mrs_cause_schema, ft_num) |>
  get_dimension_table("where")
```

`get_existing_fact_instances` *Get existing fact instances*

Description

From the planned update, it obtains the instances of the update facts that are already included in the star database facts to be updated.

Usage

```
get_existing_fact_instances(sdbu)

## S3 method for class 'star_database_update'
get_existing_fact_instances(sdbu)
```

Arguments

sdbu	A `star_database_update` object.

Details

The most common thing is that refresh operations only include new instances in fact tables, but it may be the case that repeated instances appear: They may have different values in the measures, but the same values in the dimension foreign keys. When the update occurs, we need to determine what happens to these instances.

Value

A `tibble` object.

See Also

`star_database`

Other star database refresh functions: `get_lookup_tables()`, `get_new_dimension_instances()`, `get_star_database()`, `get_star_schema()`, `get_transformation_code()`, `get_transformation_file()`, `incremental_refresh()`, `update_according_to()`

Examples

```
f1 <-
  flat_table('ft_num', ft_cause_rpd[ft_cause_rpd$City != 'Cambridge' &
                                      ft_cause_rpd$WEEK != '4',]) |>
  as_star_database(mrs_cause_schema_rpd) |>
  role_playing_dimension(rpd = "When",
                         roles = c("When Available", "When Received"))
f2 <- flat_table('ft_num2', ft_cause_rpd[ft_cause_rpd$City != 'Bridgeport' &
                                           ft_cause_rpd$WEEK != '2',])
```

```
f2 <- f2 |>
  update_according_to(f1)
fact_instances <- f2 |>
  get_existing_fact_instances()
```

`get_fact_names` *Get the names of the facts of a star database*

Description

Obtain the names of the facts of a star database.

Usage

```
get_fact_names(db)

## S3 method for class 'star_database'
get_fact_names(db)
```

Arguments

db	A `star_database` object.

Value

A vector of strings, fact names.

See Also

`star_schema`, `flat_table`

Other star database definition functions: `get_dimension_names()`, `get_dimension_table()`, `get_role_playing_dimens`
`get_table_names()`, `group_dimension_instances()`, `role_playing_dimension()`, `star_database()`

Examples

```
names <- star_database(mrs_cause_schema, ft_num) |>
  get_fact_names()
```

`get_geoattributes` *Get geoattributes*

Description

For each dimension, get a list of available geoattributes.

Usage

```
get_geoattributes(db)

## S3 method for class 'star_database'
get_geoattributes(db)
```

Arguments

db	A `star_database` object.

Value

A list of dimension geoattributes.

See Also

Other star database geographic attributes: `check_geoattribute_geometry()`, `define_geoattribute()`, `get_geoattribute_geometries()`, `get_layer_geometry()`, `get_point_geometry()`, `summarize_layer()`

Examples

```
db <- mrs_db |>
  define_geoattribute(
    dimension = "where",
    attribute = "state",
    from_layer = us_layer_state,
    by = "STUSPS"
  )

attributes <- db |>
    get_geoattributes()
```

get_geoattribute_geometries *Get geoattribute geometries*

Description

For each geoattribute, get its geometries.

Usage

```
get_geoattribute_geometries(db, dimension, attribute)

## S3 method for class 'star_database'
get_geoattribute_geometries(db, dimension = NULL, attribute = NULL)
```

Arguments

db	A star_database object.
dimension	A string, dimension name.
attribute	A vector, attribute names.

Details

If the name of the dimension is not indicated, it is considered the first one that has geoattributes defined.

Value

A vector of strings.

See Also

Other star database geographic attributes: check_geoattribute_geometry(), define_geoattribute(), get_geoattributes(), get_layer_geometry(), get_point_geometry(), summarize_layer()

Examples

```
db <- mrs_db |>
  define_geoattribute(
    dimension = "where",
    attribute = "state",
    from_layer = us_layer_state,
    by = "STUSPS"
  )

geometries <- db |>
  get_geoattribute_geometries(
    dimension = "where",
```

```
  attribute = "state"
)
```

`get_layer` *Get geographic information layer*

Description

Get the geographic information layer from a `geolayer` object.

Usage

```
get_layer(gl, keep_all_variables_na)

## S3 method for class 'geolayer'
get_layer(gl, keep_all_variables_na = FALSE)
```

Arguments

`gl`	A `geolayer` object.
`keep_all_variables_na`	A boolean, keep rows with all variables NA.

Details

By default, rows that are NA for all variables are eliminated.

Value

A `sf` object.

See Also

Other query functions: `as_GeoPackage()`, `as_geolayer()`, `filter_dimension()`, `get_variable_description()`, `get_variables()`, `run_query()`, `select_dimension()`, `select_fact()`, `set_layer()`, `set_variables()`, `star_query()`

Examples

```
gl <- mrs_db_geo |>
  as_geolayer()

l <- gl |>
  get_layer()
```

get_layer_geometry *Get layer geometry*

Description

Get the geometry of a layer. It will only be valid if one of the two geometries is interpreted: *point* or *polygon*.

Usage

```
get_layer_geometry(layer)
```

Arguments

layer	A sf object.

Value

A string.

See Also

Other star database geographic attributes: check_geoattribute_geometry(), define_geoattribute(), get_geoattribute_geometries(), get_geoattributes(), get_point_geometry(), summarize_layer()

Examples

```
geometry <- get_layer_geometry(us_layer_state)
```

get_lookup_tables *Get lookup tables*

Description

From the planned update, it obtains the lookup tables used to define the data.

Usage

```
get_lookup_tables(sdbu)

## S3 method for class 'star_database_update'
get_lookup_tables(sdbu)
```

Arguments

sdbu	A star_database_update object.

Value

A list of `flat_table` objects.

See Also

`star_database`

Other star database refresh functions: `get_existing_fact_instances()`, `get_new_dimension_instances()`, `get_star_database()`, `get_star_schema()`, `get_transformation_code()`, `get_transformation_file()`, `incremental_refresh()`, `update_according_to()`

Examples

```
f1 <- flat_table('ft_num', ft_cause_rpd) |>
  as_star_database(mrs_cause_schema_rpd)
f2 <- flat_table('ft_num2', ft_cause_rpd) |>
  update_according_to(f1)
ft <- f2 |>
  get_lookup_tables()
```

---

get_measure_names.flat_table

Get the names of the measures

Description

Obtain the names of the measures in a flat table or in a star database.

Usage

```
## S3 method for class 'flat_table'
get_measure_names(db, name = NULL, ordered = FALSE, as_definition = FALSE)

get_measure_names(db, name, ordered, as_definition)

## S3 method for class 'star_database'
get_measure_names(db, name = NULL, ordered = FALSE, as_definition = FALSE)
```

Arguments

`db`	A `flat_table` or `star_database` object.
`name`	A string, dimension name.
`ordered`	A boolean, sort names alphabetically.
`as_definition`	A boolean, get the names as a vector definition in R.

Value

A vector of strings or a string, measure names.

See Also

star_database, flat_table

Other star database and flat table functions: get_attribute_names.flat_table(), get_similar_attribute_values.fla
get_similar_attribute_values_individually.flat_table(), get_unique_attribute_values.flat_table(),
replace_attribute_values.flat_table(), set_attribute_names.flat_table(), set_measure_names.flat_table
snake_case.flat_table()

Examples

```
names <- star_database(mrs_cause_schema, ft_num) |>
  get_measure_names()

names <- flat_table('iris', iris) |>
  get_measure_names()
```

---

get_new_dimension_instances

Get new dimension instances

Description

From the planned update, it obtains the instances of the update dimensions that are not included in the star database dimensions to be updated.

Usage

```
get_new_dimension_instances(sdbu)

## S3 method for class 'star_database_update'
get_new_dimension_instances(sdbu)
```

Arguments

sdbu	A star_database_update object.

Value

A list of tibble objects.

See Also

star_database

Other star database refresh functions: get_existing_fact_instances(), get_lookup_tables(), get_star_database(), get_star_schema(), get_transformation_code(), get_transformation_file(), incremental_refresh(), update_according_to()

Examples

```
f1 <-
  flat_table('ft_num', ft_cause_rpd[ft_cause_rpd$City != 'Cambridge' &
                                      ft_cause_rpd$WEEK != '4',]) |>
  as_star_database(mrs_cause_schema_rpd) |>
  role_playing_dimension(rpd = "When",
                         roles = c("When Available", "When Received"))
f2 <- flat_table('ft_num2', ft_cause_rpd[ft_cause_rpd$City != 'Bridgeport' &
                                           ft_cause_rpd$WEEK != '2',])
f2 <- f2 |>
  update_according_to(f1)
dim_instances <- f2 |>
  get_new_dimension_instances()
```

---

get_pk_attribute_names

Get the names of the primary key attributes of a flat table

Description

Obtain the names of the attributes that form the primary key of a flat table, if defined.

Usage

```
get_pk_attribute_names(ft, as_definition)

## S3 method for class 'flat_table'
get_pk_attribute_names(ft, as_definition = FALSE)
```

Arguments

ft	A flat_table object.
as_definition	A boolean, as the definition of the vector in R.

Value

A vector of strings or a tibble, attribute names.

See Also

flat_table

Other flat table join functions: check_lookup_table(), join_lookup_table(), lookup_table()

Examples

```
ft <- flat_table('iris', iris) |>
  lookup_table(
    measures = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"),
    measure_agg = c('MAX', 'MIN', 'SUM', 'MEAN')
  )
names <- ft |>
  get_pk_attribute_names()
```

---

get_point_geometry *Get point geometry*

Description

Obtain point geometry from polygon geometry.

Usage

```
get_point_geometry(layer)
```

Arguments

layer	A sf object.

Value

A sf object.

See Also

Other star database geographic attributes: check_geoattribute_geometry(), define_geoattribute(), get_geoattribute_geometries(), get_geoattributes(), get_layer_geometry(), summarize_layer()

Examples

```
layer <-
  get_point_geometry(us_layer_state)
```

`get_role_playing_dimension_names` *Get the names of the role playing dimensions*

Description

Role playing dimensions are defined in star_databases. When integrating several star_databases to form a constellation, role playing dimensions are also integrated. This function allows you to see the result.

Usage

```
get_role_playing_dimension_names(db)

## S3 method for class 'star_database'
get_role_playing_dimension_names(db)
```

Arguments

db	A `constellation` object.

Value

A list of vector of strings with dimension names.

See Also

`star_schema`, `flat_table`

Other star database definition functions: `get_dimension_names()`, `get_dimension_table()`, `get_fact_names()`, `get_table_names()`, `group_dimension_instances()`, `role_playing_dimension()`, `star_database()`

Examples

```
db1 <- star_database(mrs_cause_schema_rpd, ft_cause_rpd) |>
  role_playing_dimension(
    rpd = "When",
    roles = c("When Available", "When Received")
  )

db2 <- star_database(mrs_age_schema_rpd, ft_age_rpd) |>
  role_playing_dimension(
    rpd = "When Arrived",
    roles = c("When Available")
  )
rpd <- constellation("MRS", db1, db2) |>
  get_role_playing_dimension_names()
```

`get_similar_attribute_values.flat_table` *Get similar attribute values combination*

Description

Get sets of attribute values that differ only by tildes, spaces, or punctuation marks, for the combination of the given set of attributes. If no attributes are indicated, they are all considered together.

Usage

```
## S3 method for class 'flat_table'
get_similar_attribute_values(
  db,
  name = NULL,
  attributes = NULL,
  exclude_numbers = FALSE,
  col_as_vector = NULL
)

get_similar_attribute_values(
  db,
  name,
  attributes,
  exclude_numbers,
  col_as_vector
)

## S3 method for class 'star_database'
get_similar_attribute_values(
  db,
  name = NULL,
  attributes = NULL,
  exclude_numbers = FALSE,
  col_as_vector = NULL
)
```

Arguments

`db`	A `flat_table` or `star_database` object.
`name`	A string, dimension name.
`attributes`	A vector of strings, attribute names.
`exclude_numbers`	A boolean, exclude numbers from comparison.
`col_as_vector`	A string, name of the column to include a vector of values.

Details

For star databases, a list of dimensions can be indicated, otherwise it considers all dimensions. If a dimension is indicated, a list of attributes to be considered in it can also be indicated.

You can indicate that the numbers are ignored to make the comparison.

If a name is indicated in the `col_as_vector` parameter, it includes a column with the data in vector form to be used in other functions.

Value

A vector of `tibble` objects with similar instances.

See Also

`star_database`, `flat_table`

Other star database and flat table functions: `get_attribute_names.flat_table()`, `get_measure_names.flat_table()`, `get_similar_attribute_values_individually.flat_table()`, `get_unique_attribute_values.flat_table()`, `replace_attribute_values.flat_table()`, `set_attribute_names.flat_table()`, `set_measure_names.flat_table()`, `snake_case.flat_table()`

Examples

```
instances <- star_database(mrs_cause_schema, ft_num) |>
  get_similar_attribute_values(name = "where")

db <- star_database(mrs_cause_schema, ft_num)
db$dimensions$where$table$City[2] <- " BrId  gEport "
instances <- db |>
  get_similar_attribute_values("where")

db <- star_database(mrs_cause_schema, ft_num)
db$dimensions$where$table$City[2] <- " BrId  gEport "
instances <- db |>
  get_similar_attribute_values("where",
    attributes = c("City", "State"),
    col_as_vector = "As a vector")

ft <- flat_table('iris', iris)
ft$table$Species[20] <- "se.Tosa."
ft$table$Species[60] <- "Versicolor"
instances <- ft |>
  get_similar_attribute_values()
```

`get_similar_attribute_values_individually.flat_table` *Get similar values for individual attributes*

Description

Get sets of attribute values for individual attributes that differ only by tildes, spaces, or punctuation marks. If no attributes are indicated, all are considered.

Usage

```
## S3 method for class 'flat_table'
get_similar_attribute_values_individually(
  db,
  name = NULL,
  attributes = NULL,
  exclude_numbers = FALSE,
  col_as_vector = NULL
)

get_similar_attribute_values_individually(
  db,
  name,
  attributes,
  exclude_numbers,
  col_as_vector
)

## S3 method for class 'star_database'
get_similar_attribute_values_individually(
  db,
  name = NULL,
  attributes = NULL,
  exclude_numbers = FALSE,
  col_as_vector = NULL
)
```

Arguments

`db`	A `flat_table` or `star_database` object.
`name`	A vector of strings, dimension names.
`attributes`	A vector of strings, attribute names.
`exclude_numbers`	A boolean, exclude numbers from comparison.
`col_as_vector`	A string, name of the column to include a vector of values.

Details

For star databases, if no dimension name is indicated, all dimensions are considered.

You can indicate that the numbers are ignored to make the comparison.

If a name is indicated in the `col_as_vector` parameter, it includes a column with the data in vector form to be used in other functions.

Value

A vector of `tibble` objects with similar instances.

See Also

`star_database`, `flat_table`

Other star database and flat table functions: `get_attribute_names.flat_table()`, `get_measure_names.flat_table()`, `get_similar_attribute_values.flat_table()`, `get_unique_attribute_values.flat_table()`, `replace_attribute_values.flat_table()`, `set_attribute_names.flat_table()`, `set_measure_names.flat_table()`, `snake_case.flat_table()`

Examples

```
instances <- star_database(mrs_cause_schema, ft_num) |>
  get_similar_attribute_values_individually(name = c("where", "when"))

instances <- star_database(mrs_cause_schema, ft_num) |>
  get_similar_attribute_values_individually()

ft <- flat_table('iris', iris)
ft$table$Species[20] <- "se.Tosa."
ft$table$Species[60] <- "Versicolor"
instances <- ft |>
  get_similar_attribute_values_individually()
```

---

get_star_database *Get star database*

Description

It obtains the star database: For updates, the one defined from the data; for constellations, the one indicated by the parameter.

Usage

```
get_star_database(db, name)

## S3 method for class 'star_database_update'
get_star_database(db, name = NULL)
```

```
## S3 method for class 'star_database'
get_star_database(db, name)
```

Arguments

db	A star_database_update object.
name	A string, star database name (fact name).

Value

A star_database object.

See Also

star_database

Other star database refresh functions: get_existing_fact_instances(), get_lookup_tables(), get_new_dimension_instances(), get_star_schema(), get_transformation_code(), get_transformation_file(), incremental_refresh(), update_according_to()

Examples

```
f1 <- flat_table('ft_num', ft_cause_rpd) |>
  as_star_database(mrs_cause_schema_rpd)
f2 <- flat_table('ft_num2', ft_cause_rpd) |>
  update_according_to(f1)
st <- f2 |>
  get_star_database()

db1 <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()
db2 <- star_database(mrs_age_schema, ft_age) |>
  snake_case()
ct <- constellation("MRS", db1, db2)
names <- ct |>
  get_fact_names()
st <- ct |>
  get_star_database(names[1])
```

---

get_star_schema *Get star schema*

Description

From the planned update, it obtains the star schema used to define the data.

Usage

```
get_star_schema(sdbu)

## S3 method for class 'star_database_update'
get_star_schema(sdbu)
```

Arguments

sdbu	A star_database_update object.

Value

A star_schema object.

See Also

star_database

Other star database refresh functions: get_existing_fact_instances(), get_lookup_tables(), get_new_dimension_instances(), get_star_database(), get_transformation_code(), get_transformation_file(incremental_refresh(), update_according_to()

Examples

```
f1 <- flat_table('ft_num', ft_cause_rpd) |>
  as_star_database(mrs_cause_schema_rpd)
f2 <- flat_table('ft_num2', ft_cause_rpd) |>
  update_according_to(f1)
st <- f2 |>
  get_star_schema()
```

---

get_table *Get the table of the flat table*

Description

Obtain the table of a flat table.

Usage

```
get_table(ft)

## S3 method for class 'flat_table'
get_table(ft)
```

Arguments

ft	A flat_table object.

Value

A tibble, the table.

See Also

star_database

Other flat table definition functions: as_star_database(), flat_table(), get_unknown_value_defined(), get_unknown_values(), read_flat_table_file(), read_flat_table_folder()

Examples

```
table <- flat_table('iris', iris) |>
  get_table()
```

---

get_table_names *Get the names of the tables of a star database*

Description

Obtain the names of the tables of a star database.

Usage

```
get_table_names(db)

## S3 method for class 'star_database'
get_table_names(db)
```

Arguments

db	A star_database object.

Value

A vector of strings, table names.

See Also

star_schema, flat_table

Other star database definition functions: get_dimension_names(), get_dimension_table(), get_fact_names(), get_role_playing_dimension_names(), group_dimension_instances(), role_playing_dimension(), star_database()

Examples

```
names <- star_database(mrs_cause_schema, ft_num) |>
  get_table_names()
```

get_transformation_code *Get transformation function code*

Description

From the planned update, it obtains the function with the source code of the transformations performed on the original data in string vector format.

Usage

```
get_transformation_code(sdbu)

## S3 method for class 'star_database_update'
get_transformation_code(sdbu)
```

Arguments

sdbu	A star_database_update object.

Value

A vector of strings.

See Also

star_database

Other star database refresh functions: get_existing_fact_instances(), get_lookup_tables(), get_new_dimension_instances(), get_star_database(), get_star_schema(), get_transformation_file(), incremental_refresh(), update_according_to()

Examples

```
f1 <- flat_table('ft_num', ft_cause_rpd) |>
  as_star_database(mrs_cause_schema_rpd) |>
  replace_attribute_values(
    name = "When Available",
    old = c('1962', '11', '1962-03-14'),
    new = c('1962', '3', '1962-01-15')
  ) |>
  group_dimension_instances(name = "When")
f2 <- flat_table('ft_num2', ft_cause_rpd) |>
  update_according_to(f1)
code <- f2 |>
  get_transformation_code()
```

`get_transformation_file` *Get transformation function file*

Description

From the planned update, it obtains the function with the source code of the transformations performed on the original data in file format.

Usage

```
get_transformation_file(sdbu, file)

## S3 method for class 'star_database_update'
get_transformation_file(sdbu, file = NULL)
```

Arguments

`sdbu`	A `star_database_update` object.
`file`	A string, file name.

Value

A string, file name.

See Also

`star_database`

Other star database refresh functions: `get_existing_fact_instances()`, `get_lookup_tables()`, `get_new_dimension_instances()`, `get_star_database()`, `get_star_schema()`, `get_transformation_code()`, `incremental_refresh()`, `update_according_to()`

Examples

```
f1 <- flat_table('ft_num', ft_cause_rpd) |>
  as_star_database(mrs_cause_schema_rpd) |>
  replace_attribute_values(
    name = "When Available",
    old = c('1962', '11', '1962-03-14'),
    new = c('1962', '3', '1962-01-15')
  ) |>
  group_dimension_instances(name = "When")
f2 <- flat_table('ft_num2', ft_cause_rpd) |>
  update_according_to(f1)
file <- f2 |>
  get_transformation_file()
```

`get_unique_attribute_values.flat_table` *Get unique attribute values*

Description

Get unique set of values for the given attributes. If no attributes are indicated, all are considered.

Usage

```
## S3 method for class 'flat_table'
get_unique_attribute_values(
  db,
  name = NULL,
  attributes = NULL,
  col_as_vector = NULL
)

get_unique_attribute_values(db, name, attributes, col_as_vector)

## S3 method for class 'star_database'
get_unique_attribute_values(
  db,
  name = NULL,
  attributes = NULL,
  col_as_vector = NULL
)
```

Arguments

`db`	A `flat_table` or `star_database` object.
`name`	A string, dimension name.
`attributes`	A vector of strings, attribute names.
`col_as_vector`	A string, name of the column to include a vector of values.

Details

If we work on a star database, a dimension must be indicated.

Value

A vector of `tibble` objects with unique instances.

See Also

star_database, flat_table

Other star database and flat table functions: get_attribute_names.flat_table(), get_measure_names.flat_table(), get_similar_attribute_values.flat_table(), get_similar_attribute_values_individually.flat_table(), replace_attribute_values.flat_table(), set_attribute_names.flat_table(), set_measure_names.flat_table(), snake_case.flat_table()

Examples

```
instances <- star_database(mrs_cause_schema, ft_num) |>
  get_unique_attribute_values()

instances <- star_database(mrs_cause_schema, ft_num) |>
  get_unique_attribute_values(name = "where")

instances <- star_database(mrs_cause_schema, ft_num) |>
  get_unique_attribute_values("where",
    attributes = c("REGION", "State"))

instances <- flat_table('iris', iris) |>
  get_unique_attribute_values()
```

---

get_unknown_values *Get unknown attribute values*

Description

Obtain the instances that have an empty or unknown value in any given attribute. If no attribute is given, all are considered.

Usage

```
get_unknown_values(ft, attributes, col_as_vector)

## S3 method for class 'flat_table'
get_unknown_values(ft, attributes = NULL, col_as_vector = NULL)
```

Arguments

ft	A flat_table object.
attributes	A vector of strings, attribute names.
col_as_vector	A string, name of the column to include a vector of values.

Details

If a name is indicated in the col_as_vector parameter, it includes a column with the data in vector form to be used in other functions.

Value

A `tibble` with unknown values in instances.

See Also

`star_database`

Other flat table definition functions: `as_star_database()`, `flat_table()`, `get_table()`, `get_unknown_value_defined()`
`read_flat_table_file()`, `read_flat_table_folder()`

Examples

```
iris2 <- iris
iris2[10, 'Species'] <- NA
instances <- flat_table('iris', iris2) |>
  get_unknown_values()
```

---

get_unknown_value_defined

Get the unknown value defined

Description

Obtain the unknown value of a flat table.

Usage

```
get_unknown_value_defined(ft)

## S3 method for class 'flat_table'
get_unknown_value_defined(ft)
```

Arguments

`ft`	A `flat_table` object.

Value

A string.

See Also

`star_database`

Other flat table definition functions: `as_star_database()`, `flat_table()`, `get_table()`, `get_unknown_values()`,
`read_flat_table_file()`, `read_flat_table_folder()`

Examples

```
table <- flat_table('iris', iris) |>
  get_unknown_value_defined()
```

`get_variables` *Get the variables layer*

Description

The variables layer includes the names and description through various fields of the variables contained in the geolayer.

Usage

```
get_variables(gl)

## S3 method for class 'geolayer'
get_variables(gl)
```

Arguments

`gl`	A `geolayer` object.

Details

The way to select the variables we want to work with is to filter this layer and subsequently set it as the object's variables layer using the `set_variables()` function.

Value

A `tibble` object.

See Also

Other query functions: `as_GeoPackage()`, `as_geolayer()`, `filter_dimension()`, `get_layer()`, `get_variable_description()`, `run_query()`, `select_dimension()`, `select_fact()`, `set_layer()`, `set_variables()`, `star_query()`

Examples

```
gl <- mrs_db_geo |>
  as_geolayer()

v <- gl |>
  get_variables()
```

get_variable_description *Get variable description*

Description

Obtain a description of the variables whose name is indicated. If no name is indicated, all are returned.

Usage

```
get_variable_description(gl, name, only_values)

## S3 method for class 'geolayer'
get_variable_description(gl, name = NULL, only_values = FALSE)
```

Arguments

gl	A geolayer object.
name	A string vector.
only_values	A boolean, add names to component values.

Details

Using the parameter only_values, we can obtain only the combination of values or also the combination of names with values.

Value

A string vector.

See Also

Other query functions: as_GeoPackage(), as_geolayer(), filter_dimension(), get_layer(), get_variables(), run_query(), select_dimension(), select_fact(), set_layer(), set_variables(), star_query()

Examples

```
gl <- mrs_db_geo |>
  as_geolayer()

vd <- gl |>
  get_variable_description()
```

group_dimension_instances *Group instances of a dimension*

Description

After changes in values in the instances of a dimension, groups the instances and, if necessary, also the related facts.

Usage

```
group_dimension_instances(db, name)

## S3 method for class 'star_database'
group_dimension_instances(db, name)
```

Arguments

db	A star_database object.
name	A string, dimension name.

Value

A star_database object.

See Also

star_schema, flat_table

Other star database definition functions: get_dimension_names(), get_dimension_table(), get_fact_names(), get_role_playing_dimension_names(), get_table_names(), role_playing_dimension(), star_database()

Examples

```
db <- star_database(mrs_cause_schema, ft_num) |>
  group_dimension_instances(name = "where")
```

`incremental_refresh` *Refresh a star database in a constellation*

Description

Incremental update of a star database from the star database generated with the new data.

Usage

```
incremental_refresh(db, sdbu, existing_instances, replace_transformations, ...)

## S3 method for class 'star_database'
incremental_refresh(
  db,
  sdbu,
  existing_instances = "ignore",
  replace_transformations = FALSE,
  ...
)
```

Arguments

`db`	A `star_database` object.
`sdbu`	A `star_database_update` object.
`existing_instances`	A string, operation to be carried out on the instances of already existing facts. The possible values are: "ignore", "replace", "group" and "delete".
`replace_transformations`	A boolean, replace the `star_database` transformation code with the `star_database_update` one.
`...`	internal test parameters.

Details

There may be data in the update that already exists in the facts: it is indicated what to do with it, replace it, group it, delete it or ignore it in the update.

If to obtain the update data we have had to perform new transformations (which were not necessary to obtain the star database), we can indicate that these are the new transformation operations for the star database. These operations are not applied to the star database, they will only be applied to new periodic updates.

Value

A `star_database` object.

See Also

star_database

Other star database refresh functions: get_existing_fact_instances(), get_lookup_tables(), get_new_dimension_instances(), get_star_database(), get_star_schema(), get_transformation_code(), get_transformation_file(), update_according_to()

Examples

```
db <-
  flat_table('ft_num', ft_cause_rpd[ft_cause_rpd$City != 'Cambridge' &
                                      ft_cause_rpd$WEEK != '4',]) |>
  as_star_database(mrs_cause_schema_rpd) |>
  role_playing_dimension(rpd = "When",
                         roles = c("When Available", "When Received"))
f2 <- flat_table('ft_num2', ft_cause_rpd[ft_cause_rpd$City != 'Bridgeport' &
                                           ft_cause_rpd$WEEK != '2',])
f2 <- f2 |>
  update_according_to(db)

db <- db |>
  incremental_refresh(f2)
```

---

join_lookup_table *Join a flat table with a lookup table*

Description

To join a flat table with a lookup table, the attributes of the first table that will be used in the operation are indicated. The lookup table must have the primary key previously defined.

Usage

```
join_lookup_table(ft, fk_attributes, lookup)

## S3 method for class 'flat_table'
join_lookup_table(ft, fk_attributes = NULL, lookup)
```

Arguments

ft	A flat_table object.
fk_attributes	A vector of strings, attribute names.
lookup	A flat_table object.

Details

If no attributes are indicated, those that form the primary key of the lookup table are considered in the flat table.

Value

A `flat_table` object.

See Also

`flat_table`

Other flat table join functions: `check_lookup_table()`, `get_pk_attribute_names()`, `lookup_table()`

Examples

```
lookup <- flat_table('iris', iris) |>
  lookup_table(
    measures = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"),
    measure_agg = c('MAX', 'MIN', 'SUM', 'MEAN')
  )
ft <- flat_table('iris', iris) |>
  join_lookup_table(lookup = lookup)
```

---

load_star_database *Load star_database (from a RDS file)*

Description

Load star_database (from a RDS file)

Usage

```
load_star_database(file)
```

Arguments

`file`	A string, name of the file that stores the object.

Value

A `star_database` object.

See Also

`star_database`

Other star database deployment functions: `cancel_deployment()`, `deploy()`, `get_deployment_names()`

Examples

```
mrs_rdb_file <- tempfile("mrs", fileext = ".rdb")
mrs_sqlite_file <- tempfile("mrs", fileext = ".sqlite")

mrs_sqlite_connect <- function() {
  DBI::dbConnect(RSQLite::SQLite(),
                 dbname = mrs_sqlite_file)
}

mrs_db <- mrs_db |>
  deploy(
    name = "mrs",
    connect = mrs_sqlite_connect,
    file = mrs_rdb_file
  )

mrs_db2 <- load_star_database(mrs_rdb_file)
```

---

lookup_table *Transform a flat table into a look up table*

Description

Checks that the given attributes form a primary key of the table. Otherwise, group the records so that they form a primary key. To carry out the groupings, aggregation functions for attributes and measures must be provided.

Usage

```
lookup_table(
  ft,
  pk_attributes,
  attributes,
  attribute_agg,
  measures,
  measure_agg
)

## S3 method for class 'flat_table'
lookup_table(
  ft,
  pk_attributes = NULL,
  attributes = NULL,
  attribute_agg = NULL,
  measures = NULL,
  measure_agg = NULL
)
```

Arguments

ft	A flat_table object.
pk_attributes	A vector of strings, attribute names.
attributes	A vector of strings, rest of attribute names.
attribute_agg	A vector of strings, attribute aggregation functions.
measures	A vector of strings, measure names.
measure_agg	A vector of strings, measure aggregation functions.

Details

If the table does not have measures, attributes with equal values are grouped without the need to indicate a grouping function.

If no attribute is indicated, all the attributes are considered to form the primary key.

Value

A flat_table object.

See Also

flat_table

Other flat table join functions: check_lookup_table(), get_pk_attribute_names(), join_lookup_table()

Examples

```
ft <- flat_table('iris', iris) |>
  lookup_table(
    measures = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"),
    measure_agg = c('MAX', 'MIN', 'SUM', 'MEAN')
  )
```

---

mrs_age_schema *Star schema for Mortality Reporting System by Age*

Description

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

Usage

```
mrs_age_schema
```

Format

A star_schema object.

Details

Dimension schemes can be defined using variables so that you do not have to repeat the definition in several multidimensional designs.

See Also

ft_age

Other mrs example schema: mrs_age_schema_rpd, mrs_cause_schema, mrs_cause_schema_rpd

Examples

```
# Defined by:

when <- dimension_schema(name = "When",
                         attributes = c("Year"))
where <- dimension_schema(name = "Where",
                          attributes = c("REGION",
                                         "State",
                                         "City"))
mrs_age_schema <- star_schema() |>
  define_facts(name = "MRS Age",
               measures = c("All Deaths")) |>
  define_dimension(when) |>
  define_dimension(where) |>
  define_dimension(name = "Who",
                   attributes = c("Age"))
```

---

mrs_age_schema_rpd *Star schema for Mortality Reporting System by Age with additional dates*

Description

Definition of schemas for facts and dimensions for the Mortality Reporting System considering the cause classification with additional dates to be used as role playing dimensions..

Usage

```
mrs_age_schema_rpd
```

Format

A star_schema object.

See Also

ft_age_rpd

Other mrs example schema: mrs_age_schema, mrs_cause_schema, mrs_cause_schema_rpd

Examples

```
# Defined by:

mrs_age_schema_rpd <- star_schema() |>
  define_facts(fact_schema(
    name = "mrs_age",
    measures = c(
      "Deaths"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When",
    attributes = c(
      "Year",
      "WEEK",
      "Week Ending Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When Available",
    attributes = c(
      "Data Availability Year",
      "Data Availability Week",
      "Data Availability Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When Arrived",
    attributes = c(
      "Arrival Year",
      "Arrival Week",
      "Arrival Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "Who",
    attributes = c(
      "Age Range"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "where",
    attributes = c(
      "REGION",
      "State",
      "City"
    )
```

```
))
```

`mrs_cause_schema` *Star schema for Mortality Reporting System by Cause*

Description

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

Usage

```
mrs_cause_schema
```

Format

A `star_schema` object.

Details

Dimension schemes can be defined using variables so that you do not have to repeat the definition in several multidimensional designs.

See Also

`ft_num`

Other mrs example schema: `mrs_age_schema`, `mrs_age_schema_rpd`, `mrs_cause_schema_rpd`

Examples

```
# Defined by:

when <- dimension_schema(name = "When",
                         attributes = c("Year"))
where <- dimension_schema(name = "Where",
                          attributes = c("REGION",
                                         "State",
                                         "City"))
mrs_cause_schema <- star_schema() |>
  define_facts(name = "MRS Cause",
               measures = c("Pneumonia and Influenza Deaths",
                            "All Deaths")) |>
  define_dimension(when) |>
  define_dimension(where)
```

`mrs_cause_schema_rpd` *Star schema for Mortality Reporting System by Cause with additional dates*

Description

Definition of schemas for facts and dimensions for the Mortality Reporting System considering the cause classification with additional dates to be used as role playing dimensions..

Usage

```
mrs_cause_schema_rpd
```

Format

A `star_schema` object.

See Also

`ft_cause_rpd`

Other mrs example schema: `mrs_age_schema`, `mrs_age_schema_rpd`, `mrs_cause_schema`

Examples

```
# Defined by:

mrs_cause_schema_rpd <- star_schema() |>
  define_facts(fact_schema(
    name = "mrs_cause",
    measures = c(
      "Pneumonia and Influenza Deaths",
      "All Deaths"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When",
    attributes = c(
      "Year",
      "WEEK",
      "Week Ending Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When Available",
    attributes = c(
      "Data Availability Year",
      "Data Availability Week",
      "Data Availability Date"
    )
```

```
)) |>
define_dimension(dimension_schema(
  name = "When Received",
  attributes = c(
    "Reception Year",
    "Reception Week",
    "Reception Date"
  )
)) |>
define_dimension(dimension_schema(
  name = "where",
  attributes = c(
    "REGION",
    "State",
    "City"
  )
))
```

`mrs_db` *Constellation generated from MRS file*

Description

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, from the original file, we have stored in the package a file with the same format as the original file but that includes only 1% of its data, selected at random.

Usage

```
mrs_db
```

Format

A `star_database`.

Details

From these data the constellation in the vignette titled 'Obtaining and transforming flat tables' has been generated. This variable contains the defined constellation.

Source

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

See Also

Other mrs example data: `ft`, `ft_age`, `ft_age_rpd`, `ft_cause_rpd`, `ft_num`, `mrs_db_geo`, `mrs_ft`, `mrs_ft_new`

mrs_db_geo *Constellation generated from MRS file through a query and with geographic information*

Description

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, from the original file, we have stored in the package a file with the same format as the original file but that includes only 1% of its data, selected at random.

Usage

```
mrs_db_geo
```

Format

A star_database.

Details

From these data the constellation in the vignette titled 'Obtaining and transforming flat tables' has been generated. This variable contains the defined constellation.

Source

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

See Also

Other mrs example data: ft, ft_age, ft_age_rpd, ft_cause_rpd, ft_num, mrs_db, mrs_ft, mrs_ft_new

Examples

```
# Defined by:

sq <- mrs_db |>
  star_query() |>
  select_dimension(name = "where",
                   attributes = "state") |>
  select_dimension(name = "when",
                   attributes = "year") |>
  select_fact(
    name = "mrs_age",
    measures = "all_deaths"
  )  |>
  select_fact(
    name = "mrs_cause",
    measures = "pneumonia_and_influenza_deaths"
```

```
  )

db <- mrs_db |>
  run_query(sq)

mrs_db_geo <- db |>
  define_geoattribute(
    dimension = "where",
    attribute = "state",
    from_layer = us_layer_state,
    by = "STUSPS"
  )
```

---

mrs_ft *Flat table generated from MRS file*

Description

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, from the original file, we have stored in the package a file with the same format as the original file but that includes only 1% of its data, selected at random.

Usage

```
mrs_ft
```

Format

A `flat_table`.

Source

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

See Also

Other mrs example data: `ft`, `ft_age`, `ft_age_rpd`, `ft_cause_rpd`, `ft_num`, `mrs_db`, `mrs_db_geo`, `mrs_ft_new`

mrs_ft_new *Flat table generated from MRS file*

Description

The original dataset covers from 1962 to 2016. For each week, in 122 US cities, from the original file, we have stored in the package a file with the same format as the original file but that includes only 0,1% of its data, selected at random to test the incremental refresh.

Usage

```
mrs_ft_new
```

Format

A `flat_table`.

Source

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

See Also

Other mrs example data: `ft`, `ft_age`, `ft_age_rpd`, `ft_cause_rpd`, `ft_num`, `mrs_db`, `mrs_db_geo`, `mrs_ft`

multiple_value_key *Multiple value key*

Description

Gets the keys that have multiple values associated with them. The first field in the table is the key, the rest of fields are the values.

Usage

```
multiple_value_key(tb, col_as_vector = NULL)
```

Arguments

`tb`	A `tibble`.
`col_as_vector`	A string, name of the column to include a vector of values.

Details

If a name is indicated in the `col_as_vector` parameter, it includes a column with the data in vector form to be used in other functions.

Value

A tibble.

Examples

```
tb <- unique(ft[, c('WEEK', 'Week Ending Date')])
mvk <- multiple_value_key(tb)
```

---

read_flat_table_file *Import flat table file*

Description

Reads a text file and creates a flat_table object. The file is expected to contain a flat table whose first row contains the name of the columns. All columns are considered to be of type String.

Usage

```
read_flat_table_file(name, file, sep = ",", page = NULL, unknown_value = NULL)
```

Arguments

name	A string, flat table name.
file	A string, name of a text file.
sep	Column separator character.
page	A string, name of the new field in which to include the name of the file.
unknown_value	A string, value used to replace empty and NA values in attributes.

Details

When multiple files are handled, the file name may contain information associated with the flat table, it could be the table page information if the name of a new field in which to store it is indicated in the page parameter.

We can also indicate the value that is used in the data with undefined values.

Value

A flat_table object.

See Also

star_database

Other flat table definition functions: as_star_database(), flat_table(), get_table(), get_unknown_value_defined(), get_unknown_values(), read_flat_table_folder()

Examples

```
file <-
  system.file("extdata/mrs",
              "mrs_122_us_cities_1962_2016_new.csv",
              package = "rolap")

ft <- read_flat_table_file('mrs_new', file)
```

`read_flat_table_folder` *Import all flat table files in a folder*

Description

Reads all text files in a folder and creates a `flat_table` object. Each file is expected to contain a flat table, all with the same structure, whose first row contains the name of the columns. All columns are considered to be of type String.

Usage

```
read_flat_table_folder(
  name,
  folder,
  sep = ",",
  page = NULL,
  unknown_value = NULL,
  same_columns = FALSE,
  snake_case = FALSE
)
```

Arguments

`name`	A string, flat table name.
`folder`	A string, folder name.
`sep`	Column separator character.
`page`	A string, name of the new field in which to include the name of the file.
`unknown_value`	A string, value used to replace empty and NA values in attributes.
`same_columns`	A boolean, indicates whether all tables have the same columns in the same order.
`snake_case`	A boolean, indicates if we want to transform the names of the columns to snake case.

Details

When multiple files are handled, the file name may contain information associated with the flat table, it could be the table page information if the name of a new field in which to store it is indicated.

We can also indicate the value that is used in the data with undefined values.

In some situations all the files have the same structure but the column names may change slightly. In these cases it can be useful to transform the names to snake case or consider for all the files the names of the columns of the first one. These operations can be indicated by the corresponding parameters.

Value

A `flat_table` object.

See Also

`star_database`

Other flat table definition functions: `as_star_database()`, `flat_table()`, `get_table()`, `get_unknown_value_defined()`, `get_unknown_values()`, `read_flat_table_file()`

Examples

```
file <- system.file("extdata/mrs", package = "rolap")

ft <- read_flat_table_folder('mrs_new', file)
```

---

remove_instances_without_measures

Remove instances without measures

Description

Delete instances that have all measures undefined.

Usage

```
remove_instances_without_measures(ft)

## S3 method for class 'flat_table'
remove_instances_without_measures(ft)
```

Arguments

`ft`	A `flat_table` object.

Value

A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  remove_instances_without_measures()
```

---

replace_attribute_values.flat_table

Replace instance values

Description

Given the values of a possible instance, for that combination, replace them with the new data values.

Usage

```
## S3 method for class 'flat_table'
replace_attribute_values(db, name = NULL, attributes = NULL, old, new)

replace_attribute_values(db, name, attributes, old, new)

## S3 method for class 'star_database'
replace_attribute_values(db, name, attributes = NULL, old, new)
```

Arguments

db	A flat_table or star_database object.
name	A string, dimension name.
attributes	A vector of strings, attribute names.
old	A vector of values.
new	A vector of values.

Value

A flat_table or star_database object.

See Also

star_database, flat_table

Other star database and flat table functions: get_attribute_names.flat_table(), get_measure_names.flat_table(), get_similar_attribute_values.flat_table(), get_similar_attribute_values_individually.flat_table(), get_unique_attribute_values.flat_table(), set_attribute_names.flat_table(), set_measure_names.flat_tal
snake_case.flat_table()

Examples

```
db <- star_database(mrs_cause_schema, ft_num) |>
  replace_attribute_values(name = "where",
    old = c('1', 'CT', 'Bridgeport'),
    new = c('1', 'CT', 'Hartford'))

db <- star_database(mrs_cause_schema, ft_num) |>
  replace_attribute_values(name = "where",
                           attributes = c('REGION', 'State'),
                           old = c('1', 'CT'),
                           new = c('2', 'CT'))

ft <- flat_table('iris', iris) |>
  replace_attribute_values(
    attributes = 'Species',
    old = c('setosa'),
    new = c('versicolor')
  )
```

---

replace_empty_values *Replace empty values with the unknown value*

Description

Transforms the given attributes by replacing the empty values with the unknown value.

Usage

```
replace_empty_values(ft, attributes, empty_values)

## S3 method for class 'flat_table'
replace_empty_values(ft, attributes = NULL, empty_values = NULL)
```

Arguments

ft	A flat_table object.
attributes	A vector of names.
empty_values	A vector of values that correspond to empty values.

Details

In addition to the NA or empty values, those indicated (e.g., "-") can be considered as empty values.

Value

A `flat_table` object.

See Also

`flat_table`

Other flat table transformation functions: `add_custom_column()`, `remove_instances_without_measures()`, `replace_string()`, `replace_unknown_values()`, `select_attributes()`, `select_instances()`, `select_instances_by_comparison()`, `select_measures()`, `separate_measures()`, `transform_attribute_format()`, `transform_from_values()`, `transform_to_attribute()`, `transform_to_measure()`, `transform_to_values()`

Examples

```
iris2 <- iris
iris2[10, 'Species'] <- NA
ft <- flat_table('iris', iris2) |>
  replace_empty_values()
```

---

replace_string *Replace strings*

Description

Transforms the given attributes by replacing the string values with the replacement value.

Usage

```
replace_string(ft, attributes, string, replacement)

## S3 method for class 'flat_table'
replace_string(ft, attributes = NULL, string, replacement = NULL)
```

Arguments

`ft`	A `flat_table` object.
`attributes`	A vector of strings, attribute names.
`string`	A character string to replace.
`replacement`	A replacement for matched string.

Value

A `flat_table` object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_unknown_values(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  replace_string(
    attributes = 'Species',
    string = c('set'),
    replacement = c('Set')
  )
```

---

replace_unknown_values

Replace unknown values with the given value

Description

Transforms the given attributes by replacing unknown values in them with the given value.

Usage

```
replace_unknown_values(ft, attributes, value)

## S3 method for class 'flat_table'
replace_unknown_values(ft, attributes = NULL, value)
```

Arguments

ft	A flat_table object.
attributes	A vector of names.
value	A value.

Value

A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
iris2 <- iris
iris2[10, 'Species'] <- NA
ft <- flat_table('iris', iris2) |>
  replace_empty_values() |>
  replace_unknown_values(value = "Not available")
```

---

role_playing_dimension

Define a role playing dimension and its associated dimensions

---

Description

The same dimension can play several roles in relation to the facts. We can define the main dimension and the dimensions that play different roles.

Usage

```
role_playing_dimension(db, rpd, roles, rpd_att_names, att_names)

## S3 method for class 'star_database'
role_playing_dimension(db, rpd, roles, rpd_att_names = FALSE, att_names = NULL)
```

Arguments

db	A star_database object.
rpd	A string, dimension name (role playing dimension).
roles	A vector of strings, dimension names (dimension roles).
rpd_att_names	A boolean, common attribute names taken from rpd dimension.
att_names	A vector of strings, common attribute names.

Details

As a result, all the dimensions will have the same instances and, if we deem it necessary, also the same name of their attributes (except the surrogate key).

Value

A star_database object.

See Also

star_schema, flat_table

Other star database definition functions: get_dimension_names(), get_dimension_table(), get_fact_names(), get_role_playing_dimension_names(), get_table_names(), group_dimension_instances(), star_database()

Examples

```
s <- star_schema() |>
  define_facts(fact_schema(
    name = "mrs_cause",
    measures = c(
      "Pneumonia and Influenza Deaths",
      "All Deaths"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When",
    attributes = c(
      "Year",
      "WEEK",
      "Week Ending Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When Available",
    attributes = c(
      "Data Availability Year",
      "Data Availability Week",
      "Data Availability Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "When Received",
    attributes = c(
      "Reception Year",
      "Reception Week",
      "Reception Date"
    )
  )) |>
  define_dimension(dimension_schema(
    name = "where",
    attributes = c(
      "REGION",
      "State",
      "City"
    )
```

```
  ))

db <- star_database(s, ft_cause_rpd) |>
  role_playing_dimension(
    rpd = "When",
    roles = c("When Available", "When Received"),
    rpd_att_names = TRUE
  )

db <- star_database(s, ft_cause_rpd) |>
  role_playing_dimension("When",
                         c("When Available", "When Received"),
                         att_names = c("Year", "Week", "Date"))
```

`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(db, sq)

## S3 method for class 'star_database'
run_query(db, sq)
```

Arguments

db	A `star_database` object.
sq	A `star_query` object.

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 `star_database` object.

See Also

Other query functions: `as_GeoPackage()`, `as_geolayer()`, `filter_dimension()`, `get_layer()`, `get_variable_description()`, `get_variables()`, `select_dimension()`, `select_fact()`, `set_layer()`, `set_variables()`, `star_query()`

Examples

```
sq <- mrs_db |>
  star_query() |>
  select_dimension(name = "where",
                   attributes = c("city", "state")) |>
  select_dimension(name = "when",
                   attributes = "year") |>
  select_fact(
    name = "mrs_age",
    measures = "all_deaths",
    agg_functions = "MAX"
  ) |>
  select_fact(
    name = "mrs_cause",
    measures = c("pneumonia_and_influenza_deaths", "all_deaths")
  ) |>
  filter_dimension(name = "when", week <= " 3") |>
  filter_dimension(name = "where", city == "Bridgeport")

mrs_db_2 <- mrs_db |>
  run_query(sq)
```

---

select_attributes *Select attributes of a flat table*

Description

Select only the indicated attributes from the flat table.

Usage

```
select_attributes(ft, attributes)

## S3 method for class 'flat_table'
select_attributes(ft, attributes)
```

Arguments

ft	A flat_table object.
attributes	A vector of names.

Value

A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  select_attributes(attributes = c('Species'))

ft <- flat_table('ft_num', ft_num) |>
  select_attributes(attributes = c('Year', 'WEEK', 'Week Ending Date'))
```

---

select_dimension *Select dimension*

Description

To add a dimension in a star_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(sq, name, attributes)

## S3 method for class 'star_query'
select_dimension(sq, name = NULL, attributes = NULL)
```

Arguments

sq	A star_query object.
name	A string, name of the dimension.
attributes	A vector of attribute names.

Value

A star_query object.

See Also

Other query functions: as_GeoPackage(), as_geolayer(), filter_dimension(), get_layer(), get_variable_description(), get_variables(), run_query(), select_fact(), set_layer(), set_variables(), star_query()

Examples

```
sq <- mrs_db |>
  star_query() |>
  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, another of aggregation functions and another of new names for measures are also indicated.

Usage

```
select_fact(sq, name, measures, agg_functions, new, nrow_agg)

## S3 method for class 'star_query'
select_fact(
  sq,
  name = NULL,
  measures = NULL,
  agg_functions = NULL,
  new = NULL,
  nrow_agg = NULL
)
```

Arguments

`sq`	A `star_query` object.
`name`	A string, name of the fact.
`measures`	A vector of measure names.
`agg_functions`	A vector of aggregation function names, each one for its corresponding measure. They can be SUM, MAX or MIN.
`new`	A vector of measure new names.
`nrow_agg`	A string, name of a new measure that represents the COUNT of rows aggregated for each resulting row.

Details

If there is only one fact table, it is the one that is considered if no name is indicated.

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

If no new names are given, the original names will be considered. If the aggregation function is different from the one defined by default, it will be included as a prefix to the name.

Value

A `star_query` object.

See Also

Other query functions: `as_GeoPackage()`, `as_geolayer()`, `filter_dimension()`, `get_layer()`, `get_variable_description()`, `get_variables()`, `run_query()`, `select_dimension()`, `set_layer()`, `set_variables()`, `star_query()`

Examples

```
sq <- mrs_db |>
  star_query()

sq_1 <- sq |>
  select_fact(
    name = "mrs_age",
    measures = "all_deaths",
    agg_functions = "MAX"
  )

sq_2 <- sq |>
  select_fact(name = "mrs_age",
              measures = "all_deaths")

sq_3 <- sq |>
  select_fact(name = "mrs_age")
```

---

select_instances *Select instances of a flat table by value*

Description

Select only the indicated instances from the flat table.

Usage

```
select_instances(ft, not, attributes, values)

## S3 method for class 'flat_table'
select_instances(ft, not = FALSE, attributes = NULL, values)
```

Arguments

ft	A flat_table object.
not	A boolean.
attributes	A vector of names.
values	A list of value vectors.

Details

Several values can be indicated for attributes (performs an OR operation) or several attributes and a value for each one (performs an AND operation).

If the parameter not is true, the reported values are those that are not included.

Value

A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  select_instances(attributes = c('Species'),
                   values = c('versicolor', 'virginica'))

ft <- flat_table('ft_num', ft_num) |>
  select_instances(
    not = TRUE,
    attributes = c('Year', 'WEEK'),
    values = list(c('1962', '2'), c('1964', '2'))
  )
```

---

select_instances_by_comparison

Select instances of a flat table by comparison

Description

Select only the indicated instances from the flat table by comparison.

Usage

```
select_instances_by_comparison(ft, not, attributes, comparisons, values)

## S3 method for class 'flat_table'
select_instances_by_comparison(
  ft,
  not = FALSE,
  attributes = NULL,
  comparisons,
  values
)
```

Arguments

ft	A flat_table object.
not	A boolean.
attributes	A list of name vectors.
comparisons	A list of comparison operator vectors.
values	A list of value vectors.

Details

The elements of the three parameter lists correspond (all three must have the same structure and length or be of length 1). AND is performed for each combination of attribute, operator and value within each element of each list and OR between elements of the lists.

If the parameter not is true, the negation operation will be applied to the result.

Value

A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  select_instances_by_comparison(attributes = 'Species',
                                 comparisons = '>=',
                                 values = 'v')

ft <- flat_table('ft_num', ft_num) |>
  select_instances_by_comparison(
```

```
    not = FALSE,
    attributes = c('Year', 'Year', 'WEEK'),
    comparisons = c('>=', '<=', '=='),
    values = c('1962', '1964', '2')
  )

ft <- flat_table('ft_num', ft_num) |>
  select_instances_by_comparison(
    not = FALSE,
    attributes = c('Year', 'Year', 'WEEK'),
    comparisons = c('>=', '<=', '=='),
    values = list(c('1962', '1964', '2'),
                  c('1962', '1964', '4'))
  )
```

`select_measures` *Select measures of a flat table*

Description

Select only the indicated measures from the flat table.

Usage

```
select_measures(ft, measures, na_rm)

## S3 method for class 'flat_table'
select_measures(ft, measures = NULL, na_rm = TRUE)
```

Arguments

`ft`	A `flat_table` object.
`measures`	A vector of names.
`na_rm`	A boolean, remove rows from output where all measure values are NA.

Value

A `flat_table` object.

See Also

`flat_table`

Other flat table transformation functions: `add_custom_column()`, `remove_instances_without_measures()`, `replace_empty_values()`, `replace_string()`, `replace_unknown_values()`, `select_attributes()`, `select_instances()`, `select_instances_by_comparison()`, `separate_measures()`, `transform_attribute_format()`, `transform_from_values()`, `transform_to_attribute()`, `transform_to_measure()`, `transform_to_values()`

Examples

```
ft <- flat_table('iris', iris) |>
  select_measures(measures = c('Sepal.Length', 'Sepal.Width'))
```

`separate_measures` *Separate measures in flat tables*

Description

Separate groups of measures into different flat tables. For each group we must indicate a name. If we indicate more names than groups of measures, the measures not included in other groups are also included in a new group.

Usage

```
separate_measures(ft, measures, names, na_rm)

## S3 method for class 'flat_table'
separate_measures(ft, measures = NULL, names = NULL, na_rm = TRUE)
```

Arguments

`ft`	A `flat_table` object.
`measures`	A list of string vectors, groups of measure names.
`names`	A list of string, measure group names.
`na_rm`	A boolean, remove rows from output where all measure values are NA.

Details

A list of flat tables is returned. It assign the names to the result list.

Value

A list of `flat_table` objects.

See Also

`flat_table`

Other flat table transformation functions: `add_custom_column()`, `remove_instances_without_measures()`, `replace_empty_values()`, `replace_string()`, `replace_unknown_values()`, `select_attributes()`, `select_instances()`, `select_instances_by_comparison()`, `select_measures()`, `transform_attribute_format()`, `transform_from_values()`, `transform_to_attribute()`, `transform_to_measure()`, `transform_to_values()`

Examples

```
lft <- flat_table('iris', iris) |>
  separate_measures(
    measures = list(
      c('Petal.Length'),
      c('Petal.Width'),
      c('Sepal.Length')
    ),
    names = c('PL', 'PW', 'SL', 'SW')
  )
```

`set_attribute_names.flat_table` *Rename attributes*

Description

Rename attributes in a flat table or a dimension in a star database.

Usage

```
## S3 method for class 'flat_table'
set_attribute_names(db, name = NULL, old = NULL, new)

set_attribute_names(db, name, old, new)

## S3 method for class 'star_database'
set_attribute_names(db, name, old = NULL, new)
```

Arguments

`db`	A `flat_table` or `star_database` object.
`name`	A string, dimension name.
`old`	A vector of names.
`new`	A vector of names.

Details

To rename the attributes there are three possibilities: 1) give only one vector with the new names for all the attributes; 2) a vector of old names and another of new names that must correspond; 3) a vector of new names whose names are the old names they replace.

Value

A `flat_table` or `star_database` object.

See Also

star_database, flat_table

Other star database and flat table functions: get_attribute_names.flat_table(), get_measure_names.flat_table(), get_similar_attribute_values.flat_table(), get_similar_attribute_values_individually.flat_table(), get_unique_attribute_values.flat_table(), replace_attribute_values.flat_table(), set_measure_names.fla snake_case.flat_table()

Examples

```
db <- star_database(mrs_cause_schema, ft_num) |>
  set_attribute_names(
    name = "where",
    new = c(
      "Region",
      "State",
      "City"
    )
  )

db <- star_database(mrs_cause_schema, ft_num) |>
  set_attribute_names(name = "where",
                      old = "REGION",
                      new = "Region")

new <- "Region"
names(new) <- "REGION"
db <- star_database(mrs_cause_schema, ft_num) |>
  set_attribute_names(name = "where",
                      new = new)

ft <- flat_table('iris', iris) |>
  set_attribute_names(
    old = 'Species',
    new = 'species')

new <- "species"
names(new) <- "Species"
ft <- flat_table('iris', iris) |>
  set_attribute_names(
    new = new)
```

---

set_layer *Set geographic layer*

Description

If for some reason we modify the geographic layer, for example, to add a new calculated variable, we can set that layer to become the new geographic layer of the geolayer object using this function.

Usage

```
set_layer(gl, layer)

## S3 method for class 'geolayer'
set_layer(gl, layer)
```

Arguments

gl	A geolayer object.
layer	A sf object.

Value

A geolayer object.

See Also

Other query functions: as_GeoPackage(), as_geolayer(), filter_dimension(), get_layer(), get_variable_description(), get_variables(), run_query(), select_dimension(), select_fact(), set_variables(), star_query()

Examples

```
gl <- mrs_db_geo |>
  as_geolayer()

l <- gl |>
  get_layer()

l$tpc_001 <- l$var_002 * 100 / l$var_001

gl <- gl |>
  set_layer(l)
```

---

set_measure_names.flat_table

Rename measures

Description

Rename measures in a flat table or in facts in a star database.

Usage

```
## S3 method for class 'flat_table'
set_measure_names(db, name = NULL, old = NULL, new)

set_measure_names(db, name, old, new)

## S3 method for class 'star_database'
set_measure_names(db, name = NULL, old = NULL, new)
```

Arguments

db	A flat_table or star_database object.
name	A string, fact name.
old	A vector of names.
new	A vector of names.

Details

To rename the measures there are three possibilities: 1) give only one vector with the new names for all the measures; 2) a vector of old names and another of new names that must correspond; 3) a vector of new names whose names are the old names they replace.

Value

A flat_table or star_database object.

See Also

star_database, flat_table

Other star database and flat table functions: get_attribute_names.flat_table(), get_measure_names.flat_table(), get_similar_attribute_values.flat_table(), get_similar_attribute_values_individually.flat_table(), get_unique_attribute_values.flat_table(), replace_attribute_values.flat_table(), set_attribute_names.f snake_case.flat_table()

Examples

```
db <- star_database(mrs_cause_schema, ft_num) |>
  set_measure_names(
    new = c(
      "Pneumonia and Influenza",
      "All",
      "Rows Aggregated"
    )
  )

ft <- flat_table('iris', iris) |>
  set_measure_names(
    old = c('Petal.Length', 'Petal.Width', 'Sepal.Length', 'Sepal.Width'),
    new = c('pl', 'pw', 'ls', 'sw'))
```

```
new <- c('pl', 'pw', 'ls', 'sw')
names(new) <- c('Petal.Length', 'Petal.Width', 'Sepal.Length', 'Sepal.Width')
ft <- flat_table('iris', iris) |>
  set_measure_names(
    new = new)
```

`set_variables` *Set variables layer*

Description

The variables layer includes the names and description through various fields of the variables contained in the reports.

Usage

```
set_variables(gl, variables, keep_all_variables_na)

## S3 method for class 'geolayer'
set_variables(gl, variables, keep_all_variables_na = FALSE)
```

Arguments

`gl`	A `geolayer` object.
`variables`	A `tibble` object.
`keep_all_variables_na`	A boolean, keep rows with all variables NA.

Details

When we set the variables layer, after filtering it, the data layer is also filtered keeping only the variables from the variables layer.

By default, rows that are NA for all variables are eliminated.

Value

A `sf` object.

See Also

Other query functions: `as_GeoPackage()`, `as_geolayer()`, `filter_dimension()`, `get_layer()`, `get_variable_description()`, `get_variables()`, `run_query()`, `select_dimension()`, `select_fact()`, `set_layer()`, `star_query()`

Examples

```
gl <- mrs_db_geo |>
  as_geolayer()

v <- gl |>
  get_variables()

v <- v |>
  dplyr::filter(year == '1966' | year == '2016')

gl_sel <- gl |>
  set_variables(v)
```

`snake_case.flat_table` *Transform names according to the snake case style*

Description

For flat tables, transform attribute and measure names according to the snake case style. For star databases, transform fact, dimension, measures, and attribute names according to the snake case style.

Usage

```
## S3 method for class 'flat_table'
snake_case(db)

snake_case(db)

## S3 method for class 'star_database'
snake_case(db)
```

Arguments

db	A `flat_table` or `star_database` object.

Details

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

Value

A `flat_table` or `star_database` object.

See Also

star_database, flat_table

Other star database and flat table functions: get_attribute_names.flat_table(), get_measure_names.flat_table(), get_similar_attribute_values.flat_table(), get_similar_attribute_values_individually.flat_table(), get_unique_attribute_values.flat_table(), replace_attribute_values.flat_table(), set_attribute_names.f set_measure_names.flat_table()

Examples

```
db <- star_database(mrs_cause_schema, ft_num) |>
  snake_case()

ft <- flat_table('iris', iris) |>
  snake_case()
```

---

star_database star_database *S3 class*

Description

A star_database object is created from a star_schema object and a flat table that contains the data from which database instances are derived.

Usage

```
star_database(schema, instances, unknown_value = NULL)
```

Arguments

schema	A star_schema object.
instances	A flat table to define the database instances according to the schema.
unknown_value	A string, value used to replace NA values in dimensions.

Details

Measures and measures of the star_schema must correspond to the names of the columns of the flat table.

Since NA values cause problems when doing Join operations between tables, you can indicate the value that will be used to replace them before doing these operations. If none is indicated, a default value is taken.

Value

A star_database object.

See Also

star_schema, flat_table

Other star database definition functions: get_dimension_names(), get_dimension_table(), get_fact_names(), get_role_playing_dimension_names(), get_table_names(), group_dimension_instances(), role_playing_dimension()

Examples

```
db <- star_database(mrs_cause_schema, ft_num)
```

---

star_query star_query *S3 class*

Description

An empty star_query object is created where we can select facts and measures, dimensions, dimension attributes and filter dimension rows.

Usage

```
star_query(db)

## S3 method for class 'star_database'
star_query(db)
```

Arguments

db	A star_database object.

Value

A star_query object.

See Also

Other query functions: as_GeoPackage(), as_geolayer(), filter_dimension(), get_layer(), get_variable_description(), get_variables(), run_query(), select_dimension(), select_fact(), set_layer(), set_variables()

Examples

```
sq <- mrs_db |>
  star_query()
```

star_schema *star_schema S3 class*

Description

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

Usage

```
star_schema()
```

Details

To get a star database (a star_database object) we need a flat table and a star_schema object. The definition of facts and dimensions in the star_schema object is made from the flat table columns.

Value

A star_schema object.

See Also

star_database

Other star schema definition functions: define_dimension(), define_facts(), dimension_schema(), fact_schema()

Examples

```
s <- star_schema()
```

summarize_layer *Summarize geometry of a layer*

Description

Groups the geometric elements of a layer according to the values of the indicated attribute.

Usage

```
summarize_layer(layer, attribute)
```

Arguments

layer	A sf object.
attribute	A string, attribute name.

Value

A sf object.

See Also

Other star database geographic attributes: check_geoattribute_geometry(), define_geoattribute(), get_geoattribute_geometries(), get_geoattributes(), get_layer_geometry(), get_point_geometry()

Examples

```
layer <-
  summarize_layer(us_layer_state, "REGION")
```

---

transform_attribute_format

Transform attribute format

Description

Transforms numeric attributes adapting their format as indicated.

Usage

```
transform_attribute_format(
  ft,
  attributes,
  width,
  decimal_places,
  k_sep,
  decimal_sep,
  space_filling
)

## S3 method for class 'flat_table'
transform_attribute_format(
  ft,
  attributes,
  width = 1,
  decimal_places = 0,
  k_sep = NULL,
  decimal_sep = NULL,
  space_filling = TRUE
)
```

Arguments

ft	A flat_table object.
attributes	A vector of strings, attribute names.
width	An integer, string length.
decimal_places	An integer, number of decimal places.
k_sep	A character, thousands separator used (It can not be changed).
decimal_sep	A character, decimal separator used (It can not be changed).
space_filling	A boolean, fill on the left with spaces (with '0' otherwise).

Details

If a number > 1 is specified in the width parameter, at least that length will be obtained in the result, padded with blanks on the left.

Value

ft A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_from_values(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  transform_to_attribute(measures = "Sepal.Length", decimal_places = 2) |>
  transform_attribute_format(
    attributes = "Sepal.Length",
    width = 5,
    decimal_places = 1
  )
```

---

transform_from_values *Transform attribute values into measure names*

Description

The values of an attribute will become measure names. There can only be one measure that will be from where the new defined measures take the values.

Usage

```
transform_from_values(ft, attribute)

## S3 method for class 'flat_table'
transform_from_values(ft, attribute = NULL)
```

Arguments

ft	A flat_table object.
attribute	A string, attribute that stores the measures names.

Value

A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_to_attribute(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  transform_to_values(attribute = 'Characteristic',
                      measure = 'Value',
                      id_reverse = 'id')
ft <- ft |>
  transform_from_values(attribute = 'Characteristic')
```

---

transform_to_attribute

Transform to attribute

Description

Transform measures into attributes. We can indicate if we want all the numbers in the result to have the same length and the number of decimal places.

Usage

```
transform_to_attribute(ft, measures, width, decimal_places, k_sep, decimal_sep)

## S3 method for class 'flat_table'
transform_to_attribute(
  ft,
  measures,
  width = 1,
  decimal_places = 0,
  k_sep = ",",
  decimal_sep = "."
)
```

Arguments

ft	A flat_table object.
measures	A vector of strings, measure names.
width	An integer, string length.
decimal_places	An integer, number of decimal places.
k_sep	A character, indicates thousands separator.
decimal_sep	A character, indicates decimal separator.

Details

If a number > 1 is specified in the width parameter, at least that length will be obtained in the result, padded with blanks on the left.

Value

ft A flat_table object.

See Also

flat_table

Other flat table transformation functions: add_custom_column(), remove_instances_without_measures(), replace_empty_values(), replace_string(), replace_unknown_values(), select_attributes(), select_instances(), select_instances_by_comparison(), select_measures(), separate_measures(), transform_attribute_format(), transform_from_values(), transform_to_measure(), transform_to_values()

Examples

```
ft <- flat_table('iris', iris) |>
  transform_to_attribute(
    measures = "Sepal.Length",
    width = 3,
    decimal_places = 2
  )
```

`transform_to_measure` *Transform to measure*

Description

Transform attributes into measures.

Usage

```
transform_to_measure(ft, attributes, k_sep, decimal_sep)

## S3 method for class 'flat_table'
transform_to_measure(ft, attributes, k_sep = NULL, decimal_sep = NULL)
```

Arguments

`ft`	A `flat_table` object.
`attributes`	A vector of strings, attribute names.
`k_sep`	A character, thousands separator to remove.
`decimal_sep`	A character, new decimal separator to use, if necessary.

Details

We can indicate a thousands indicator to remove and a decimal separator to use. The only decimal separators considered are "." and ",".

Value

ft A `flat_table` object.

See Also

`flat_table`

Other flat table transformation functions: `add_custom_column()`, `remove_instances_without_measures()`, `replace_empty_values()`, `replace_string()`, `replace_unknown_values()`, `select_attributes()`, `select_instances()`, `select_instances_by_comparison()`, `select_measures()`, `separate_measures()`, `transform_attribute_format()`, `transform_from_values()`, `transform_to_attribute()`, `transform_to_values()`

Examples

```
ft <- flat_table('iris', iris) |>
  transform_to_attribute(measures = "Sepal.Length", decimal_places = 2) |>
  transform_to_measure(attributes = "Sepal.Length", decimal_sep = ".")
```

`transform_to_values` *Transform measure names into attribute values*

Description

Transforms the measure names into values of a new attribute. The values of the measures will become values of the new measure that is indicated.

Usage

```
transform_to_values(ft, attribute, measure, id_reverse, na_rm)

## S3 method for class 'flat_table'
transform_to_values(
  ft,
  attribute = NULL,
  measure = NULL,
  id_reverse = NULL,
  na_rm = TRUE
)
```

Arguments

`ft`	A `flat_table` object.
`attribute`	A string, new attribute that will store the measures names.
`measure`	A string, new measure that will store the measure value.
`id_reverse`	A string, name of a new attribute that will store the row id.
`na_rm`	A boolean, remove rows from output where the value column is NA.

Details

If we wanted to perform the reverse operation later using the `transform_from_values` function, we would need to uniquely identify each original row. By indicating a value in the `id_reverse` parameter, an identifier is added that will allow us to always carry out the inverse operation.

Value

A `flat_table` object.

See Also

`flat_table`

Other flat table transformation functions: `add_custom_column()`, `remove_instances_without_measures()`, `replace_empty_values()`, `replace_string()`, `replace_unknown_values()`, `select_attributes()`, `select_instances()`, `select_instances_by_comparison()`, `select_measures()`, `separate_measures()`, `transform_attribute_format()`, `transform_from_values()`, `transform_to_attribute()`, `transform_to_measure()`

Examples

```
ft <- flat_table('iris', iris) |>
  transform_to_values(attribute = 'Characteristic',
                      measure = 'Value')

ft <- flat_table('iris', iris) |>
  transform_to_values(attribute = 'Characteristic',
                      measure = 'Value',
                      id_reverse = 'id')
```

---

update_according_to *Update a flat table according to another structure*

Description

Update a flat table with the operations of another structure based on a flat table.

Usage

```
update_according_to(ft, sdb, star, sdb_operations)

## S3 method for class 'flat_table'
update_according_to(ft, sdb, star = 1, sdb_operations = NULL)
```

Arguments

ft	A flat_table object.
sdb	A star_database object with defined modification operations.
star	A string or integer, star database name or index in constellation.
sdb_operations	A star_database object with new defined modification operations.

Value

A star_database_update object.

See Also

star_database

Other star database refresh functions: get_existing_fact_instances(), get_lookup_tables(), get_new_dimension_instances(), get_star_database(), get_star_schema(), get_transformation_code(), get_transformation_file(), incremental_refresh()

Examples

```
f1 <- flat_table('ft_num', ft_cause_rpd) |>
  as_star_database(mrs_cause_schema_rpd) |>
  replace_attribute_values(
    name = "When Available",
    old = c('1962', '11', '1962-03-14'),
    new = c('1962', '3', '1962-01-15')
  ) |>
  group_dimension_instances(name = "When")
f2 <- flat_table('ft_num2', ft_cause_rpd) |>
  update_according_to(f1)
```

---

us_census_state *Census of US States, by sex and age*

Description

Census of US States, by sex and age, obtained from the United States Census Bureau (USCB), American Community Survey (ACS). Obtained from the variables defined in reports, classifying the concepts according to the defined subjects.

Usage

```
us_census_state
```

Format

A `tibble`.

Details

U.S. Census Bureau. "Government Units: US and State: Census Years 1942 - 2022." Public Sector, PUB Public Sector Annual Surveys and Census of Governments, Table CG00ORG01, 2022, https://data.census.gov/table/GOVSTIMESERIES.CG00ORG01?q=census+state+year. Accessed on October 25, 2023.

Source

https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-data.2021.html

us_layer_state *Geographic layer of US States*

Description

Geographic layer with data from the States of the USA in polygon format, with simplified geometry so that it takes up less space.

Usage

```
us_layer_state
```

Format

A sf.

Details

It has been obtained from the geographic data included in the US census prepared by the U.S. Census Bureau.

Source

https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-data.2021.html

Index