Package ‘omopgenerics’

May 9, 2026

Title Methods and Classes for the OMOP Common Data Model

Version 1.3.7

Description Provides definitions of core classes and methods used by analytic pipelines that query the OMOP (Observational Medical Outcomes Partnership) common data model.

License Apache License (>= 2)

Encoding UTF-8

RoxygenNote 7.3.3

Imports cli, dbplyr (>= 2.5.1), dplyr, generics, glue, lifecycle, methods, purrr, rlang, snakecase, stringi, stringr, tidyr, vctrs

Depends R (>= 4.1)

Suggests bit64, CDMConnector, covr, duckdb, gt, here, jsonlite, knitr, omock, readr, rmarkdown, testthat (>= 3.0.0), withr

URL `https://darwin-eu.github.io/omopgenerics/`

BugReports `https://github.com/darwin-eu/omopgenerics/issues`

Config/testthat/edition 3

Config/testthat/parallel true

VignetteBuilder knitr

NeedsCompilation no

Author Martí Català [aut, cre] (ORCID: <`https://orcid.org/0000-0003-3308-9905`>),
Edward Burn [aut] (ORCID: <`https://orcid.org/0000-0002-9286-1128`>),
Mike Du [ctb] (ORCID: <`https://orcid.org/0000-0002-9517-8834`>),
Yuchen Guo [ctb] (ORCID: <`https://orcid.org/0000-0002-0847-4855`>),
Adam Black [ctb] (ORCID: <`https://orcid.org/0000-0001-5576-8701`>),
Marta Alcalde-Herraiz [ctb] (ORCID: <`https://orcid.org/0009-0002-4405-1814`>)

Maintainer Martí Català <`marti.catalasabate@ndorms.ox.ac.uk`>

Repository CRAN

Date/Publication 2026-03-07 06:10:20 UTC

Contents

`achillesColumns` *Required columns for each of the achilles result tables*

Description

Required columns for each of the achilles result tables

Usage

```
achillesColumns(table, version = "5.3", onlyRequired = lifecycle::deprecated())
```

Arguments

`table`	Table for which to see the required columns. One of "achilles_analysis", "achilles_results", or "achilles_results_dist".
`version`	Version of the OMOP Common Data Model.
`onlyRequired`	deprecated.

Value

Character vector with the column names

Examples

```
library(omopgenerics)
achillesColumns("achilles_analysis")
achillesColumns("achilles_results")
achillesColumns("achilles_results_dist")
```

`achillesTables` *Names of the tables that contain the results of achilles analyses*

Description

Names of the tables that contain the results of achilles analyses

Usage

```
achillesTables(version = "5.3")
```

Arguments

`version`	Version of the OMOP Common Data Model.

Value

Names of the tables that are contain the results from the achilles analyses

Examples

```
library(omopgenerics)
achillesTables()
```

`additionalColumns` *Identify variables in additional_name column*

Description

Identifies and returns the unique values in additional_name column.

Usage

```
additionalColumns(result)
```

Arguments

`result`	A tibble.

Value

Unique values of the additional name column.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> additionalColumns()
}
```

`addSettings` *Add settings columns to a* `<summarised_result>` *object*

Description

Add settings columns to a `<summarised_result>` object

Usage

```
addSettings(result, settingsColumn = settingsColumns(result))
```

Arguments

`result`	A `<summarised_result>` object.
`settingsColumn`	Settings to be added as columns, by default `settingsColumns(result)` will be added. If NULL or empty character vector, no settings will be added.

Value

A `<summarised_result>` object with the added setting columns.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> addSettings()
}
```

`assertCharacter` *Assert that an object is a character and fulfill certain conditions.*

Description

Assert that an object is a character and fulfill certain conditions.

Usage

```
assertCharacter(
  x,
  length = NULL,
  na = FALSE,
  null = FALSE,
  unique = FALSE,
  named = FALSE,
  minNumCharacter = 0,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Variable to check.
`length`	Required length. If `NULL` length is not checked.
`na`	Whether it can contain NA values.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique elements.
`named`	Whether it has to be named.
`minNumCharacter`	Minimum number of characters that all elements must have.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`assertChoice` *Assert that an object is within a certain oprtions.*

Description

Assert that an object is within a certain oprtions.

Usage

```
assertChoice(
  x,
  choices,
  length = NULL,
  na = FALSE,
  null = FALSE,
  unique = FALSE,
  named = FALSE,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Variable to check.
`choices`	Options that x is allowed to be.
`length`	Required length. If `NULL` length is not checked.
`na`	Whether it can contain NA values.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique elements.
`named`	Whether it has to be named.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`assertClass` *Assert that an object has a certain class.*

Description

Assert that an object has a certain class.

Usage

```
assertClass(
  x,
  class,
  length = NULL,
  null = FALSE,
  all = FALSE,
  extra = TRUE,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	To check.
`class`	Expected class or classes.
`length`	Required length. If `NULL` length is not checked.
`null`	Whether it can be NULL.
`all`	Whether it should have all the classes or only at least one of them.
`extra`	Whether the object can have extra classes.
`call`	Call argument that will be passed to `cli`.
`msg`	Custom error message.

`assertDate` *Assert Date*

Description

Assert Date

Usage

```
assertDate(
  x,
  length = NULL,
  na = FALSE,
  null = FALSE,
  unique = FALSE,
  named = FALSE,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Expression to check.
`length`	Required length.
`na`	Whether it can contain NA values.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique elements.
`named`	Whether it has to be named.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

Value

x

`assertList` *Assert that an object is a list.*

Description

Assert that an object is a list.

Usage

```
assertList(
  x,
  length = NULL,
  na = FALSE,
  null = FALSE,
  unique = FALSE,
  named = FALSE,
  class = NULL,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Variable to check.
`length`	Required length. If `NULL` length is not checked.
`na`	Whether it can contain NA values.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique elements.
`named`	Whether it has to be named.
`class`	Class that the elements must have.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`assertLogical` *Assert that an object is a logical.*

Description

Assert that an object is a logical.

Usage

```
assertLogical(
  x,
  length = NULL,
  na = FALSE,
  null = FALSE,
  unique = FALSE,
  named = FALSE,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Variable to check.
`length`	Required length. If `NULL` length is not checked.
`na`	Whether it can contain NA values.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique elements.
`named`	Whether it has to be named.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`assertNumeric` *Assert that an object is a numeric.*

Description

Assert that an object is a numeric.

Usage

```
assertNumeric(
  x,
  integerish = FALSE,
  min = -Inf,
  max = Inf,
  length = NULL,
  na = FALSE,
  null = FALSE,
  unique = FALSE,
  named = FALSE,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Variable to check.
`integerish`	Whether it has to be an integer
`min`	Minimum value that the object can be.
`max`	Maximum value that the object can be.
`length`	Required length. If `NULL` length is not checked.
`na`	Whether it can contain NA values.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique elements.
`named`	Whether it has to be named.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`assertTable` *Assert that an object is a table.*

Description

Assert that an object is a table.

Usage

```
assertTable(
  x,
  class = NULL,
  numberColumns = NULL,
  numberRows = NULL,
  columns = character(),
  allowExtraColumns = TRUE,
  null = FALSE,
  unique = FALSE,
  call = parent.frame(),
  msg = NULL
)
```

Arguments

`x`	Variable to check.
`class`	A class that the table must have: "tbl", "data.fram", "tbl_sql", ...
`numberColumns`	Number of columns that it has to contain.
`numberRows`	Number of rows that it has to contain.
`columns`	Name of the columns required.
`allowExtraColumns`	Whether extra columns are allowed.
`null`	Whether it can be NULL.
`unique`	Whether it has to contain unique rows.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`assertTrue` *Assert that an expression is TRUE.*

Description

Assert that an expression is TRUE.

Usage

```
assertTrue(x, null = FALSE, call = parent.frame(), msg = NULL)
```

Arguments

`x`	Expression to check.
`null`	Whether it can be NULL.
`call`	Call argument that will be passed to `cli` error message.
`msg`	Custom error message.

`attrition` *Get attrition from an object.*

Description

Get attrition from an object.

Usage

```
attrition(x)
```

Arguments

`x`	An object for which to get an attrition summary.

Value

A table with the attrition.

`attrition.cohort_table` *Get cohort attrition from a cohort_table object.*

Description

Get cohort attrition from a cohort_table object.

Usage

```
## S3 method for class 'cohort_table'
attrition(x)
```

Arguments

`x`	A cohort_table

Value

A table with the attrition.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort <- tibble(
  cohort_definition_id = c(1, 1, 1, 2),
  subject_id = 1,
 cohort_start_date = as.Date(c("2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01")),
  cohort_end_date = as.Date(c("2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01")),
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "my_example_cdm",
  cohortTables = list("cohort1" = cohort)
)

attrition(cdm$cohort1)
```

`bind` *Bind two or more objects of the same class.*

Description

Bind two or more objects of the same class.

Usage

```
bind(...)
```

Arguments

`...`	Objects to bind.

Value

New object.

`bind.cohort_table` *Bind two or more cohort tables*

Description

Bind two or more cohort tables

Usage

```
## S3 method for class 'cohort_table'
bind(..., name)
```

Arguments

`...`	Generated cohort set objects to bind. At least two must be provided.
`name`	Name of the new generated cohort set.

Value

The cdm object with a new generated cohort set containing all of the cohorts passed.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cohort1 <- tibble(
  cohort_definition_id = 1,
  subject_id = 1:3,
  cohort_start_date = as.Date("2010-01-01"),
  cohort_end_date = as.Date("2010-01-05")
)
cohort2 <- tibble(
  cohort_definition_id = c(2, 2, 3, 3, 3),
  subject_id = c(1, 2, 3, 1, 2),
  cohort_start_date = as.Date("2010-01-01"),
  cohort_end_date = as.Date("2010-01-05")
)
cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock",
  cohortTables = list("cohort1" = cohort1, "cohort2" = cohort2)
)

cdm <- bind(cdm$cohort1, cdm$cohort2, name = "cohort3")
settings(cdm$cohort3)
cdm$cohort3
```

`bind.summarised_result`

Bind two or summarised_result objects

Description

Bind two or summarised_result objects

Usage

```
## S3 method for class 'summarised_result'
bind(...)
```

Arguments

`...`	summarised_result objects

Value

A summarised_result object the merged objects.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock",
  cohortTables = list("cohort1" = tibble(
    cohort_definition_id = 1,
    subject_id = 1:3,
    cohort_start_date = as.Date("2010-01-01"),
    cohort_end_date = as.Date("2010-01-05")
  ))
)

result1 <- summary(cdm)
result2 <- summary(cdm$cohort1)

mergedResult <- bind(result1, result2)
mergedResult
```

`cdmClasses` *Separate the cdm tables in classes*

Description

Separate the cdm tables in classes

Usage

```
cdmClasses(cdm)
```

Arguments

`cdm`	A cdm_reference object.

Value

A list of table names, the name of the list indicates the class.

`cdmDisconnect` *Disconnect from a cdm object.*

Description

Disconnect from a cdm object.

Usage

```
cdmDisconnect(cdm, ...)
```

Arguments

`cdm`	A cdm reference or the source of a cdm reference.
`...`	Used for consistency.

Value

TRUE if process wass successful. library(omopgenerics) library(dplyr, warn.conflicts = FALSE)

person <- tibble(person_id = 1, gender_concept_id = 0, year_of_birth = 1990, race_concept_id = 0, ethnicity_concept_id = 0) observation_period <- tibble(observation_period_id = 1, person_id = 1, observation_period_start_date = as.Date("2000-01-01"), observation_period_end_date = as.Date("2023-12-31"), period_type_concept_id = 0) cdm <- cdmFromTables(tables = list("person" = person, "observation_period" = observation_period), cdmName = "mock")

cdmDisconnect(cdm)

`cdmFromTables` *Create a cdm object from local tables*

Description

Create a cdm object from local tables

Usage

```
cdmFromTables(tables, cdmName, cohortTables = list(), cdmVersion = NULL)
```

Arguments

`tables`	List of tables to be part of the cdm object.
`cdmName`	Name of the cdm object.
`cohortTables`	List of tables that contains cohort, cohort_set and cohort_attrition can be provided as attributes.
`cdmVersion`	Version of the cdm_reference

Value

A `cdm_reference` object.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)
```

`cdmName` *Get the name of a cdm_reference associated object*

Description

Get the name of a cdm_reference associated object

Usage

```
cdmName(x)
```

Arguments

`x`	A cdm_reference or cdm_table object.

Value

Name of the cdm_reference.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdmName(cdm)

cdmName(cdm$person)
```

`cdmReference` *Get the* `cdm_reference` *of a* `cdm_table`*.*

Description

Get the `cdm_reference` of a `cdm_table`.

Usage

```
cdmReference(table)
```

Arguments

`table`	A cdm_table.

Value

A cdm_reference.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdmReference(cdm$person)
```

`cdmSelect` *Restrict the cdm object to a subset of tables.*

Description

Restrict the cdm object to a subset of tables.

Usage

```
cdmSelect(cdm, ...)
```

Arguments

`cdm`	A cdm_reference object.
`...`	Selection of tables to use, it supports tidyselect expressions.

Value

A cdm_reference with only the specified tables.

Examples

```
cdm <- emptyCdmReference("my cdm")
cdm

cdm |>
  cdmSelect("person")
```

`cdmSource` *Get the cdmSource of an object.*

Description

Get the cdmSource of an object.

Usage

```
cdmSource(x, cdm = lifecycle::deprecated())
```

Arguments

`x`	Object to obtain the cdmSource.
`cdm`	Deprecated, use x please.

Value

A cdm_source object.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdmSource(cdm)
cdmSource(cdm$person)
```

`cdmSourceType` *Get the source type of a cdm_reference object.*

Description

[Deprecated]

Usage

```
cdmSourceType(cdm)
```

Arguments

`cdm`	A cdm_reference object.

Value

A character vector with the type of source of the cdm_reference object.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdmSourceType(cdm)
```

`cdmTableFromSource` *This is an internal developer focused function that creates a cdm_table from a table that shares the source but it is not a cdm_table. Please use insertTable if you want to insert a table to a cdm_reference object.*

Description

This is an internal developer focused function that creates a cdm_table from a table that shares the source but it is not a cdm_table. Please use insertTable if you want to insert a table to a cdm_reference object.

Usage

```
cdmTableFromSource(src, value)
```

Arguments

`src`	A cdm_source object.
`value`	A table that shares source with the cdm_reference object.

Value

A cdm_table.

`cdmVersion` *Get the version of an object.*

Description

Get the version of an object.

Usage

```
cdmVersion(x)
```

Arguments

`x`	Object to know the cdm version of an object.

Value

A character vector indicating the cdm version.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdmVersion(cdm)
cdmVersion(cdm$person)
```

`checkCohortRequirements`

Check whether a cohort table satisfies requirements

Description

[Deprecated]

Usage

```
checkCohortRequirements(
  cohort,
  checkEndAfterStart = TRUE,
  checkOverlappingEntries = TRUE,
  checkMissingValues = TRUE,
  checkInObservation = TRUE,
  type = "error",
  call = parent.frame()
)
```

Arguments

`cohort`	`cohort_table` object.
`checkEndAfterStart`	If TRUE a check that all cohort end dates come on or after cohort start date will be performed.
`checkOverlappingEntries`	If TRUE a check that no individuals have overlapping cohort entries will be performed.
`checkMissingValues`	If TRUE a check that there are no missing values in required fields will be performed.
`checkInObservation`	If TRUE a check that cohort entries are within the individuals observation periods will be performed.
`type`	Can be either "error" or "warning". If "error" any check failure will result in an error, whereas if "warning" any check failure will result in a warning.
`call`	The call for which to return the error message.

Value

An error will be returned if any of the selected checks fail.

`cohortCodelist` *Get codelist from a cohort_table object.*

Description

Get codelist from a cohort_table object.

Usage

```
cohortCodelist(
  cohort,
  cohortId = NULL,
  codelistType = c("index event", "inclusion criteria", "exit criteria"),
  type = lifecycle::deprecated(),
  cohortTable = lifecycle::deprecated()
)
```

Arguments

`cohort`	A cohort_table object.
`cohortId`	A particular cohort definition id that is present in the cohort table. If NULL the codelists of all cohorts will be retrieved.
`codelistType`	The reason for the codelist. Can be "index event", "inclusion criteria", or "exit criteria".
`type`	deprecated.
`cohortTable`	deprecated.

Value

A table with the codelists used.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort <- tibble(
```

```
  cohort_definition_id = c(1, 1, 1, 2),
  subject_id = 1,
  cohort_start_date = as.Date(c(
    "2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01"
  )),
  cohort_end_date = as.Date(c(
    "2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01"
  ))
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "my_example_cdm",
  cohortTables = list("cohort1" = cohort)
)
cdm$cohort1 <- newCohortTable(table = cdm$cohort1,
                                cohortCodelistRef = dplyr::tibble(
                                cohort_definition_id = c(1,1,1,2,2),
                                codelist_name =c("disease X", "disease X", "disease X",
                                                 "disease Y", "disease Y"),
                                concept_id = c(1,2,3,4,5),
                                codelist_type = "index event"
                              ))
cohortCodelist(cdm$cohort1, cohortId = 1, codelistType = "index event")
```

`cohortColumns` *Required columns for a generated cohort set.*

Description

Required columns for a generated cohort set.

Usage

```
cohortColumns(table, version = "5.3")
```

Arguments

`table`	Either `cohort`, `cohort_set` or `cohort_attrition`
`version`	Version of the OMOP Common Data Model.

Value

Character vector with the column names

Required columns

Examples

```
library(omopgenerics)
cohortColumns("cohort")
```

`cohortCount` *Get cohort counts from a cohort_table object.*

Description

Get cohort counts from a cohort_table object.

Usage

```
cohortCount(cohort)
```

Arguments

`cohort`	A cohort_table object.

Value

A table with the counts.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort <- tibble(
  cohort_definition_id = c(1, 1, 1, 2),
  subject_id = 1,
  cohort_start_date = as.Date(c(
    "2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01"
  )),
  cohort_end_date = as.Date(c(
    "2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01"
  )),
)
```

```
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "my_example_cdm",
  cohortTables = list("cohort1" = cohort)
)

cohortCount(cdm$cohort1)
```

`cohortTables` *Cohort tables that a cdm reference can contain in the OMOP Common Data Model.*

Description

Cohort tables that a cdm reference can contain in the OMOP Common Data Model.

Usage

```
cohortTables(version = "5.3")
```

Arguments

`version`	Version of the OMOP Common Data Model.

Value

cohort tables

Examples

```
library(omopgenerics)
cohortTables()
```

`collect.cdm_reference` *Retrieves the cdm reference into a local cdm.*

Description

Retrieves the cdm reference into a local cdm.

Usage

```
## S3 method for class 'cdm_reference'
collect(x, ...)
```

Arguments

`x`	A cdm_reference object.
`...`	For compatibility only, not used.

Value

A local cdm_reference.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = dplyr::tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = dplyr::tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

collect(cdm)
```

`collect.cohort_table` *To collect a* `cohort_table` *object.*

Description

To collect a `cohort_table` object.

Usage

```
## S3 method for class 'cohort_table'
collect(x, ...)
```

Arguments

`x`	`cohort_table` object.
`...`	Not used (for compatibility).

Value

A data frame with the `cohort_table`

`combineStrata` *Provide all combinations of strata levels.*

Description

Provide all combinations of strata levels.

Usage

```
combineStrata(levels, overall = FALSE)
```

Arguments

`levels`	Vector of all strata levels to combine.
`overall`	Whether to provide an empty element `character()`.

Value

A vector of all combinations of strata.

Examples

```
combineStrata(character())
combineStrata(character(), overall = TRUE)
combineStrata(c("age", "sex"), overall = TRUE)
combineStrata(c("age", "sex", "year"))
```

`compute.cdm_table` *Store results in a table.*

Description

Store results in a table.

Usage

```
## S3 method for class 'cdm_table'
compute(
  x,
  name = NULL,
  temporary = NULL,
  overwrite = TRUE,
  logPrefix = NULL,
  ...
)
```

Arguments

`x`	Table in the cdm.
`name`	Name to store the table with.
`temporary`	Whether to store table temporarily (TRUE) or permanently (FALSE).
`overwrite`	Whether to overwrite previously existing table with name same.
`logPrefix`	Prefix to use when saving a log file.
`...`	For compatibility (not used).

Value

Reference to a table in the cdm

`createIndexes` *Create the missing indexes*

Description

[Experimental]

Usage

```
createIndexes(cdm, name = NULL)
```

Arguments

`cdm`	A cdm_reference object.
`name`	Name(s) of the cdm tables.

Value

Whether the process was completed successfully.

`createLogFile` *Create a log file*

Description

Create a log file

Usage

```
createLogFile(logFile = here::here("log_{date}_{time}"))
```

Arguments

`logFile`	File path to write logging messages. You can use '{date}' and '{time}' to add the date and time in the log file name.

Value

Invisible TRUE if logger was created correctly.

Examples

```
library(dplyr)

logFile <- tempfile(pattern = "log_{date}_{time}", fileext = ".txt")
createLogFile(logFile = logFile)

logMessage("Starting analysis")
1 + 1
logMessage("Analysis finished")

res <- summariseLogFile()

glimpse(res)

tidy(res)
```

`createTableIndex` *Create a table index*

Description

[Experimental]

Usage

```
createTableIndex(table, index)
```

Arguments

`table`	A cdm_table object.
`index`	Index to be created.

Value

Whether the index could be created

`dropSourceTable` *Drop a table from a cdm object.*

Description

Drop a table from a cdm object.

Usage

```
dropSourceTable(cdm, name)
```

Arguments

`cdm`	A cdm reference.
`name`	Name(s) of the table(s) to insert. Tidyselect statements are supported.

Value

The table in the cdm reference.

`dropTable` *Drop a table from a cdm object.* **[Deprecated]**

Description

Drop a table from a cdm object. **[Deprecated]**

Usage

```
dropTable(cdm, name)
```

Arguments

`cdm`	A cdm reference.
`name`	Name(s) of the table(s) to drop Tidyselect statements are supported.

Value

The cdm reference.

`emptyAchillesTable` *Create an empty achilles table*

Description

Create an empty achilles table

Usage

```
emptyAchillesTable(cdm, name)
```

Arguments

`cdm`	A cdm_reference to create the table.
`name`	Name of the table to create.

Value

The cdm_reference with an achilles empty table

Examples

```
library(omopgenerics)
cdm <- emptyCdmReference("my_example_cdm")
emptyAchillesTable(cdm = cdm, name = "achilles_results")
```

`emptyCdmReference` *Create an empty cdm_reference*

Description

Create an empty cdm_reference

Usage

```
emptyCdmReference(cdmName, cdmVersion = NULL)
```

Arguments

`cdmName`	Name of the cdm_reference
`cdmVersion`	Version of the cdm_reference

Value

An empty cdm_reference

Examples

```
library(omopgenerics)
emptyCdmReference(cdmName = "my_example_cdm")
```

`emptyCodelist` *Empty* `codelist` *object.*

Description

Empty `codelist` object.

Usage

```
emptyCodelist()
```

Value

An empty codelist object.

Examples

```
emptyCodelist()
```

`emptyCodelistWithDetails` *Empty* `codelist` *object.*

Description

Empty `codelist` object.

Usage

```
emptyCodelistWithDetails()
```

Value

An empty codelist object.

Examples

```
emptyCodelistWithDetails()
```

`emptyCohortTable` *Create an empty cohort_table object*

Description

Create an empty cohort_table object

Usage

```
emptyCohortTable(cdm, name, overwrite = TRUE)
```

Arguments

`cdm`	A cdm_reference to create the table.
`name`	Name of the table to create.
`overwrite`	Whether to overwrite an existent table.

Value

The cdm_reference with an empty cohort table

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)

cdm <- emptyCohortTable(cdm, "my_empty_cohort")

cdm
cdm$my_empty_cohort
settings(cdm$my_empty_cohort)
attrition(cdm$my_empty_cohort)
cohortCount(cdm$my_empty_cohort)
```

`emptyConceptSetExpression` *Empty* `concept_set_expression` *object.*

Description

Empty `concept_set_expression` object.

Usage

```
emptyConceptSetExpression()
```

Value

An empty concept_set_expression object.

Examples

```
emptyConceptSetExpression()
```

`emptyOmopTable` *Create an empty omop table*

Description

Create an empty omop table

Usage

```
emptyOmopTable(cdm, name)
```

Arguments

`cdm`	A cdm_reference to create the table.
`name`	Name of the table to create.

Value

The cdm_reference with an empty cohort table

Examples

```
library(omopgenerics)

person <- dplyr::tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- dplyr::tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)

cdm <- emptyOmopTable(cdm, "drug_exposure")

cdm$drug_exposure
```

`emptySummarisedResult` *Empty* `summarised_result` *object.*

Description

Empty `summarised_result` object.

Usage

```
emptySummarisedResult(settings = NULL)
```

Arguments

`settings`	Tibble/data.frame with the settings of the empty summarised_result. It has to contain at least `result_id` column.

Value

An empty `summarised_result` object.

Examples

```
library(omopgenerics)

emptySummarisedResult()
```

`estimateTypeChoices` *Choices that can be present in* `estimate_type` *column.*

Description

Choices that can be present in `estimate_type` column.

Usage

```
estimateTypeChoices()
```

Value

A character vector with the options that can be present in `estimate_type` column in the summarised_result objects.

Examples

```
library(omopgenerics)

estimateTypeChoices()
```

`existingIndexes` *Existing indexes in a cdm object*

Description

[Experimental]

Usage

```
existingIndexes(cdm, name)
```

Arguments

`cdm`	A cdm_reference object.
`name`	Name(s) of the cdm tables.

Value

A tibble with 3 columns: `table_class` class of the table, `table_name` name of the table, and `existing_index` index definition.

`expectedIndexes` *Expected indexes in a cdm object*

Description

[Experimental]

Usage

```
expectedIndexes(cdm, name)
```

Arguments

`cdm`	A cdm_reference object.
`name`	Name(s) of the cdm tables.

Value

A tibble with 3 columns: `table_class` class of the table, `table_name` name of the table, and `expected_index` index definition.

`exportCodelist` *Export a codelist object.*

Description

Export a codelist object.

Usage

```
exportCodelist(x, path, type = "json")
```

Arguments

`x`	A codelist
`path`	Path to where files will be created.
`type`	Type of files to export. Currently 'json' and 'csv' are supported.

Value

Files with codelists

`exportConceptSetExpression` *Export a concept set expression.*

Description

Export a concept set expression.

Usage

```
exportConceptSetExpression(x, path, type = "json")
```

Arguments

`x`	A concept set expression
`path`	Path to where files will be created.
`type`	Type of files to export. Currently 'json' and 'csv' are supported.

Value

Files with concept set expressions

`exportSummarisedResult` *Export a summarised_result object to a csv file.*

Description

Export a summarised_result object to a csv file.

Usage

```
exportSummarisedResult(
  ...,
  minCellCount = 5,
  fileName = "results_{cdm_name}_{date}.csv",
  path = getwd(),
  logFile = getOption("omopgenerics.logFile"),
  logSqlPath = getOption("omopgenerics.log_sql_path"),
  logExplainPath = lifecycle::deprecated()
)
```

Arguments

`...`	A set of summarised_result objects.
`minCellCount`	Minimum count for suppression purposes.
`fileName`	Name of the file that will be created. Use {cdm_name} to refer to the cdmName of the objects and {date} to add the export date.
`path`	Path where to create the csv file. It is ignored if fileName it is a full name with path included.
`logFile`	Path to the log file to export.
`logSqlPath`	Path to the folder that contains the sql logs to export.
`logExplainPath`	deprecated.

`filterAdditional` *Filter the additional_name-additional_level pair in a summarised_result*

Description

Filter the additional_name-additional_level pair in a summarised_result

Usage

```
filterAdditional(result, ...)
```

Arguments

`result`	A <`summarised_result`> object.
`...`	Expressions that return a logical value (`additionalColumns()` are used to evaluate the expression), and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.

Value

A <`summarised_result`> object with only the rows that fulfill the required specified additional.

Examples

```
library(dplyr)
library(omopgenerics)

x <- tibble(
  "result_id" = 1L,
  "cdm_name" = "eunomia",
  "group_name" = "cohort_name",
  "group_level" = c("cohort1", "cohort2", "cohort3"),
  "strata_name" = "sex",
```

```
    "strata_level" = "Female",
    "variable_name" = "number subjects",
    "variable_level" = NA_character_,
    "estimate_name" = "count",
    "estimate_type" = "integer",
    "estimate_value" = c("100", "44", "14"),
    "additional_name" = c("year", "time_step", "year &&& time_step"),
    "additional_level" = c("2010", "4", "2015 &&& 5")
  ) |>
    newSummarisedResult()

  x |>
    filterAdditional(year == "2010")
```

`filterGroup` *Filter the group_name-group_level pair in a summarised_result*

Description

Filter the group_name-group_level pair in a summarised_result

Usage

```
filterGroup(result, ...)
```

Arguments

`result`	A <`summarised_result`> object.
`...`	Expressions that return a logical value (`groupColumns()` are used to evaluate the expression), and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.

Value

A <`summarised_result`> object with only the rows that fulfill the required specified group.

Examples

```
library(dplyr)
library(omopgenerics)

x <- tibble(
  "result_id" = 1L,
  "cdm_name" = "eunomia",
  "group_name" = c("cohort_name", "age_group &&& cohort_name", "age_group"),
  "group_level" = c("my_cohort", ">40 &&& second_cohort", "<40"),
  "strata_name" = "sex",
```

```
    "strata_level" = "Female",
    "variable_name" = "number subjects",
    "variable_level" = NA_character_,
    "estimate_name" = "count",
    "estimate_type" = "integer",
    "estimate_value" = c("100", "44", "14"),
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult()

  x |>
    filterGroup(cohort_name == "second_cohort")
```

`filterSettings` *Filter a* `<summarised_result>` *using the settings*

Description

Filter a `<summarised_result>` using the settings

Usage

```
filterSettings(result, ...)
```

Arguments

`result`	A `<summarised_result>` object.
`...`	Expressions that return a logical value (columns in settings are used to evaluate the expression), and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.

Value

A `<summarised_result>` object with only the result_id rows that fulfill the required specified settings.

Examples

```
library(dplyr)
library(omopgenerics)

x <- tibble(
  "result_id" = as.integer(c(1, 2)),
  "cdm_name" = c("cprd", "eunomia"),
  "group_name" = "cohort_name",
  "group_level" = "my_cohort",
```

```
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> filterSettings(custom == "A")
```

`filterStrata` *Filter the strata_name-strata_level pair in a summarised_result*

Description

Filter the strata_name-strata_level pair in a summarised_result

Usage

```
filterStrata(result, ...)
```

Arguments

`result`	A `<summarised_result>` object.
`...`	Expressions that return a logical value (`strataColumns()` are used to evaluate the expression), and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.

Value

A `<summarised_result>` object with only the rows that fulfill the required specified strata.

Examples

```
library(dplyr)
library(omopgenerics)

x <- tibble(
  "result_id" = 1L,
```

```
    "cdm_name" = "eunomia",
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = c("sex", "sex &&& age_group", "sex &&& year"),
    "strata_level" = c("Female", "Male &&& <40", "Female &&& 2010"),
    "variable_name" = "number subjects",
    "variable_level" = NA_character_,
    "estimate_name" = "count",
    "estimate_type" = "integer",
    "estimate_value" = c("100", "44", "14"),
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult()

  x |>
    filterStrata(sex == "Female")
```

`getCohortId` *Get the cohort definition id of a certain name*

Description

Get the cohort definition id of a certain name

Usage

```
getCohortId(cohort, cohortName = NULL)
```

Arguments

`cohort`	A cohort_table object.
`cohortName`	Names of the cohort of interest. If NULL all cohort names are shown.

Value

Cohort definition ids

`getCohortName` *Get the cohort name of a certain cohort definition id*

Description

Get the cohort name of a certain cohort definition id

Usage

```
getCohortName(cohort, cohortId = NULL)
```

Arguments

`cohort`	A cohort_table object.
`cohortId`	Cohort definition id of interest. If NULL all cohort ids are shown.

Value

Cohort names

`getPersonIdentifier` *Get the column name with the person identifier from a table (either subject_id or person_id), it will throw an error if it contains both or neither.*

Description

Get the column name with the person identifier from a table (either subject_id or person_id), it will throw an error if it contains both or neither.

Usage

```
getPersonIdentifier(x, call = parent.frame())
```

Arguments

`x`	A table.
`call`	A call argument passed to cli functions.

Value

Person identifier column.

`groupColumns` *Identify variables in group_name column*

Description

Identifies and returns the unique values in group_name column.

Usage

```
groupColumns(result)
```

Arguments

`result`	A tibble.

Value

Unique values of the group name column.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> groupColumns()
}
```

`importCodelist` *Import a codelist.*

Description

Import a codelist.

Usage

```
importCodelist(path, type = "json")
```

Arguments

`path`	Path to where files will be created.
`type`	Type of files to export. Currently 'json' and 'csv' are supported.

Value

A codelist

`importConceptSetExpression` *Import a concept set expression.*

Description

Import a concept set expression.

Usage

```
importConceptSetExpression(path, type = "json")
```

Arguments

`path`	Path to where files will be created.
`type`	Type of files to export. Currently 'json' and 'csv' are supported.

Value

A concept set expression

`importSummarisedResult` *Import a set of summarised results.*

Description

Import a set of summarised results.

Usage

```
importSummarisedResult(path, recursive = FALSE, ...)
```

Arguments

`path`	Path to directory with CSV files containing summarised results or to a specific CSV file with a summarised result.
`recursive`	If TRUE and path is a directory, search for files will recurse into directories
`...`	Passed to `readr::read_csv`.

Value

A summarised result

`insertCdmTo` *Insert a cdm_reference object to a different source.*

Description

Insert a cdm_reference object to a different source.

Usage

```
insertCdmTo(cdm, to)
```

Arguments

`cdm`	A cdm_reference, if not local it will be collected into memory.
`to`	A cdm_source or another cdm_reference, with a valid cdm_source.

Value

The first cdm_reference object inserted to the source.

`insertFromSource` *Convert a table that is not a cdm_table but have the same original source to a cdm_table. This Table is not meant to be used to insert tables in the cdm, please use insertTable instead.*

Description

[Deprecated]

Usage

```
insertFromSource(cdm, value)
```

Arguments

`cdm`	A cdm_reference object.
`value`	A table that shares source with the cdm_reference object.

Value

A table in the cdm_reference environment

`insertTable` *Insert a table to a cdm object.*

Description

Insert a table to a cdm object.

Usage

```
insertTable(cdm, name, table, overwrite = TRUE, temporary = FALSE, ...)
```

Arguments

`cdm`	A cdm reference or the source of a cdm reference.
`name`	Name of the table to insert.
`table`	Table to insert to the cdm.
`overwrite`	Whether to overwrite an existent table.
`temporary`	Whether to create a temporary table.
`...`	For compatibility.

Value

The cdm reference. library(omopgenerics) library(dplyr, warn.conflicts = FALSE)

person <- tibble(person_id = 1, gender_concept_id = 0, year_of_birth = 1990, race_concept_id = 0, ethnicity_concept_id = 0) observation_period <- tibble(observation_period_id = 1, person_id = 1, observation_period_start_date = as.Date("2000-01-01"), observation_period_end_date = as.Date("2023-12-31"), period_type_concept_id = 0) cdm <- cdmFromTables(tables = list("person" = person, "observation_period" = observation_period), cdmName = "my_example_cdm")

x <- tibble(a = 1)

cdm <- insertTable(cdm = cdm, name = "new_table", table = x)

cdm$new_table

`isResultSuppressed` *To check whether an object is already suppressed to a certain min cell count.*

Description

To check whether an object is already suppressed to a certain min cell count.

Usage

```
isResultSuppressed(result, minCellCount = 5)
```

Arguments

`result`	The suppressed result to check
`minCellCount`	Minimum count of records used when suppressing

Value

Warning or message with check result

Examples

```
x <- dplyr::tibble(
  "result_id" = 1L,
  "cdm_name" = "eunomia",
  "group_name" = "cohort_name",
  "group_level" = "my_cohort",
  "strata_name" = c("sex", "sex &&& age_group", "sex &&& year"),
  "strata_level" = c("Female", "Male &&& <40", "Female &&& 2010"),
  "variable_name" = "number subjects",
  "variable_level" = NA_character_,
  "estimate_name" = "count",
  "estimate_type" = "integer",
  "estimate_value" = c("100", "44", "14"),
  "additional_name" = "overall",
```

```
    "additional_level" = "overall"
  ) |>
    newSummarisedResult()

  isResultSuppressed(x)
```

`isTableEmpty` *Check if a table is empty or not*

Description

Check if a table is empty or not

Usage

```
isTableEmpty(table)
```

Arguments

`table`	a table

Value

Boolean to indicate if a cdm_table is empty (TRUE or FALSE).

`listSourceTables` *List tables that can be accessed though a cdm object.*

Description

List tables that can be accessed though a cdm object.

Usage

```
listSourceTables(cdm)
```

Arguments

`cdm`	A cdm reference or the source of a cdm reference.

Value

A character vector with the names of tables.

`logMessage` *Log a message to a logFile*

Description

The message is written to the logFile and displayed in the console, if `logFile` does not exist the message is only displayed in the console.

Usage

```
logMessage(
  message = "Start logging file",
  logFile = getOption("omopgenerics.logFile")
)
```

Arguments

`message`	Message to log.
`logFile`	File path to write logging messages. Create a logFile with `createLogFile()`.

Value

Invisible TRUE if the logging message is written to a log file.

Examples

```
library(dplyr)

logFile <- tempfile(pattern = "log_{date}_{time}", fileext = ".txt")
createLogFile(logFile = logFile)

logMessage("Starting analysis")
1 + 1
logMessage("Analysis finished")

res <- summariseLogFile()

glimpse(res)

tidy(res)
```

`newAchillesTable` *Create an achilles table from a cdm_table.*

Description

Create an achilles table from a cdm_table.

Usage

```
newAchillesTable(table, version = "5.3", cast = FALSE)
```

Arguments

`table`	A cdm_table.
`version`	version of the cdm.
`cast`	Whether to cast columns to the correct type.

Value

An achilles_table object

`newCdmReference` `cdm_reference` *objects constructor*

Description

`cdm_reference` objects constructor

Usage

```
newCdmReference(tables, cdmName, cdmVersion = NULL, .softValidation = FALSE)
```

Arguments

`tables`	List of tables that are part of the OMOP Common Data Model reference.
`cdmName`	Name of the cdm object.
`cdmVersion`	Version of the cdm. Supported versions 5.3 and 5.4.
`.softValidation`	Whether to perform a soft validation of consistency. If set to FALSE, non overlapping observation periods are ensured.

Value

A `cdm_reference` object.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdmTables <- list(
  "person" = tibble(
    person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
    race_concept_id = 0, ethnicity_concept_id = 0
  ) |>
    newCdmTable(newLocalSource(), "person"),
  "observation_period" = tibble(
    observation_period_id = 1, person_id = 1,
    observation_period_start_date = as.Date("2000-01-01"),
    observation_period_end_date = as.Date("2023-12-31"),
    period_type_concept_id = 0
  ) |>
    newCdmTable(newLocalSource(), "observation_period")
)
cdm <- newCdmReference(tables = cdmTables, cdmName = "mock")

cdm
```

`newCdmSource` *Create a cdm source object.*

Description

Create a cdm source object.

Usage

```
newCdmSource(src, sourceType)
```

Arguments

`src`	Source to a cdm object.
`sourceType`	Type of the source object.

Value

A validated cdm source object.

`newCdmTable` *Create an cdm table.*

Description

Create an cdm table.

Usage

```
newCdmTable(table, src, name)
```

Arguments

`table`	A table that is part of a cdm.
`src`	The source of the table.
`name`	The name of the table.

Value

A cdm_table object

`newCodelist` *'codelist' object constructor*

Description

'codelist' object constructor

Usage

```
newCodelist(x)
```

Arguments

`x`	A named list where each element contains a vector of concept IDs.

Value

A codelist object.

`newCodelistWithDetails` *'codelist' object constructor*

Description

'codelist' object constructor

Usage

```
newCodelistWithDetails(x)
```

Arguments

`x`	A named list where each element contains a tibble with the column concept_id

Value

A codelist object.

`newCohortTable` `cohort_table` *objects constructor.*

Description

`cohort_table` objects constructor.

Usage

```
newCohortTable(
  table,
  cohortSetRef = attr(table, "cohort_set"),
  cohortAttritionRef = attr(table, "cohort_attrition"),
  cohortCodelistRef = attr(table, "cohort_codelist"),
  .softValidation = FALSE
)
```

Arguments

`table`	cdm_table object with at least: cohort_definition_id, subject_id, cohort_start_date, cohort_end_date.
`cohortSetRef`	Table with at least: cohort_definition_id, cohort_name
`cohortAttritionRef`	Table with at least: cohort_definition_id, number_subjects, number_records, reason_id, reason, excluded_subjects, excluded_records.

`cohortCodelistRef`	Table with at least: cohort_definition_id, codelist_name, concept_id and codelist_type.
`.softValidation`	Whether to perform a soft validation of consistency. If set to FALSE four additional checks will be performed: 1) a check that cohort end date is not before cohort start date, 2) a check that there are no missing values in required columns, 3) a check that cohort duration is all within observation period, and 4) that there are no overlapping cohort entries

Value

A cohort_table object

Examples

```
person <- dplyr::tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- dplyr::tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort1 <- dplyr::tibble(
  cohort_definition_id = 1, subject_id = 1,
  cohort_start_date = as.Date("2020-01-01"),
  cohort_end_date = as.Date("2020-01-10")
)
cdm <- cdmFromTables(
  tables = list(
    "person" = person,
    "observation_period" = observation_period,
    "cohort1" = cohort1
  ),
  cdmName = "test"
)
cdm
cdm$cohort1 <- newCohortTable(table = cdm$cohort1)
cdm
settings(cdm$cohort1)
attrition(cdm$cohort1)
cohortCount(cdm$cohort1)
```

---

newConceptSetExpression

'concept_set_expression' object constructor

Description

’concept_set_expression’ object constructor

Usage

```
newConceptSetExpression(x)
```

Arguments

`x`	a named list of tibbles, each of which containing concept set definitions

Value

A concept_set_expression

`newLocalSource` *A new local source for the cdm*

Description

A new local source for the cdm

Usage

```
newLocalSource()
```

Value

A list in the format of a cdm source

Examples

```
library(omopgenerics)
newLocalSource()
```

`newOmopTable` *Create an omop table from a cdm table.*

Description

Create an omop table from a cdm table.

Usage

```
newOmopTable(table, version = "5.3", cast = FALSE)
```

Arguments

`table`	A cdm_table.
`version`	version of the cdm.
`cast`	Whether to cast columns to the correct type.

Value

An omop_table object

`newSummarisedResult` *'summarised_results' object constructor*

Description

'summarised_results' object constructor

Usage

```
newSummarisedResult(x, settings = attr(x, "settings"))
```

Arguments

`x`	Table.
`settings`	Settings for the summarised_result object.

Value

A `summarised_result` object

Examples

```
library(dplyr)
library(omopgenerics)

x <- tibble(
  "result_id" = 1L,
  "cdm_name" = "cprd",
  "group_name" = "cohort_name",
  "group_level" = "acetaminophen",
  "strata_name" = "sex &&& age_group",
  "strata_level" = c("male &&& <40", "male &&& >=40"),
  "variable_name" = "number_subjects",
  "variable_level" = NA_character_,
  "estimate_name" = "count",
  "estimate_type" = "integer",
  "estimate_value" = c("5", "15"),
  "additional_name" = "overall",
  "additional_level" = "overall"
) |>
  newSummarisedResult()

x
settings(x)
summary(x)

x <- tibble(
  "result_id" = 1L,
  "cdm_name" = "cprd",
  "group_name" = "cohort_name",
  "group_level" = "acetaminophen",
  "strata_name" = "sex &&& age_group",
  "strata_level" = c("male &&& <40", "male &&& >=40"),
  "variable_name" = "number_subjects",
  "variable_level" = NA_character_,
  "estimate_name" = "count",
  "estimate_type" = "integer",
  "estimate_value" = c("5", "15"),
  "additional_name" = "overall",
  "additional_level" = "overall"
) |>
  newSummarisedResult(settings = tibble(
    result_id = 1L, result_type = "custom_summary", mock = TRUE, value = 5
  ))

x
settings(x)
summary(x)
```

`numberRecords` *Count the number of records that a* `cdm_table` *has.*

Description

Count the number of records that a `cdm_table` has.

Usage

```
numberRecords(x)
```

Arguments

`x`	A cdm_table.

Value

An integer with the number of records in the table.

Examples

```
person <- dplyr::tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- dplyr::tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)

numberRecords(cdm$observation_period)
```

`numberSubjects` *Count the number of subjects that a* `cdm_table` *has.*

Description

Count the number of subjects that a `cdm_table` has.

Usage

```
numberSubjects(x)
```

Arguments

`x`	A cdm_table.

Value

An integer with the number of subjects in the table.

Examples

```
person <- dplyr::tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- dplyr::tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)

numberSubjects(cdm$observation_period)
```

`omopColumns` *Required columns that the standard tables in the OMOP Common Data Model must have.*

Description

Required columns that the standard tables in the OMOP Common Data Model must have.

Usage

```
omopColumns(
  table,
  field = NULL,
  version = "5.3",
  onlyRequired = lifecycle::deprecated()
)
```

Arguments

`table`	Table to see required columns.
`field`	Name of the specific field.
`version`	Version of the OMOP Common Data Model.
`onlyRequired`	deprecated

Value

Character vector with the column names

Examples

```
library(omopgenerics)

omopColumns("person")
```

`omopDataFolder` *Check or set the OMOP_DATA_FOLDER where the OMOP related data is stored.*

Description

Check or set the OMOP_DATA_FOLDER where the OMOP related data is stored.

Usage

```
omopDataFolder(path = NULL)
```

Arguments

`path`	Path to a folder to store the OMOP related data. If NULL the current `OMOP_DATA_FOLDER` is returned.

Value

The OMOP data folder.

Examples

```
omopDataFolder()
omopDataFolder(file.path(tempdir(), "OMOP_DATA"))
omopDataFolder()
```

`omopTableFields` *Return a table of omop cdm fields informations*

Description

Return a table of omop cdm fields informations

Usage

```
omopTableFields(cdmVersion = "5.3")
```

Arguments

`cdmVersion`	cdm version of the omop cdm.

Value

a tibble contain informations on all the different fields in omop cdm.

`omopTables` *Standard tables that a cdm reference can contain in the OMOP Common Data Model.*

Description

Standard tables that a cdm reference can contain in the OMOP Common Data Model.

Usage

```
omopTables(version = "5.3")
```

Arguments

`version`	Version of the OMOP Common Data Model.

Value

Standard tables

Examples

```
library(omopgenerics)

omopTables()
```

`pivotEstimates` *Set estimates as columns*

Description

Pivot the estimates as new columns in result table.

Usage

```
pivotEstimates(result, pivotEstimatesBy = "estimate_name", nameStyle = NULL)
```

Arguments

`result`	A <summarised_result>.
`pivotEstimatesBy`	Names from which pivot wider the estimate values. If NULL the table will not be pivotted.
`nameStyle`	Name style (glue package specifications) to customise names when pivotting estimates. If NULL standard tidyr::pivot_wider formatting will be used.

Value

A tibble.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = 1L,
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult()

  x |>
    pivotEstimates()
```

```
}
```

`print.cdm_reference` *Print a CDM reference object*

Description

Print a CDM reference object

Usage

```
## S3 method for class 'cdm_reference'
print(x, ...)
```

Arguments

`x`	A cdm_reference object
`...`	Included for compatibility with generic. Not used.

Value

Invisibly returns the input

Examples

```
library(omopgenerics)

cdm <- cdmFromTables(
  tables = list(
    "person" = dplyr::tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = dplyr::tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

print(cdm)
```

`print.codelist` *Print a codelist*

Description

Print a codelist

Usage

```
## S3 method for class 'codelist'
print(x, ...)
```

Arguments

`x`	A codelist
`...`	Included for compatibility with generic. Not used.

Value

Invisibly returns the input

Examples

```
codes <- list("disease X" = c(1, 2, 3), "disease Y" = c(4, 5))
codes <- newCodelist(codes)
print(codes)
```

`print.codelist_with_details` *Print a codelist with details*

Description

Print a codelist with details

Usage

```
## S3 method for class 'codelist_with_details'
print(x, ...)
```

Arguments

`x`	A codelist with details
`...`	Included for compatibility with generic. Not used.

Value

Invisibly returns the input

Examples

```
codes <- list("disease X" = dplyr::tibble(
  concept_id = c(1, 2, 3),
  other = c("a", "b", "c")
))
codes <- newCodelistWithDetails(codes)
print(codes)
```

`print.concept_set_expression`
Print a concept set expression

Description

Print a concept set expression

Usage

```
## S3 method for class 'concept_set_expression'
print(x, ...)
```

Arguments

`x`	A concept set expression
`...`	Included for compatibility with generic. Not used.

Value

Invisibly returns the input

Examples

```
asthma_cs <- list(
  "asthma_narrow" = dplyr::tibble(
    "concept_id" = 1,
    "excluded" = FALSE,
    "descendants" = TRUE,
    "mapped" = FALSE
  ),
  "asthma_broad" = dplyr::tibble(
    "concept_id" = c(1, 2),
    "excluded" = FALSE,
    "descendants" = TRUE,
```

```
    "mapped" = FALSE
  )
)
asthma_cs <- newConceptSetExpression(asthma_cs)
print(asthma_cs)
```

`readSourceTable` *Read a table from the cdm_source and add it to to the cdm.*

Description

Read a table from the cdm_source and add it to to the cdm.

Usage

```
readSourceTable(cdm, name)
```

Arguments

`cdm`	A cdm reference.
`name`	Name of a table to read in the cdm_source space.

Value

A cdm_reference with new table.

`recordCohortAttrition` *Update cohort attrition.*

Description

Update cohort attrition.

Usage

```
recordCohortAttrition(cohort, reason, cohortId = NULL)
```

Arguments

`cohort`	A cohort_table object.
`reason`	A character string.
`cohortId`	Cohort definition id of the cohort to update attrition. If NULL all cohort_definition_id are updated.

Value

cohort_table with updated attrition.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort <- tibble(
  cohort_definition_id = c(1, 1, 1, 2),
  subject_id = 1,
 cohort_start_date = as.Date(c("2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01")),
  cohort_end_date = as.Date(c("2020-01-01", "2021-01-01", "2022-01-01", "2022-01-01")),
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "my_example_cdm",
  cohortTables = list("cohort1" = cohort)
)

cdm$cohort1
attrition(cdm$cohort1)

cdm$cohort1 <- cdm$cohort1 |>
  group_by(cohort_definition_id, subject_id) |>
  filter(cohort_start_date == min(cohort_start_date)) |>
  ungroup() |>
  compute(name = "cohort1", temporary = FALSE) |>
  recordCohortAttrition("Restrict to first observation")

cdm$cohort1
attrition(cdm$cohort1)
```

`resultColumns` *Required columns that the result tables must have.*

Description

Required columns that the result tables must have.

Usage

```
resultColumns(table = "summarised_result")
```

Arguments

`table`	Table to see required columns.

Value

Required columns

Examples

```
library(omopgenerics)

resultColumns()
```

`resultPackageVersion` *Check if different packages version are used for summarise_results object*

Description

Check if different packages version are used for summarise_results object

Usage

```
resultPackageVersion(result)
```

Arguments

`result`	a summarised results object

Value

a summarised results object

`settings` *Get settings from an object.*

Description

Get settings from an object.

Usage

```
settings(x)
```

Arguments

`x`	Object

Value

A table with the settings of the object.

`settings.cohort_table` *Get cohort settings from a cohort_table object.*

Description

Get cohort settings from a cohort_table object.

Usage

```
## S3 method for class 'cohort_table'
settings(x)
```

Arguments

`x`	A cohort_table object.

Value

A table with the details of the cohort settings.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort <- tibble(
  cohort_definition_id = 1,
  subject_id = 1,
  cohort_start_date = as.Date("2010-01-01"),
  cohort_end_date = as.Date("2012-01-01")
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test",
  cohortTables = list("my_cohort" = cohort)
)

settings(cdm$my_cohort)

cdm$my_cohort <- cdm$my_cohort |>
  newCohortTable(cohortSetRef = tibble(
    cohort_definition_id = 1, cohort_name = "new_name"
  ))

settings(cdm$my_cohort)
```

`settings.summarised_result`

Get settings from a summarised_result object.

Description

Get settings from a summarised_result object.

Usage

```
## S3 method for class 'summarised_result'
settings(x)
```

Arguments

`x`	A summarised_result object.

Value

A table with the settings.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cohort <- tibble(
  cohort_definition_id = 1,
  subject_id = 1,
  cohort_start_date = as.Date("2010-01-01"),
  cohort_end_date = as.Date("2012-01-01")
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test",
  cohortTables = list("my_cohort" = cohort)
)

result <- summary(cdm$my_cohort)

settings(result)
```

`settingsColumns` *Identify settings columns of a* `<summarised_result>`

Description

Identifies and returns the columns of the settings table obtained by using `settings()` in a `<summarised_result>` object.

Usage

```
settingsColumns(result, metadata = FALSE)
```

Arguments

`result`	A `<summarised_result>`.
`metadata`	Whether to include metadata columns in settings or not.

Value

Vector with names of the settings columns

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> settingsColumns()
}
```

`sourceType` *Get the source type of an object.*

Description

Get the source type of an object.

Usage

```
sourceType(x)
```

Arguments

`x`	Object to know the source type.

Value

A character vector that defines the type of cdm_source.

`splitAdditional` *Split additional_name and additional_level columns*

Description

Pivots the input dataframe so the values of the column additional_name are transformed into columns that contain values from the additional_level column.

Usage

```
splitAdditional(result, keep = FALSE, fill = "overall")
```

Arguments

`result`	A dataframe with at least the columns additional_name and additional_level.
`keep`	Whether to keep the original group_name and group_level columns.
`fill`	Optionally, a character that specifies what value should be filled in with when missing.

Value

A dataframe.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
```

```
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> splitAdditional()
}
```

`splitAll` *Split all pairs name-level into columns.*

Description

Pivots the input dataframe so any pair name-level columns are transformed into columns (name) that contain values from the corresponding level.

Usage

```
splitAll(result, keep = FALSE, fill = "overall", exclude = "variable")
```

Arguments

`result`	A data.frame.
`keep`	Whether to keep the original name-level columns.
`fill`	A character that specifies what value should be filled in when missing.
`exclude`	Name of a column pair to exclude.

Value

A dataframe with group, strata and additional as columns.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
```

```
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> splitAll()
}
```

`splitGroup` *Split group_name and group_level columns*

Description

Pivots the input dataframe so the values of the column group_name are transformed into columns that contain values from the group_level column.

Usage

```
splitGroup(result, keep = FALSE, fill = "overall")
```

Arguments

`result`	A dataframe with at least the columns group_name and group_level.
`keep`	Whether to keep the original group_name and group_level columns.
`fill`	Optionally, a character that specifies what value should be filled in with when missing.

Value

A dataframe.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
```

```
      "cdm_name" = c("cprd", "eunomia"),
      "group_name" = "cohort_name",
      "group_level" = "my_cohort",
      "strata_name" = "sex",
      "strata_level" = "male",
      "variable_name" = "Age group",
      "variable_level" = "10 to 50",
      "estimate_name" = "count",
      "estimate_type" = "numeric",
      "estimate_value" = "5",
      "additional_name" = "overall",
      "additional_level" = "overall"
    ) |>
      newSummarisedResult(settings = tibble(
        "result_id" = c(1, 2), "custom" = c("A", "B")
      ))

    x

    x |> splitGroup()
  }
```

---

splitStrata *Split strata_name and strata_level columns*

Description

Pivots the input dataframe so the values of the column strata_name are transformed into columns that contain values from the strata_level column.

Usage

```
splitStrata(result, keep = FALSE, fill = "overall")
```

Arguments

`result`	A dataframe with at least the columns strata_name and strata_level.
`keep`	Whether to keep the original group_name and group_level columns.
`fill`	Optionally, a character that specifies what value should be filled in with when missing.

Value

A dataframe.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> splitStrata()
}
```

`statusIndexes` *Status of the indexes*

Description

[Experimental]

Usage

```
statusIndexes(cdm, name = NULL)
```

Arguments

`cdm`	A cdm_reference object.
`name`	Name(s) of the cdm tables.

Value

A tibble with 3 columns: `table_class` class of the table, `table_name` name of the table, `index` index definition, and `index_status` status of the index, either: 'missing', 'extra', 'present'.

---

strataColumns *Identify variables in strata_name column*

Description

Identifies and returns the unique values in strata_name column.

Usage

```
strataColumns(result)
```

Arguments

`result`	A tibble.

Value

Unique values of the strata name column.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))

  x

  x |> strataColumns()
}
```

`summariseLogFile` *Summarise and extract the information of a log file into a* `summarised_result` *object.*

Description

Summarise and extract the information of a log file into a `summarised_result` object.

Usage

```
summariseLogFile(
  logFile = getOption("omopgenerics.logFile"),
  cdmName = "unknown"
)
```

Arguments

`logFile`	File path to the log file to summarise. Create a logFile with `createLogFile()`.
`cdmName`	Name of the cdm for the `summarise_result` object.

Value

A `summarise_result` with the information of the log file.

Examples

```
library(dplyr)

logFile <- tempfile(pattern = "log_{date}_{time}", fileext = ".txt")
createLogFile(logFile = logFile)

logMessage("Starting analysis")
1 + 1
logMessage("Analysis finished")

res <- summariseLogFile()

glimpse(res)

tidy(res)
```

`summary.cdm_reference` *Summary a cdm reference*

Description

Summary a cdm reference

Usage

```
## S3 method for class 'cdm_reference'
summary(object, ...)
```

Arguments

`object`	A cdm reference object.
`...`	For compatibility (not used).

Value

A summarised_result object with a summary of the data contained in the cdm.

Examples

```
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)

summary(cdm)
```

`summary.cdm_source` *Summarise a* `cdm_source` *object*

Description

Summarise a `cdm_source` object

Usage

```
## S3 method for class 'cdm_source'
summary(object, ...)
```

Arguments

`object`	A generated cohort set object.
`...`	For compatibility (not used).

Value

A list of properties of the `cdm_source` object.

Examples

```
summary(newLocalSource())
```

`summary.cohort_table` *Summary a generated cohort set*

Description

Summary a generated cohort set

Usage

```
## S3 method for class 'cohort_table'
summary(object, ...)
```

Arguments

`object`	A generated cohort set object.
`...`	For compatibility (not used).

Value

A summarised_result object with a summary of a cohort_table.

Examples

```
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test",
  cohortTables = list("cohort1" = tibble(
    cohort_definition_id = 1,
    subject_id = 1,
    cohort_start_date = as.Date("2010-01-01"),
    cohort_end_date = as.Date("2010-01-05")
  ))
)

summary(cdm$cohort1)
```

`summary.summarised_result`

Summary a summarised_result

Description

Summary a summarised_result

Usage

```
## S3 method for class 'summarised_result'
summary(object, ...)
```

Arguments

`object`	A summarised_result object.
`...`	For compatibility (not used).

Value

A summary of the result_types contained in a summarised_result object.

Examples

```
library(dplyr, warn.conflicts = FALSE)

person <- tibble(
  person_id = 1, gender_concept_id = 0, year_of_birth = 1990,
  race_concept_id = 0, ethnicity_concept_id = 0
)
observation_period <- tibble(
  observation_period_id = 1, person_id = 1,
  observation_period_start_date = as.Date("2000-01-01"),
  observation_period_end_date = as.Date("2023-12-31"),
  period_type_concept_id = 0
)
cdm <- cdmFromTables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdmName = "test"
)

result <- summary(cdm)

summary(result)
```

`suppress` *Function to suppress counts in result objects*

Description

Function to suppress counts in result objects

Usage

```
suppress(result, minCellCount = 5)
```

Arguments

`result`	Result object
`minCellCount`	Minimum count of records to report results.

Value

Table with suppressed counts

`suppress.summarised_result` *Function to suppress counts in result objects*

Description

Function to suppress counts in result objects

Usage

```
## S3 method for class 'summarised_result'
suppress(result, minCellCount = 5)
```

Arguments

`result`	summarised_result object.
`minCellCount`	Minimum count of records to report results.

Value

summarised_result with suppressed counts.

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(omopgenerics)

my_result <- tibble(
  "result_id" = "1",
  "cdm_name" = "mock",
  "result_type" = "summarised_characteristics",
  "package_name" = "omopgenerics",
  "package_version" = as.character(utils::packageVersion("omopgenerics")),
  "group_name" = "overall",
  "group_level" = "overall",
  "strata_name" = c(rep("overall", 6), rep("sex", 3)),
  "strata_level" = c(rep("overall", 6), "male", "female", "female"),
  "variable_name" = c(
    "number records", "age_group", "age_group",
    "age_group", "age_group", "my_variable", "number records", "age_group",
    "age_group"
  ),
  "variable_level" = c(
    NA, "<50", "<50", ">=50", ">=50", NA, NA,
    "<50", "<50"
  ),
  "estimate_name" = c(
    "count", "count", "percentage", "count", "percentage",
    "random", "count", "count", "percentage"
```

```
    ),
    "estimate_type" = c(
      "integer", "integer", "percentage", "integer",
      "percentage", "numeric", "integer", "integer", "percentage"
    ),
    "estimate_value" = c("10", "5", "50", "3", "30", "1", "3", "12", "6"),
    "additional_name" = "overall",
    "additional_level" = "overall"
  )
my_result <- newSummarisedResult(my_result)
my_result |> glimpse()
my_result <- suppress(my_result, minCellCount = 5)
my_result |> glimpse()
```

`tableName` *Get the table name of a* `cdm_table`.

Description

Get the table name of a `cdm_table`.

Usage

```
tableName(table)
```

Arguments

`table`	A cdm_table.

Value

A character with the name.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
```

```
    )
  ),
  cdmName = "mock"
)

tableName(cdm$person)
```

`tableSource` *Get the table source of a* `cdm_table`.

Description

Get the table source of a `cdm_table`.

Usage

```
tableSource(table)
```

Arguments

`table`	A cdm_table.

Value

A cdm_source object.

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

tableSource(cdm$person)
```

`tidy.summarised_result` *Turn a* `<summarised_result>` *object into a tidy tibble*

Description

[Experimental] Provides tools for obtaining a tidy version of a `<summarised_result>` object. This tidy version will include the settings as columns, `estimate_value` will be pivotted into columns using `estimate_name` as names, and group, strata, and additional will be splitted.

Usage

```
## S3 method for class 'summarised_result'
tidy(x, ...)
```

Arguments

`x`	A `<summarised_result>`.
`...`	For compatibility (not used).

Value

A tibble.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))
```

```
  x

  x |> tidy()
}
```

`tidyColumns` *Identify tidy columns of a* `<summarised_result>`

Description

Identifies and returns the columns that the tidy version of the `<summarised_result>` will have.

Usage

```
tidyColumns(result)
```

Arguments

`result`	A `<summarised_result>`.

Value

Table columns after applying `tidy()` function to a `<summarised_result>`.

Examples

```
{
  library(dplyr)
  library(omopgenerics)

  x <- tibble(
    "result_id" = as.integer(c(1, 2)),
    "cdm_name" = c("cprd", "eunomia"),
    "group_name" = "cohort_name",
    "group_level" = "my_cohort",
    "strata_name" = "sex",
    "strata_level" = "male",
    "variable_name" = "Age group",
    "variable_level" = "10 to 50",
    "estimate_name" = "count",
    "estimate_type" = "numeric",
    "estimate_value" = "5",
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult(settings = tibble(
      "result_id" = c(1, 2), "custom" = c("A", "B")
    ))
```

```
  x

  x |> tidyColumns()
}
```

`tmpPrefix` *Create a temporary prefix for tables, that contains a unique prefix that starts with tmp.*

Description

Create a temporary prefix for tables, that contains a unique prefix that starts with tmp.

Usage

```
tmpPrefix()
```

Value

A temporary prefix.

Examples

```
library(omopgenerics)
tmpPrefix()
```

`toSnakeCase` *Convert a character vector to snake case*

Description

Convert a character vector to snake case

Usage

```
toSnakeCase(x)
```

Arguments

`x`	Character vector to convert

Value

A snake_case vector

Examples

```
toSnakeCase("myVariable")

toSnakeCase(c("cohort1", "Cohort22b"))
```

`transformToSummarisedResult` *Create a <summarised_result> object from a data.frame, given a set of specifications.*

Description

Create a <summarised_result> object from a data.frame, given a set of specifications.

Usage

```
transformToSummarisedResult(
  x,
  group = character(),
  strata = character(),
  additional = character(),
  estimates = character(),
  settings = character()
)
```

Arguments

`x`	A data.frame.
`group`	Columns in x to be used in group_name-group_level formatting.
`strata`	Columns in x to be used in strata_name-strata_level formatting.
`additional`	Columns in x to be used in additional_name-additional_level formatting.
`estimates`	Columns in x to be formatted into: estimate_name-estimate_type-estimate_value.
`settings`	Columns in x thta form the settings of the <summarised_result> object.

Value

A <summarised_result> object.

Examples

```
x <- dplyr::tibble(
  cohort_name = c("cohort1", "cohort2"),
  variable_name = "age",
  mean = c(50, 45.3),
  median = c(55L, 44L)
```

```
)

transformToSummarisedResult(
  x = x,
  group = c("cohort_name"),
  estimates = c("mean", "median")
)
```

`uniqueId`	*Get a unique Identifier with a certain number of characters and a prefix.*

Description

Get a unique Identifier with a certain number of characters and a prefix.

Usage

```
uniqueId(n = 1, exclude = character(), nChar = 3, prefix = "id_")
```

Arguments

`n`	Number of identifiers.
`exclude`	Columns to exclude.
`nChar`	Number of characters.
`prefix`	A prefix for the identifiers.

Value

A character vector with n unique identifiers.

`uniqueTableName`	*Create a unique table name*

Description

Create a unique table name

Usage

```
uniqueTableName(prefix = "")
```

Arguments

`prefix`	Prefix for the table names.

Value

A string that can be used as a dbplyr temp table name

Examples

```
library(omopgenerics)
uniqueTableName()
```

`uniteAdditional` *Unite one or more columns in additional_name-additional_level format*

Description

Unites targeted table columns into additional_name-additional_level columns.

Usage

```
uniteAdditional(
  x,
  cols = character(0),
  keep = FALSE,
  ignore = c(NA, "overall")
)
```

Arguments

`x`	Tibble or dataframe.
`cols`	Columns to aggregate.
`keep`	Whether to keep the original columns.
`ignore`	Level values to ignore.

Value

A tibble with the new columns.

Examples

```
x <- dplyr::tibble(
  variable = "number subjects",
  value = c(10, 15, 40, 78),
  sex = c("Male", "Female", "Male", "Female"),
  age_group = c("<40", ">40", ">40", "<40")
)

x |>
  uniteAdditional(c("sex", "age_group"))
```

`uniteGroup` *Unite one or more columns in group_name-group_level format*

Description

Unites targeted table columns into group_name-group_level columns.

Usage

```
uniteGroup(x, cols = character(0), keep = FALSE, ignore = c(NA, "overall"))
```

Arguments

`x`	Tibble or dataframe.
`cols`	Columns to aggregate.
`keep`	Whether to keep the original columns.
`ignore`	Level values to ignore.

Value

A tibble with the new columns.

Examples

```
x <- dplyr::tibble(
  variable = "number subjects",
  value = c(10, 15, 40, 78),
  sex = c("Male", "Female", "Male", "Female"),
  age_group = c("<40", ">40", ">40", "<40")
)

x |>
  uniteGroup(c("sex", "age_group"))
```

`uniteStrata` *Unite one or more columns in strata_name-strata_level format*

Description

Unites targeted table columns into strata_name-strata_level columns.

Usage

```
uniteStrata(x, cols = character(0), keep = FALSE, ignore = c(NA, "overall"))
```

Arguments

`x`	Tibble or dataframe.
`cols`	Columns to aggregate.
`keep`	Whether to keep the original columns.
`ignore`	Level values to ignore.

Value

A tibble with the new columns.

Examples

```
x <- dplyr::tibble(
  variable = "number subjects",
  value = c(10, 15, 40, 78),
  sex = c("Male", "Female", "Male", "Female"),
  age_group = c("<40", ">40", ">40", "<40")
)

x |>
  uniteStrata(c("sex", "age_group"))
```

`validateAchillesTable` *Validate if a cdm_table is a valid achilles table.*

Description

Validate if a cdm_table is a valid achilles table.

Usage

```
validateAchillesTable(
  table,
  version = NULL,
  cast = FALSE,
  call = parent.frame()
)
```

Arguments

`table`	A cdm_table to validate.
`version`	The cdm vocabulary version.
`cast`	Whether to cast columns to required type.
`call`	Passed to cli call.

Value

invisible achilles table

`validateAgeGroupArgument` *Validate the ageGroup argument. It must be a list of two integerish numbers lower age and upper age, both of the must be greater or equal to 0 and lower age must be lower or equal to the upper age. If not named automatic names will be given in the output list.*

Description

Validate the ageGroup argument. It must be a list of two integerish numbers lower age and upper age, both of the must be greater or equal to 0 and lower age must be lower or equal to the upper age. If not named automatic names will be given in the output list.

Usage

```
validateAgeGroupArgument(
  ageGroup,
  multipleAgeGroup = TRUE,
  overlap = FALSE,
  null = TRUE,
  ageGroupName = "age_group",
  call = parent.frame()
)
```

Arguments

`ageGroup`	age group in a list.
`multipleAgeGroup`	allow mutliple age group.
`overlap`	allow overlapping ageGroup.
`null`	null age group allowed true or false.
`ageGroupName`	Name of the default age group.
`call`	parent frame.

Value

validate ageGroup

Examples

```
validateAgeGroupArgument(list(c(0, 39), c(40, Inf)))
```

`validateCdmArgument` *Validate if an object in a valid cdm_reference.*

Description

Validate if an object in a valid cdm_reference.

Usage

```
validateCdmArgument(
  cdm,
  checkOverlapObservation = FALSE,
  checkStartBeforeEndObservation = FALSE,
  checkPlausibleObservationDates = FALSE,
  checkPerson = FALSE,
  requiredTables = character(),
  validation = "error",
  call = parent.frame()
)
```

Arguments

`cdm`	A cdm_reference object
`checkOverlapObservation`	TRUE to perform check on no overlap observation period
`checkStartBeforeEndObservation`	TRUE to perform check on correct observational start and end date
`checkPlausibleObservationDates`	TRUE to perform check that there are no implausible observation period start dates (before 1800-01-01) or end dates (after the current date)
`checkPerson`	TRUE to perform check on person id in all clinical table are in person table
`requiredTables`	Name of tables that are required to be part of the cdm_reference object.
`validation`	How to perform validation: "error", "warning".
`call`	A call argument to pass to cli functions.

Value

A cdm_reference object

Examples

```
cdm <- cdmFromTables(
  tables = list(
    "person" = dplyr::tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
```

```
    ),
    "observation_period" = dplyr::tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

validateCdmArgument(cdm)
```

`validateCdmTable` *Validate if a table is a valid cdm_table object.*

Description

Validate if a table is a valid cdm_table object.

Usage

```
validateCdmTable(table, name = NULL, call = parent.frame())
```

Arguments

`table`	Object to validate.
`name`	If we want to validate that the table has a specific name.
`call`	Call argument that will be passed to `cli`.

Value

The table or an error message.

`validateCohortArgument` *Validate a cohort table input.*

Description

Validate a cohort table input.

Usage

```
validateCohortArgument(
  cohort,
  checkEndAfterStart = FALSE,
  checkOverlappingEntries = FALSE,
  checkMissingValues = FALSE,
  checkInObservation = FALSE,
  checkAttributes = FALSE,
  checkPermanentTable = FALSE,
  dropExtraColumns = FALSE,
  validation = "error",
  call = parent.frame()
)
```

Arguments

`cohort`	Object to be validated as a valid cohort input.
`checkEndAfterStart`	If TRUE a check that all cohort end dates come on or after cohort start date will be performed.
`checkOverlappingEntries`	If TRUE a check that no individuals have overlapping cohort entries will be performed.
`checkMissingValues`	If TRUE a check that there are no missing values in required fields will be performed.
`checkInObservation`	If TRUE a check that cohort entries are within the individuals observation periods will be performed.
`checkAttributes`	Whether to check if attributes are present and populated correctly.
`checkPermanentTable`	Whether to check if the table has to be a permanent table.
`dropExtraColumns`	Whether to drop extra columns that are not the required ones.
`validation`	How to perform validation: "error", "warning".
`call`	A call argument to pass to cli functions.

Examples

```
cdm <- cdmFromTables(
  tables = list(
    "person" = dplyr::tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = dplyr::tibble(
```

```
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
   ),
   cohortTables = list(
    cohort = dplyr::tibble(
      cohort_definition_id = 1L,
      subject_id = 1L,
      cohort_start_date = as.Date("2020-01-01"),
      cohort_end_date = as.Date("2021-02-10")
    )
  ),
  cdmName = "mock"
)

validateCohortArgument(cdm$cohort)
```

`validateCohortIdArgument`

Validate cohortId argument. CohortId can either be a cohort_definition_id value, a cohort_name or a tidyselect expression referinc to cohort_names. If you want to support tidyselect expressions please use the function as: `validateCohortIdArgument({{cohortId}}, cohort)`.

Description

Validate cohortId argument. CohortId can either be a cohort_definition_id value, a cohort_name or a tidyselect expression referinc to cohort_names. If you want to support tidyselect expressions please use the function as: `validateCohortIdArgument({{cohortId}}, cohort)`.

Usage

```
validateCohortIdArgument(
  cohortId,
  cohort,
  null = TRUE,
  validation = "error",
  call = parent.frame()
)
```

Arguments

`cohortId`	A cohortId vector to be validated.
`cohort`	A cohort_table object.

`null`	Whether `NULL` is accepted. If NULL all `cohortId` will be returned.
`validation`	How to perform validation: "error", "warning".
`call`	A call argument to pass to cli functions.

Examples

```
cdm <- cdmFromTables(
  tables = list(
    "person" = dplyr::tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = dplyr::tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
   ),
   cohortTables = list(
    cohort = dplyr::tibble(
      cohort_definition_id = 1L,
      subject_id = 1L,
      cohort_start_date = as.Date("2020-01-01"),
      cohort_end_date = as.Date("2021-02-10")
    )
  ),
  cdmName = "mock"
)

validateCohortIdArgument(NULL, cdm$cohort)
validateCohortIdArgument(1L, cdm$cohort)
validateCohortIdArgument(2L, cdm$cohort, validation = "warning")
```

`validateColumn` *Validate whether a variable points to a certain exiting column in a table.*

Description

Validate whether a variable points to a certain exiting column in a table.

Usage

```
validateColumn(
  column,
  x,
  type = c("character", "date", "logical", "numeric", "integer"),
```

```
  validation = "error",
  call = parent.frame()
)
```

Arguments

`column`	Name of a column that you want to check that exist in `x` table.
`x`	Table to check if the column exist.
`type`	Type of the column.
`validation`	Whether to throw warning or error.
`call`	Passed to cli functions.

Value

the validated name

Examples

```
x <- dplyr::tibble(a = 1, b = "xxx")

validateColumn("a", x, validation = "warning")
validateColumn("a", x, type = "character", validation = "warning")
validateColumn("a", x, type = "numeric", validation = "warning")
validateColumn("not_existing", x, type = "numeric", validation = "warning")
```

`validateConceptSetArgument` *Validate conceptSet argument. It can either be a list, a codelist, a concept set expression or a codelist with details. The output will always be a codelist.*

Description

Validate conceptSet argument. It can either be a list, a codelist, a concept set expression or a codelist with details. The output will always be a codelist.

Usage

```
validateConceptSetArgument(
  conceptSet,
  cdm = NULL,
  validation = "error",
  call = parent.frame()
)
```

Arguments

`conceptSet`	It can be either a named list of concepts or a codelist, codelist_with_details or concept_set_expression object.
`cdm`	A cdm_reference object, needed if a concept_set_expression is provided.
`validation`	How to perform validation: "error", "warning".
`call`	A call argument to pass to cli functions.

Value

A codelist object.

Examples

```
conceptSet <- list(disease_x = c(1L, 2L))
validateConceptSetArgument(conceptSet)
```

`validateNameArgument` *Validate name argument. It must be a snake_case character vector. You can add the a cdm object to check* `name` *is not already used in that cdm.*

Description

Validate name argument. It must be a snake_case character vector. You can add the a cdm object to check `name` is not already used in that cdm.

Usage

```
validateNameArgument(
  name,
  cdm = NULL,
  validation = "error",
  null = FALSE,
  call = parent.frame()
)
```

Arguments

`name`	Name of a new table to be added to a cdm object.
`cdm`	A cdm_reference object. It will check if a table named name already exists in the cdm.
`validation`	How to perform validation: "error", "warning".
`null`	If TRUE, name can be NULL
`call`	A call argument to pass to cli functions.

Examples

```
# this is a validate name
name <- "my_new_table"
validateNameArgument(name)

# this is not
name <- "myTableNAME"
validateNameArgument(name, validation = "warning")
```

`validateNameLevel` *Validate if two columns are valid Name-Level pair.*

Description

Validate if two columns are valid Name-Level pair.

Usage

```
validateNameLevel(
  x,
  prefix,
  sep = " &&& ",
  validation = "error",
  call = parent.frame()
)
```

Arguments

`x`	A tibble.
`prefix`	Prefix for the name-level pair, e.g. 'strata' for strata_name-strata_level pair.
`sep`	Separation pattern.
`validation`	Either 'error', 'warning' or 'message'.
`call`	Will be used by cli to report errors.

`validateNameStyle` *Validate* `nameStyle` *argument. If any of the element in* `...` *has length greater than 1 it must be contained in nameStyle. Note that snake case notation is used.*

Description

Validate `nameStyle` argument. If any of the element in `...` has length greater than 1 it must be contained in nameStyle. Note that snake case notation is used.

Usage

```
validateNameStyle(nameStyle, ..., call = parent.frame())
```

Arguments

`nameStyle`	A character vector. It must contain all the `...` elements in snake_case format and between `{}`.
`...`	Elements to be included.
`call`	Passed to cli functions.

Value

invisible nameStyle.

Examples

```
validateNameStyle(
  nameStyle = "hi_{cohort_name}",
  cohortName = c("cohort1", "cohort2"),
  otherVariable = c("only 1 value")
)

## Not run:
validateNameStyle(
  nameStyle = "hi_{cohort_name}",
  cohortName = c("cohort1", "cohort2"),
  otherVariable = c("value1", "value2")
)

## End(Not run)
validateNameStyle(
  nameStyle = "{other_variable}_hi_{cohort_name}",
  cohortName = c("cohort1", "cohort2"),
  otherVariable = c("value1", "value2")
)
```

`validateNewColumn` *Validate a new column of a table*

Description

Validate a new column of a table

Usage

```
validateNewColumn(table, column, validation = "warning", call = parent.frame())
```

Arguments

`table`	The table to check if the column already exists.
`column`	Character vector with the name(s) of the new column(s).
`validation`	Whether to throw warning or error.
`call`	Passed to cli functions.

Value

table without conflicting columns.

Examples

```
x <- dplyr::tibble(
  column1 = c(1L, 2L),
  column2 = c("a", "b")
)
validateNewColumn(x, "not_exiting_column")
validateNewColumn(x, "column1")
```

`validateOmopTable` *Validate an omop_table*

Description

Validate an omop_table

Usage

```
validateOmopTable(
  omopTable,
  version = NULL,
  cast = FALSE,
  call = parent.frame()
)
```

Arguments

`omopTable`	An omop_table to check.
`version`	The version of the cdm.
`cast`	Whether to cast columns to the correct type.
`call`	Call argument that will be passed to `cli` error message.

Value

An omop_table object.

`validateResultArgument`

Validate if a an object is a valid 'summarised_result' object.

Description

Validate if a an object is a valid 'summarised_result' object.

Usage

```
validateResultArgument(
  result,
  checkNoDuplicates = FALSE,
  checkNameLevel = FALSE,
  checkSuppression = FALSE,
  validation = "error",
  call = parent.frame()
)
```

Arguments

`result`	summarised_result object to validate.
`checkNoDuplicates`	Whether there are not allowed duplicates in the result object.
`checkNameLevel`	Whether the name-level paired columns are can be correctly split.
`checkSuppression`	Whether the suppression in the result object is well defined.
`validation`	Only error is supported at the moment.
`call`	parent.frame

Value

summarise result object

Examples

```
x <- dplyr::tibble(
  "result_id" = 1L,
  "cdm_name" = "eunomia",
  "group_name" = "cohort_name",
  "group_level" = "my_cohort",
  "strata_name" = c("sex", "sex &&& age_group", "sex &&& year"),
  "strata_level" = c("Female", "Male &&& <40", "Female &&& 2010"),
  "variable_name" = "number subjects",
  "variable_level" = NA_character_,
  "estimate_name" = "count",
  "estimate_type" = "integer",
```

```
    "estimate_value" = c("100", "44", "14"),
    "additional_name" = "overall",
    "additional_level" = "overall"
  ) |>
    newSummarisedResult()

  validateResultArgument(x)
```

---

validateStrataArgument

To validate a strata list. It makes sure that elements are unique and point to columns in table.

Description

To validate a strata list. It makes sure that elements are unique and point to columns in table.

Usage

```
validateStrataArgument(strata, table, call = parent.frame())
```

Arguments

`strata`	A list of characters that point to columns in table.
`table`	A table with columns.
`call`	Passed to cli functions.

Value

The same strata input or an error if the input is incorrect.

Examples

```
strata <- list("age", "sex", c("age", "sex"))
x <- dplyr::tibble(age = 30L, sex = "Female")

validateStrataArgument(strata, x)
```

`validateWindowArgument` *Validate a window argument. It must be a list of two elements (window start and window end), both must be integerish and window start must be lower or equal than window end.*

Description

Validate a window argument. It must be a list of two elements (window start and window end), both must be integerish and window start must be lower or equal than window end.

Usage

```
validateWindowArgument(window, snakeCase = TRUE, call = parent.frame())
```

Arguments

`window`	time window
`snakeCase`	return default window name in snake case if TRUE
`call`	A call argument to pass to cli functions.

Value

time window

Examples

```
validateWindowArgument(list(c(0, 15), c(-Inf, Inf)))
validateWindowArgument(list(c(0, 15), c(-Inf, Inf)), snakeCase = FALSE)
```

`[[.cdm_reference` *Subset a cdm reference object.*

Description

Subset a cdm reference object.

Usage

```
## S3 method for class 'cdm_reference'
x[[name]]
```

Arguments

`x`	A cdm reference
`name`	The name or index of the table to extract from the cdm object.

Value

A single cdm table reference

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdm[["person"]]
```

`[[<-.cdm_reference` *Assign a table to a cdm reference.*

Description

Assign a table to a cdm reference.

Usage

```
## S3 replacement method for class 'cdm_reference'
cdm[[name]] <- value
```

Arguments

`cdm`	A cdm reference.
`name`	Name where to assign the new table.
`value`	Table with the same source than the cdm object.

Value

The cdm reference.

`$.cdm_reference` *Subset a cdm reference object.*

Description

Subset a cdm reference object.

Usage

```
## S3 method for class 'cdm_reference'
x$name
```

Arguments

`x`	A cdm reference.
`name`	The name of the table to extract from the cdm object.

Value

A single cdm table reference

Examples

```
library(omopgenerics)
library(dplyr, warn.conflicts = FALSE)

cdm <- cdmFromTables(
  tables = list(
    "person" = tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdm$person
```

`$<-.cdm_reference` *Assign an table to a cdm reference.*

Description

Assign an table to a cdm reference.

Usage

```
## S3 replacement method for class 'cdm_reference'
cdm$name <- value
```

Arguments

`cdm`	A cdm reference.
`name`	Name where to assign the new table.
`value`	Table with the same source than the cdm object.

Value

The cdm reference.

Examples

```
library(omopgenerics)

cdm <- cdmFromTables(
  tables = list(
    "person" = dplyr::tibble(
      person_id = c(1, 2, 3), gender_concept_id = 0, year_of_birth = 1990,
      race_concept_id = 0, ethnicity_concept_id = 0
    ),
    "observation_period" = dplyr::tibble(
      observation_period_id = 1:3, person_id = 1:3,
      observation_period_start_date = as.Date("2000-01-01"),
      observation_period_end_date = as.Date("2023-12-31"),
      period_type_concept_id = 0
    )
  ),
  cdmName = "mock"
)

cdm$person
```

Index