Package 'CDMConnector'

November 2, 2026

Title Connect to an OMOP Common Data Model

Version 2.8.1

Description Provides tools for working with observational health data in the Observational Medical Outcomes Partnership (OMOP) Common Data Model format with a pipe friendly syntax.
Common data model database table references are stored in a single compound object along with metadata.

License Apache License (>= 2)

URL https://darwin-eu.github.io/CDMConnector/, https://github.com/darwin-eu/CDMConnector

BugReports https://github.com/darwin-eu/CDMConnector/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports httr2, dplyr, dbplyr (>= 2.5.0), DBI (>= 0.3.0), checkmate, cli, purrr, rlang, tidyselect, glue, methods, withr, lifecycle, stringr, stringi, generics, tidyr, jsonlite, readr, omopgenerics (>= 1.4.2)

Suggests SqlRender, CirceR, rJava, covr, knitr, rmarkdown, duckdb, RSQLite, RPostgres, DatabaseConnector, odbc, ggplot2, bigrquery, lubridate, clock, tibble, testthat (>= 3.0.0), pool, snakecase, digest, rstudioapi, uuid

Enhances arrow

Config/testthat/edition 3

Config/testthat/parallel false

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Ger Inberg [aut, cre] (ORCID: <https://orcid.org/0000-0001-8993-8748>), Adam Black [aut] (ORCID: <https://orcid.org/0000-0001-5576-8701>), Artem Gorbachev [aut],

Edward Burn [aut],
Marti Catala Sabate [aut],
Ioanna Nika [aut]

Maintainer Ger Inberg <g.inberg@erasmusmc.nl>

Repository CRAN

Date/Publication 2026-11-02 15:30:02 UTC

Contents

appendPermanent . . . 3
asDate . . . 4
benchmarkCDMConnector . . . 5
cdmCommentContents . . . 5
cdmCon . . . 6
cdmDisconnect.db_cdm . . . 7
cdmFlatten . . . 8
cdmFromCohortSet . . . 9
cdmFromCon . . . 10
cdmSample . . . 13
cdmSubset . . . 14
cdmSubsetCohort . . . 15
cdmTrimVocabulary . . . 16
cdmWriteSchema . . . 17
cohdSimilarConcepts . . . 17
computeDataHashByTable . . . 19
computeQuery . . . 20
copyCdmTo . . . 21
dateadd . . . 21
datediff . . . 22
datepart . . . 23
dbms . . . 24
dbSource . . . 24
downloadEunomiaData . . . 25
dropTable.db_cdm . . . 26
eunomiaDir . . . 26
eunomiaIsAvailable . . . 28
exampleDatasets . . . 28
generateCohortSet . . . 29
generateConceptCohortSet . . . 30
inSchema . . . 32
insertTableSpark . . . 32
listTables . . . 33
readCohortSet . . . 34
requireEunomia . . . 34
snapshot . . . 35
summariseQuantile . . . 35
summariseQuantile2 . . . 36

tblGroup 38
version 39

Index **40**

---

appendPermanent *Run a dplyr query and add the result set to an existing*

Description

Run a dplyr query and add the result set to an existing

Usage

```
appendPermanent(x, name, schema = NULL)
```

Arguments

x	A dplyr query
name	Name of the table to be appended. If it does not already exist it will be created.
schema	Schema where the table exists. Can be a length 1 or 2 vector. (e.g. schema = "my_schema", schema = c("my_schema", "dbo"))

Value

A dplyr reference to the newly created table

Examples

```
## Not run:
library(CDMConnector)

con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomiaDir())
concept <- dplyr::tbl(con, "concept")

# create a table
rxnorm_count <- concept %>%
  dplyr::filter(domain_id == "Drug") %>%
  dplyr::mutate(isRxnorm = (vocabulary_id == "RxNorm")) %>%
  dplyr::count(domain_id, isRxnorm) %>%
  compute("rxnorm_count")

# append to an existing table
rxnorm_count <- concept %>%
  dplyr::filter(domain_id == "Procedure") %>%
  dplyr::mutate(isRxnorm = (vocabulary_id == "RxNorm")) %>%
  dplyr::count(domain_id, isRxnorm) %>%
  appendPermanent("rxnorm_count")

DBI::dbDisconnect(con, shutdown = TRUE)
```

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

---

asDate *as.Date dbplyr translation wrapper*

Description

This is a workaround for using as.Date inside dplyr verbs against a database backend. This function should only be used inside dplyr verbs where the first argument is a database table reference. `asDate` must be unquoted with !! inside dplyr verbs (see example).

Usage

```
asDate(x)
```

Arguments

x	an R expression

Examples

```
## Not run:
con <- DBI::dbConnect(odbc::odbc(), "Oracle")
date_tbl <- dplyr::copy_to(con,
                           data.frame(y = 2000L, m = 10L, d = 10L),
                           name = "tmp",
                           temporary = TRUE)

df <- date_tbl %>%
  dplyr::mutate(date_from_parts = !!asDate(paste0(
    .data$y, "/",
    .data$m, "/",
    .data$d
  ))) %>%
  dplyr::collect()

## End(Not run)
```

benchmarkCDMConnector *Run benchmark of tasks using CDMConnector*

Description

Run benchmark of tasks using CDMConnector

Usage

```
benchmarkCDMConnector(cdm)
```

Arguments

cdm	A CDM reference object

Value

a tibble with time taken for different analyses

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
cdm <- cdmFromCon(con, cdmSchema = "main", writeSchema = "main")
benchmarkCDMConnector(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

cdmCommentContents *Insert Patient CDM Contents as Aligned Comments in RStudio*

Description

This function retrieves the longitudinal event table for one or more persons in a CDM object and inserts it as a nicely formatted, R-style comment block directly into your active RStudio document. This is particularly useful for documenting reproducible test cases or examples by showing relevant CDM contents inline in test scripts or analysis code.

Usage

```
cdmCommentContents(cdm, personIds = NULL)
```

Arguments

cdm	A CDMConnector cdm_reference object.
personIds	Optional numeric vector of person IDs to filter the rows to include. If NULL (default), includes all persons in the cdm.

Details

Each row of patient data will be aligned in columns as a commented table, making it easy to copy, review, and maintain sample data expectations in documentation or test suites.

Requires an interactive RStudio session with the `rstudioapi` package available. The function utilizes `CDMConnector::cdmFlatten()` to extract a longitudinal view, and writes the commented results directly below the cursor in the active RStudio document.

This workflow is especially helpful when documenting expected patient timelines for use in testthat or other test scripts, or when sharing reproducible CDM content for instructional examples.

See Also

cdmFlatten

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
cdm <- cdmFromCon(con, "main", "main")
cdmCommentContents(cdm, personIds = 6)
# person_id | observation_concept_id | start_date | end_date   | type_concept_id...
# 6         | 40213296               | 2006-01-10 | 2006-01-10 | 581452         ...
# 6         | 40213227               | 2006-01-10 | 2006-01-10 | 581452         ...
# 6         | 1118084                | 2005-07-13 | 2005-07-13 | 38000177       ...
# 6         | 80180                  | 2005-07-13 | NA         | 32020          ...
cdmDisconnect(cdm)

## End(Not run)
```

---

cdmCon *Get underlying database connection*

Description

Get underlying database connection

Usage

```
cdmCon(cdm)
```

Arguments

cdm	A cdm reference object created by cdmFromCon

Value

A reference to the database containing tables in the cdm reference

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomiaDir())

cdm <- cdmFromCon(con = con, cdmName = "Eunomia",
                  cdmSchema =  "main", writeSchema = "main")

cdmCon(cdm)

DBI::dbDisconnect(con)

## End(Not run)
```

---

cdmDisconnect.db_cdm *Disconnect the connection of the cdm object*

Description

This function will disconnect from the database as well as drop "temporary" tables that were created on database systems that do not support actual temporary tables. Currently temp tables are emulated on Spark/Databricks systems.

Usage

```
## S3 method for class 'db_cdm'
cdmDisconnect(cdm, dropPrefixTables = FALSE, ...)
```

Arguments

cdm	cdm reference
dropPrefixTables	Whether to drop tables in the writeSchema prefixed with writePrefix
...	Not used. Included for compatibility with generic.

cdmFlatten *Flatten a cdm into a single observation table*

Description

This experimental function transforms the OMOP CDM into a single observation table. This is only recommended for use with a filtered CDM or a cdm that is small in size.

Usage

```
cdmFlatten(
  cdm,
  domain = c("condition_occurrence", "drug_exposure", "procedure_occurrence"),
  includeConceptName = TRUE
)
```

Arguments

cdm	A cdm_reference object
domain	Domains to include. Must be a subset of "condition_occurrence", "drug_exposure", "procedure_occurrence", "measurement", "visit_occurrence", "death", "observation"
includeConceptName	Should concept_name and type_concept_name be include in the output table? TRUE (default) or FALSE

Value

A lazy query that when evaluated will result in a single table

Examples

```
## Not run:
library(CDMConnector)
library(dplyr, warn.conflicts = FALSE)

con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())

cdm <- cdmFromCon(con, cdmSchema = "main")

all_observations <- cdmSubset(cdm, personId = c(2, 18, 42)) %>%
  cdmFlatten() %>%
  collect()

all_observations
#> # A tibble: 213 × 8
#>    person_id observation_.  start_date end_date   type_.  domain obser.  type_.
#>        <dbl>          <dbl> <date>     <date>       <dbl> <chr>  <chr>   <chr>
```

```
#>  1         2       40213201 1986-09-09 1986-09-09  5.81e5 drug   pneumo  <NA>
#>  2        18        4116491 1997-11-09 1998-01-09  3.20e4 condi  Escher  <NA>
#>  3        18       40213227 2017-01-04 2017-01-04  5.81e5 drug   tetanu  <NA>
#>  4        42        4156265 1974-06-13 1974-06-27  3.20e4 condi  Facial  <NA>
#>  5        18       40213160 1966-02-23 1966-02-23  5.81e5 drug   poliov  <NA>
#>  6        42        4198190 1933-10-29 1933-10-29  3.80e7 proce  Append  <NA>
#>  7         2        4109685 1952-07-13 1952-07-27  3.20e4 condi  Lacera  <NA>
#>  8        18       40213260 2017-01-04 2017-01-04  5.81e5 drug   zoster  <NA>
#>  9        42        4151422 1985-02-03 1985-02-03  3.80e7 proce  Sputum  <NA>
#> 10         2        4163872 1993-03-29 1993-03-29  3.80e7 proce  Plain   <NA>
#> # ... with 203 more rows, and abbreviated variable names observation_concept_id,
#> #   type_concept_id, observation_concept_name, type_concept_name

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

`cdmFromCohortSet` *Build a Synthetic CDM from a Cohort Set*

Description

Constructs a synthetic OMOP Common Data Model (CDM) using a set of cohort definitions, created using `CDMConnector::readCohortSet()`. The function generates synthetic data and returns a cdm reference object backed by a DuckDB database, containing synthetic CDM tables and generated cohort table rows.

Usage

```
cdmFromCohortSet(
  cohortSet,
  n = 100,
  cohortTable = "cohort",
  duckdbPath = NULL,
  seed = 1,
  verbose = FALSE,
  ...
)
```

Arguments

`cohortSet`	A data frame (usually from `CDMConnector::readCohortSet()`) with columns `cohort_definition_id`, `cohort_name`, and `cohort` (cohort definition as a list or JSON string).
`n`	Integer. Total number of synthetic persons to generate across all cohorts. Defaults to 100.
`cohortTable`	Character. Name of the cohort table (default `"cohort"`).

duckdbPath	Character or NULL. Path for the final merged DuckDB; if NULL a temporary file is used.
seed	Integer. Base RNG seed; each cohort uses seed + cohort_index for reproducibility (default 1).
verbose	If TRUE, print progress per cohort and per attempt (default FALSE).
...	Arguments passed through to cdmFromJson for each cohort (e.g. targetMatch, successRate, visitConceptId, eventDateJitter, visitDateJitter, demographicVariety, sourceAndTypeVariety, valueVariety). seed is overridden per cohort. targetMatch is per cohort: that fraction of each cohort's generated persons are intended to qualify for that cohort only.

Value

A cdm reference object (as returned by CDMConnector::cdmFromCon()) backed by a DuckDB database. The returned object contains synthetic CDM tables and cohort table rows generated from the specified cohort definitions. The returned cdm has an attribute synthetic_summary (a list with cohort_summaries, cohort_index, n_cohorts, summary (one-line text), any_low_match) for diagnostics and match rates.

Reproducibility

With the same seed, cohortSet, and other arguments, cdmFromCohortSet produces the same synthetic data. Changing seed or n changes the data. The data are random but reproducible.

Examples

```
## Not run:
library(CDMConnector)
cohortSet <- readCohortSet(system.file("cohorts", package = "CDMConnector"))
cdm <- cdmFromCohortSet(cohortSet, n = 100)
cdm$person

## End(Not run)
```

---

cdmFromCon *Create a CDM reference object from a database connection*

Description

Create a CDM reference object from a database connection

Usage

```
cdmFromCon(
  con,
  cdmSchema,
  writeSchema = NULL,
  vocabularySchema = NULL,
  cohortTables = NULL,
  cdmVersion = NULL,
  cdmName = NULL,
  achillesSchema = NULL,
  .softValidation = FALSE,
  writePrefix = NULL
)
```

Arguments

con	A DBI database connection to a database where an OMOP CDM instance is located.
cdmSchema	The schema where the OMOP CDM tables are located. Defaults to NULL.
writeSchema	An optional schema in the CDM database that the user has write access to.
vocabularySchema	An optional schema where the OMOP vocabulary tables are located. If NULL, vocabulary tables are expected to be in cdmSchema.
cohortTables	A character vector listing the cohort table names to be included in the CDM object.
cdmVersion	The version of the OMOP CDM. Can be "5.3", "5.4", "5.5" or NULL (default). If NULL we will attempt to automatically determine the cdm version using the cdm_source table and heuristics.
cdmName	The name of the CDM. If NULL (default) the cdm_source_name . field in the CDM_SOURCE table will be used.
achillesSchema	An optional schema in the CDM database that contains achilles tables.
.softValidation	Normally the observation period table should not have overlapping observation periods for a single person. If .softValidation is TRUE the validation check that looks for overlapping observation periods will be skipped. Other analytic packages may break or produce incorrect results if softValidation is TRUE and the observation period table contains overlapping observation periods.
writePrefix	A prefix that will be added to all tables created in the write_schema. This can be used to create namespace in your database write_schema for your tables.

Details

cdmFromCon creates a new cdm reference object from a DBI database connection. In addition to the connection the user needs to pass in the schema in the database where the cdm data can be found as well as another schema where the user has write access to create tables. Nearly all downstream

analytic packages need the ability to create temporary data in the database so the write_schema is required.

Some database systems have the idea of a catalog or a compound schema with two components. See examples below for how to pass in catalogs and schemas.

You can also specify a `writePrefix`. This is a short character string that will be added to any tables created in the `writeSchema` effectively a namespace in the schema just for your analysis. If the write_schema is a shared between multiple users setting a unique write_prefix ensures you do not overwrite existing tables and allows you to easily clean up tables by dropping all tables that start with the prefix.

Value

A list of dplyr database table references pointing to CDM tables

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())

# minimal example
cdm <- cdmFromCon(con,
                  cdmSchema = "main",
                  writeSchema = "scratch")

# write prefix is optional but recommended if write_schema is shared
cdm <- cdmFromCon(con,
                  cdmSchema = "main",
                  writeSchema = "scratch",
                  writePrefix = "tmp_")

# Some database systems use catalogs or compound schemas.
# These can be specified as follows:
cdm <- cdmFromCon(con,
                  cdmSchema = "catalog.main",
                  writeSchema = "catalog.scratch",
                  writePrefix = "tmp_")

cdm <- cdmFromCon(con,
                  cdmSchema = c("my_catalog", "main"),
                  writeSchema = c("my_catalog", "scratch"),
                  writePrefix = "tmp_")

cdm <- cdmFromCon(con,
                  cdmSchema = c(catalog = "my_catalog", schema = "main"),
                  writeSchema = c(catalog = "my_catalog", schema = "scratch"),
                  writePrefix = "tmp_")

 DBI::dbDisconnect(con)

## End(Not run)
```

cdmSample *Subset a cdm object to a random sample of individuals*

Description

cdmSample takes a cdm object and returns a new cdm that includes only a random sample of persons in the cdm. Only person_ids in both the person table and observation_period table will be considered.

Usage

```
cdmSample(cdm, n, seed = sample.int(1e+06, 1), name = "person_sample")
```

Arguments

cdm	A cdm_reference object.
n	Number of persons to include in the cdm.
seed	Seed for the random number generator.
name	Name of the table that will contain the sample of persons.

Value

A modified cdm_reference object where all clinical tables are lazy queries pointing to subset

Examples

```
## Not run:
library(CDMConnector)
library(dplyr, warn.conflicts = FALSE)

con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())

cdm <- cdmFromCon(con, cdmSchema = "main")

cdmSampled <- cdmSample(cdm, n = 2)

cdmSampled$person %>%
  select(person_id)
#> # Source:   SQL [2 x 1]
#> # Database: DuckDB 0.6.1
#>   person_id
#>       <dbl>
#> 1       155
#> 2      3422
```

```
DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

`cdmSubset` *Subset a cdm object to a set of persons*

Description

`cdmSubset` takes a cdm object and a list of person IDs as input. It returns a new cdm that includes data only for persons matching the provided person IDs. Generated cohorts in the cdm will also be subset to the IDs provided.

Usage

```
cdmSubset(cdm, personId)
```

Arguments

`cdm`	A cdm_reference object
`personId`	A numeric vector of person IDs to include in the cdm

Value

A modified cdm_reference object where all clinical tables are lazy queries pointing to subset

Examples

```
## Not run:
library(CDMConnector)
library(dplyr, warn.conflicts = FALSE)

con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())

cdm <- cdmFromCon(con, cdmSchema = "main")

cdm2 <- cdmSubset(cdm, personId = c(2, 18, 42))

cdm2$person %>%
  select(1:3)
#> # Source:   SQL [3 x 3]
#> # Database: DuckDB 0.6.1
#>   person_id gender_concept_id year_of_birth
#>       <dbl>             <dbl>         <dbl>
#> 1         2              8532          1920
#> 2        18              8532          1965
#> 3        42              8532          1909

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

cdmSubsetCohort *Subset a cdm to the individuals in one or more cohorts*

Description

cdmSubset will return a new cdm object that contains lazy queries pointing to each of the cdm tables but subset to individuals in a generated cohort. Since the cdm tables are lazy queries, the subset operation will only be done when the tables are used. computeQuery can be used to run the SQL used to subset a cdm table and store it as a new table in the database.

Usage

```
cdmSubsetCohort(cdm, cohortTable = "cohort", cohortId = NULL, verbose = FALSE)
```

Arguments

cdm	A cdm_reference object
cohortTable	The name of a cohort table in the cdm reference
cohortId	IDs of the cohorts that we want to subset from the cohort table. If NULL (default) all cohorts in cohort table are considered.
verbose	Should subset messages be printed? TRUE or FALSE (default)

Value

A modified cdm_reference with all clinical tables subset to just the persons in the selected cohorts.

Examples

```
## Not run:
library(CDMConnector)
library(dplyr, warn.conflicts = FALSE)

con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())

cdm <- cdmFromCon(con, cdmSchema = "main", writeSchema = "main")

# generate a cohort
path <- system.file("cohorts2", mustWork = TRUE, package = "CDMConnector")

cohortSet <- readCohortSet(path) %>%
  filter(cohort_name == "GIBleed_male")

# subset cdm to persons in the generated cohort
cdm <- generateCohortSet(cdm, cohortSet = cohortSet, name = "gibleed")

cdmGiBleed <- cdmSubsetCohort(cdm, cohortTable = "gibleed")

cdmGiBleed$person %>%
```

```
  tally()
#> # Source:   SQL [1 x 1]
#> # Database: DuckDB 0.6.1
#>       n
#>   <dbl>
#> 1   237

cdm$person %>%
  tally()
#> # Source:   SQL [1 x 1]
#> # Database: DuckDB 0.6.1
#>       n
#>   <dbl>
#> 1  2694


DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

`cdmTrimVocabulary` *Trim vocabulary tables to the minimum needed for the CDM*

Description

Reduces vocabulary tables to only the concepts referenced in the CDM's clinical data plus their ancestors in the concept_ancestor hierarchy. Tables like concept_synonym and source_to_concept_map are truncated (emptied). This makes the CDM much smaller and faster to upload to a remote test database via `copyCdmTo`.

Usage

```
cdmTrimVocabulary(cdm)
```

Arguments

`cdm`	A cdm_reference object (DuckDB or local)

Details

The trimmed vocabulary is "graph-complete": every concept_id referenced in concept_relationship and concept_ancestor also exists in concept. Most relationships are trimmed out but the ancestor hierarchy is preserved for all concepts in the CDM.

Only works on local (DuckDB or data frame based) CDMs since it is intended for test data preparation.

Value

A cdm_reference with trimmed vocabulary tables

cdmWriteSchema *Get cdm write schema*

Description

Get cdm write schema

Usage

```
cdmWriteSchema(cdm)
```

Arguments

cdm	A cdm reference object created by cdmFromCon

Value

The database write schema

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomiaDir())

cdm <- cdmFromCon(con = con, cdmName = "Eunomia",
                  cdmSchema =  "main", writeSchema = "main")

cdmWriteSchema(cdm)

DBI::dbDisconnect(con)

## End(Not run)
```

cohdSimilarConcepts *Get similar concepts from Columbia Open Health Data (COHD) API*

Description

Queries the COHD API association/relativeFrequency endpoint to return concepts that co-occur with the given concept(s), ranked by relative frequency. Useful for finding clinically related conditions, drugs, or procedures based on EHR prevalence. When given multiple concept IDs, returns concepts that co-occur with the input set, ranked by how many input concepts they co-occur with and by mean relative frequency.

Usage

```
cohdSimilarConcepts(
  conceptId,
  datasetId = 1,
  topN = 50,
  timeoutSec = 30,
  baseUrl = "https://cohd-api.ci.transltr.io/api"
)
```

Arguments

conceptId	Integer or character vector. One or more OMOP concept IDs to find similar concepts for (e.g. conditions, drugs, or procedures).
datasetId	Integer. COHD dataset ID (1 = 5-year, 2 = lifetime; default 1).
topN	Integer. Maximum number of similar concepts to return (default 50). For a single concept, this limits rows by strength; for multiple concepts, this limits the aggregated result.
timeoutSec	Numeric. Request timeout in seconds (default 30).
baseUrl	Character. Base URL of the COHD API (default "https://cohd-api.ci.transltr.io/api").

Value

A data frame with one row per similar concept, or NULL if the API is unavailable or the request fails. When successful:

- **Single concept**: data frame contains concept_id_1, concept_id_2, concept_count_1, concept_count_2, concept_count, relative_frequency, and other_concept_id; rows sorted by relative_frequency descending.
- **Multiple concepts**: data frame contains other_concept_id, n_concepts (how many input concepts co-occur with this one), and mean_rf (mean relative frequency); rows sorted by n_concepts descending then mean_rf descending. If no results or an error occurs, returns NULL and a message is printed.

References

Ta, Casey N.; Dumontier, Michel; Hripcsak, George; P. Tatonetti, Nicholas; Weng, Chunhua (2018). Columbia Open Health Data, a database of EHR prevalence and co-occurrence of conditions, drugs, and procedures. figshare. Collection. doi:10.6084/m9.figshare.c.4151252.v1

Examples

```
## Not run:
# Single concept: top 25 similar to concept 201826 (Type 2 diabetes)
cohdSimilarConcepts(201826, datasetId = 1, topN = 25)

# Multiple concepts: concepts likely to co-occur with this set
cohdSimilarConcepts(c(201826, 316866, 255573), datasetId = 1, topN = 50)
```

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

`computeDataHashByTable` *Compute a hash for each CDM table*

Description

Compute a hash for each CDM table

Usage

```
computeDataHashByTable(cdm)
```

Arguments

`cdm`	A cdm_reference object created by `cdmFromCon`

Details

This function is used to track changes in CDM databases. It returns a dataframe with one hash for each table. The hash is based on the overall row count and the number of unique values of one column of the table. For clinical tables we count the number of unique concept IDs. For some tables we do not calculate any unique value count (e.g. the location table) and simply use the total row count.

‘r lifecycle::badge("experimental")

Value

A dataframe with one row per table, row counts, unique value counts for one column, and a hash

Examples

```
## Not run:
 library(CDMConnector)
 con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
 cdm <- cdmFromCon(con, "main", "main")
 computeDataHashByTable(cdm)
 cdmDisconnect(cdm)

## End(Not run)
```

computeQuery *Execute dplyr query and save result in remote database*

Description

This function is a wrapper around `dplyr::compute` that is tested on several database systems. It is needed to handle edge cases where `dplyr::compute` does not produce correct SQL.

Usage

```
computeQuery(
  x,
  name = uniqueTableName(),
  temporary = TRUE,
  schema = NULL,
  overwrite = TRUE,
  ...
)
```

Arguments

`x`	A dplyr query
`name`	The name of the table to create.
`temporary`	Should the table be temporary: TRUE (default) or FALSE
`schema`	The schema where the table should be created. Ignored if temporary = TRUE.
`overwrite`	Should the table be overwritten if it already exists: TRUE (default) or FALSE Ignored if temporary = TRUE.
`...`	Further arguments passed on the `dplyr::compute`

Value

A `dplyr::tbl()` reference to the newly created table.

Examples

```
## Not run:
library(CDMConnector)

con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomiaDir())
cdm <- cdmFromCon(con, "main")

# create a temporary table in the remote database from a dplyr query
drugCount <- cdm$concept %>%
  dplyr::count(domain_id == "Drug") %>%
  computeQuery()

# create a permanent table in the remote database from a dplyr query
```

```
drugCount <- cdm$concept %>%
  dplyr::count(domain_id == "Drug") %>%
  computeQuery("tmp_table", temporary = FALSE, schema = "main")

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

---

copyCdmTo *Copy a cdm object from one database to another*

Description

It may be helpful to be able to easily copy a small test cdm from a local database to a remote for testing. copyCdmTo takes a cdm object and a connection. It copies the cdm to the remote database connection. CDM tables can be prefixed in the new database allowing for multiple cdms in a single shared database schema.

Usage

```
copyCdmTo(con, cdm, schema, overwrite = FALSE)
```

Arguments

con	A DBI database connection created by DBI::dbConnect
cdm	A cdm reference object created by CDMConnector::cdmFromCon or CDMConnector::cdm_from_con
schema	schema name in the remote database where the user has write permission
overwrite	Should the cohort table be overwritten if it already exists? TRUE or FALSE (default)

Value

A cdm reference object pointing to the newly created cdm in the remote database

---

dateadd *Add days or years to a date in a dplyr query*

Description

This function must be "unquoted" using the "bang bang" operator (!!). See example.

Usage

```
dateadd(date, number, interval = "day")
```

Arguments

date	The name of a date column in the database table as a character string
number	The number of units to add. Can be a positive or negative whole number.
interval	The units to add. Must be either "day" (default) or "year"

Value

Platform specific SQL that can be used in a dplyr query.

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
date_tbl <- dplyr::copy_to(con, data.frame(date1 = as.Date("1999-01-01")),
                           name = "tmpdate", overwrite = TRUE, temporary = TRUE)

df <- date_tbl %>%
  dplyr::mutate(date2 = !!dateadd("date1", 1, interval = "year")) %>%
  dplyr::collect()

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

---

datediff *Compute the difference between two days*

Description

This function must be "unquoted" using the "bang bang" operator (!!). See example.

Usage

```
datediff(start, end, interval = "day")
```

Arguments

start	The name of the start date column in the database as a string.
end	The name of the end date column in the database as a string.
interval	The units to use for difference calculation. Must be either "day" (default) or "year".

Value

Platform specific SQL that can be used in a dplyr query.

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
date_tbl <- dplyr::copy_to(con, data.frame(date1 = as.Date("1999-01-01")),
                           name = "tmpdate", overwrite = TRUE, temporary = TRUE)

df <- date_tbl %>%
  dplyr::mutate(date2 = !!dateadd("date1", 1, interval = "year")) %>%
  dplyr::mutate(dif_years = !!datediff("date1", "date2", interval = "year")) %>%
  dplyr::collect()

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

---

datepart *Extract the day, month or year of a date in a dplyr pipeline*

Description

Extract the day, month or year of a date in a dplyr pipeline

Usage

```
datepart(date, interval = "year", dbms = NULL)
```

Arguments

date	Character string that represents to a date column.
interval	Interval to extract from a date. Valid options are "year", "month", or "day".
dbms	Database system, if NULL it is auto detected.

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), ":memory:")
date_tbl <- dplyr::copy_to(con,
                           data.frame(birth_date = as.Date("1993-04-19")),
                           name = "tmp",
                           temporary = TRUE)
df <- date_tbl %>%
  dplyr::mutate(year = !!datepart("birth_date", "year")) %>%
  dplyr::mutate(month = !!datepart("birth_date", "month")) %>%
  dplyr::mutate(day = !!datepart("birth_date", "day")) %>%
  dplyr::collect()
DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

dbms *Get the database management system (dbms) from a cdm_reference or DBI connection*

Description

Get the database management system (dbms) from a cdm_reference or DBI connection

Usage

```
dbms(con)
```

Arguments

con	A DBI connection or cdm_reference

Value

A character string representing the dbms that can be used with SqlRender

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomiaDir())
cdm <- cdmFromCon(con)
dbms(cdm)
dbms(con)

## End(Not run)
```

dbSource *Create a source for a cdm in a database.*

Description

Create a source for a cdm in a database.

Usage

```
dbSource(con, writeSchema)
```

Arguments

con	Connection to a database.
writeSchema	Schema where cohort tables are. If provided must have read and write access to it. If NULL the cdm will be created without a write_schema.

`downloadEunomiaData` *Download Eunomia data files*

Description

Download the Eunomia data files from https://github.com/darwin-eu/EunomiaDatasets

Usage

```
downloadEunomiaData(
  datasetName = "GiBleed",
  cdmVersion = "5.3",
  pathToData = Sys.getenv("EUNOMIA_DATA_FOLDER"),
  overwrite = FALSE
)
```

Arguments

`datasetName`	The data set name as found on https://github.com/darwin-eu/EunomiaDatasets. The data set name corresponds to the folder with the data set ZIP files
`cdmVersion`	The OMOP CDM version. This version will appear in the suffix of the data file, for example: synpuf_5.3.zip. Must be '5.3' (default) '5.4' or '5.5'
`pathToData`	The path where the Eunomia data is stored on the file system., By default the value of the environment variable "EUNOMIA_DATA_FOLDER" is used.
`overwrite`	Control whether the existing archive file will be overwritten should it already exist.

Value

Invisibly returns the destination if the download was successful.

Examples

```
## Not run:
downloadEunomiaData("GiBleed")

## End(Not run)
```

dropTable.db_cdm *Drop table from a database backed cdm object*

Description

Tables will be dropped from the write schema of the cdm.

Usage

```
## S3 method for class 'db_cdm'
dropTable(cdm, name)
```

Arguments

cdm	a cdm_reference object
name	A character vector of table names to be dropped

eunomiaDir *Create a copy of an example OMOP CDM dataset*

Description

Eunomia is an OHDSI project that provides several example OMOP CDM datasets for testing and development. This function creates a copy of a Eunomia database in duckdb and returns the path to the new database file. If the dataset does not yet exist on the user's computer it will attempt to download the source data to the the path defined by the EUNOMIA_DATA_FOLDER environment variable.

Usage

```
eunomiaDir(
  datasetName = "GiBleed",
  cdmVersion = "5.3",
  databaseFile = tempfile(fileext = ".duckdb")
)
```

Arguments

datasetName	One of "GiBleed" (default), "synthea-allergies-10k", "synthea-anemia-10k", "synthea-breast_cancer-10k", "synthea-contraceptives-10k", "synthea-covid19-10k", "synthea-covid19-200k", "synthea-dermatitis-10k", "synthea-heart-10k", "synthea-hiv-10k", "synthea-lung_cancer-10k", "synthea-medications-10k", "synthea-metabolic_syndrome-10k", "synthea-opioid_addiction-10k", "synthea-rheumatoid_arthritis-10k", "synthea-snf-10k", "synthea-surgery-10k", "synthea-total_joint_replacement-10k", "synthea-veteran_prostate_cancer-10k", "synthea-veterans-10k", "synthea-weight_loss-10k", "empty_cdm", "synpuf-1k", "delphi-100k

cdmVersion	The OMOP CDM version. Must be "5.3", "5.4" or "5.5"
databaseFile	The full path to the new copy of the example CDM dataset.

Details

Most of the Eunomia datasets available in CDMConnector are from the Synthea project. Synthea is an open-source synthetic patient generator that models the medical history of synthetic patients. The Synthea datasets are generated using the Synthea tool and then converted to the OMOP CDM format using the OHDSI ETL-Synthea project https://ohdsi.github.io/ETL-Synthea/. Currently the synthea datasets are only available in the OMOP CDM v5.3 format. See https://syntheticahealth.github.io/synthea/ for details on the Synthea project.

In addition to Synthea, the Eunomia project provides the CMS Synthetic Public Use Files (SynPUFs) in both 5.3, 5.4 and 5.5 OMOP CDM formats. This data is synthetic US Medicare claims data mapped to OMOP CDM format. The OMOP CDM has a set of optional metadata tables, called Achilles tables, that include pre-computed analytics about the entire dataset such as record and person counts. The Eunomia Synpuf datasets include the Achilles tables.

Eunomia also provides empty cdms that can be used as a starting point for creating a new example CDM. This is useful for creating test data for studies or analytic packages. The empty CDM includes the vocabulary tables and all OMOP CDM tables but the clinical tables are empty and need to be populated with data. For additional information on creating small test CDM datasets see https://ohdsi.github.io/omock/ and https://darwin-eu.github.io/TestGenerator/.

To contribute synthetic observational health data to the Eunomia project please open an issue at https://github.com/OHDSI/Eunomia/issues/

Setup: To use the eunomiaDir function please set the EUNOMIA_DATA_FOLDER in your .Renviron file to a folder on your computer where the datasets will be downloaded to. This file can be opened by calling usethis::edit_r_environ().

Value

The file path to the new Eunomia dataset copy

Examples

```
## Not run:

 # The defaults GiBleed dataset is a small dataset that is useful for testing
 library(CDMConnector)
 con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
 cdm <- cdmFromCon(con, "main", "main")
 cdmDisconnect(cdm)

 # Synpuf datasets include the Achilles tables
 con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir("synpuf-1k", "5.3"))
 cdm <- cdmFromCon(con, "main", "main", achillesSchema = "main")
 cdmDisconnect(cdm)

 # Currently the only 5.4 dataset is synpuf-1k
 con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir("synpuf-1k", "5.4"))
 cdm <- cdmFromCon(con, "main", "main", achillesSchema = "main")
```

```
cdmDisconnect(cdm)

## End(Not run)
```

`eunomiaIsAvailable` *Has the Eunomia dataset been cached?*

Description

Has the Eunomia dataset been cached?

Usage

```
eunomiaIsAvailable(datasetName = "GiBleed", cdmVersion = "5.3")
```

Arguments

`datasetName`	Name of the Eunomia dataset to check. Defaults to "GiBleed".
`cdmVersion`	Version of the Eunomia dataset to check. Must be "5.3", "5.4" or "5.5"

Value

TRUE if the eunomia example dataset is available and FALSE otherwise

---

`exampleDatasets` *List the available example CDM datasets*

Description

List the available example CDM datasets

Usage

```
exampleDatasets()
```

Value

A character vector with example CDM dataset identifiers

Examples

```
## Not run:
library(CDMConnector)
exampleDatasets()[1]
#> [1] "GiBleed"

con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir("GiBleed"))
cdm <- cdmFromCon(con)

## End(Not run)
```

`generateCohortSet` *Generate a cohort set on a cdm object*

Description

A "cohort_table" object consists of four components

- A remote table reference to an OHDSI cohort table with at least the columns: cohort_definition_id, subject_id, cohort_start_date, cohort_end_date. Additional columns are optional and some analytic packages define additional columns specific to certain analytic cohorts.
- A **settings attribute** which points to a remote table containing cohort settings including the names of the cohorts.
- An **attrition attribute** which points to a remote table with attrition information recorded during generation. This attribute is optional. Since calculating attrition takes additional compute it can be skipped resulting in a NULL attrition attribute.
- A **cohortCounts attribute** which points to a remote table containing cohort counts

Usage

```
generateCohortSet(
  cdm,
  cohortSet,
  name,
  computeAttrition = TRUE,
  overwrite = TRUE
)
```

Arguments

`cdm`	A cdm reference created by CDMConnector. write_schema must be specified.
`cohortSet`	A cohortSet dataframe created with `readCohortSet()`
`name`	Name of the cohort table to be created. This will also be used as a prefix for the cohort attribute tables. This must be a lowercase character string that starts with a letter and only contains letters, numbers, and underscores.

computeAttrition
: Should attrition be computed? TRUE (default) or FALSE

overwrite
: Should the cohort table be overwritten if it already exists? TRUE (default) or FALSE

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
cdm <- cdmFromCon(con,
                  cdmSchema = "main",
                  writeSchema = "main")

cohortSet <- readCohortSet(system.file("cohorts2", package = "CDMConnector"))
cdm <- generateCohortSet(cdm, cohortSet, name = "cohort")

print(cdm$cohort)

attrition(cdm$cohort)
settings(cdm$cohort)
cohortCount(cdm$cohort)

## End(Not run)
```

---

generateConceptCohortSet

Create a new generated cohort set from a list of concept sets

---

Description

Generate a new cohort set from one or more concept sets. Each concept set will result in one cohort and represent the time during which the concept was observed for each subject/person. Concept sets can be passed to this function as:

- A named list of numeric vectors, one vector per concept set
- A named list of Capr concept sets

Clinical observation records will be looked up in the respective domain tables using the vocabulary in the CDM. If a required domain table does not exist in the cdm object a warning will be given. Concepts that are not in the vocabulary or in the data will be silently ignored. If end dates are missing or do not exist, as in the case of the procedure and observation domains, the the start date will be used as the end date.

Usage

```
generateConceptCohortSet(
  cdm,
  conceptSet = NULL,
  name,
  limit = "first",
  requiredObservation = c(0, 0),
  end = "observation_period_end_date",
  subsetCohort = NULL,
  subsetCohortId = NULL,
  overwrite = TRUE
)
```

Arguments

cdm	A cdm reference object created by CDMConnector::cdmFromCon or CDMConnector::cdm_from_con
conceptSet	A named list of numeric vectors or a Concept Set Expression created omopgenerics::newConceptSetExp
name	The name of the new generated cohort table as a character string
limit	Include "first" (default) or "all" occurrences of events in the cohort • "first" will include only the first occurrence of any event in the concept set in the cohort. • "all" will include all occurrences of the events defined by the concept set in the cohort.
requiredObservation	A numeric vector of length 2 that specifies the number of days of required observation time prior to index and post index for an event to be included in the cohort.
end	How should the cohort_end_date be defined? • "observation_period_end_date" (default): The earliest observation_period_end_date after the event start date • numeric scalar: A fixed number of days from the event start date • "event_end_date": The event end date. If the event end date is not populated then the event start date will be used
subsetCohort	A cohort table containing the individuals for which to generate cohorts for. Only individuals in the cohort table will appear in the created generated cohort set.
subsetCohortId	A set of cohort IDs from the cohort table for which to include. If none are provided, all cohorts in the cohort table will be included.
overwrite	Should the cohort table be overwritten if it already exists? TRUE (default) or FALSE.

Value

A cdm reference object with the new generated cohort set table added

inSchema *Helper for working with compound schema*

Description

Helper for working with compound schema

Usage

```
inSchema(schema, table, dbms = NULL)
```

Arguments

schema	A schema name as a character string
table	A table name as character string
dbms	The name of the database management system as returned by `dbms(connection)`

Value

A DBI::Id that represents a qualified table and schema

insertTableSpark *Fast bulk insert of a local table on Spark / Databricks*

Description

Uploads a local data frame to Spark/Databricks using multi-row `INSERT INTO ... VALUES (...), (...), ...` statements. This is the same mechanism `insertTable()` now uses by default on Spark, so you only need `insertTableSpark()` directly when you want to tune `batchSize`. Multi-row VALUES inserts are dramatically faster than the `INSERT ... SELECT ... UNION ALL` approach Spark's planner struggles with (benchmarked ~50x faster at 1000 rows).

Usage

```
insertTableSpark(cdm, name, table, overwrite = TRUE, batchSize = 5000L)
```

Arguments

cdm	A `cdm_reference` or `db_cdm` source object backed by a Spark/Databricks connection. Must have a `writeSchema`.
name	Name of the destination table (single character).
table	A local data frame to upload.
overwrite	If `TRUE` (default), drop the table first if it exists.
batchSize	Number of rows per `INSERT` statement. Default 5000. Larger batches reduce round trips but Spark imposes a query-size limit (~16MB) — reduce if you hit "query too large" errors with very wide tables.

Details

Only intended for Spark connections. For other dialects use `insertTable()`.

Value

A `cdm_table` referencing the newly inserted table.

`listTables` *List tables in a schema*

Description

DBI::dbListTables can be used to get all tables in a database but not always in a specific schema. `listTables` will list tables in a schema.

Usage

```
listTables(con, schema = NULL)
```

Arguments

`con`	A DBI connection to a database
`schema`	The name of a schema in a database. If NULL, returns DBI::dbListTables(con).

Value

A character vector of table names

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomiaDir())
listTables(con, schema = "main")

## End(Not run)
```

`readCohortSet` *Read a set of cohort definitions into R*

Description

A "cohort set" is a collection of cohort definitions. In R this is stored in a dataframe with cohort_definition_id, cohort_name, and cohort columns. On disk this is stored as a folder with a CohortsToCreate.csv file and one or more json files, or as a single .json file. If the CohortsToCreate.csv file is missing then all of the json files in the folder will be used, cohort_definition_id will be automatically assigned in alphabetical order, and cohort_name will match the file names. You may also pass the path to a single .json file to read one cohort.

Usage

```
readCohortSet(path)
```

Arguments

`path`	The path to a folder containing Circe cohort definition json files (and optionally a csv file named CohortsToCreate.csv with columns cohortId, cohortName, and jsonPath), or the path to a single .json file.

`requireEunomia` *Require eunomia to be available. The function makes sure that you can later create a eunomia database with* `eunomiaDir()`.

Description

Require eunomia to be available. The function makes sure that you can later create a eunomia database with `eunomiaDir()`.

Usage

```
requireEunomia(datasetName = "GiBleed", cdmVersion = "5.3")
```

Arguments

`datasetName`	Name of the Eunomia dataset to check. Defaults to "GiBleed".
`cdmVersion`	Version of the Eunomia dataset to check. Must be "5.3", "5.4" or "5.5".

Value

Path to eunomia database.

snapshot *Extract CDM metadata*

Description

Extract the name, version, and selected record counts from a cdm.

Usage

```
snapshot(cdm, computeDataHash = FALSE)
```

Arguments

cdm	A cdm object
computeDataHash	Compute a hash of the CDM. See ?DatabaseConnector::computeDataHash for details.

Value

A named list of attributes about the cdm including selected fields from the cdm_source table and record counts from the person and observation_period tables

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
cdm <- cdmFromCon(con, "main")
snapshot(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

summariseQuantile *Quantile calculation using dbplyr*

Description

This function provides DBMS independent syntax for quantiles estimation. Can be used by itself or in combination with `mutate()` when calculating other aggregate metrics (min, max, mean).

`summarise_quantile()`, `summarize_quantile()`, `summariseQuantile()` and `summarizeQuantile()` are synonyms.

Usage

```
summariseQuantile(.data, x = NULL, probs, nameSuffix = "value")
```

Arguments

.data	lazy data frame backed by a database query.
x	column name whose sample quantiles are wanted.
probs	numeric vector of probabilities with values in [0,1].
nameSuffix	character; is appended to numerical quantile value as a column name part.

Details

Implemented quantiles estimation algorithm returns values analogous to quantile{stats} with argument type = 1. See discussion in Hyndman and Fan (1996). Results differ from PERCENTILE_CONT natively implemented in various DBMS, where returned values are equal to quantile{stats} with default argument type = 7

Value

An object of the same type as '.data'

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
mtcars_tbl <- dplyr::copy_to(con, mtcars, name = "tmp", overwrite = TRUE, temporary = TRUE)

df <- mtcars_tbl %>%
 dplyr::group_by(cyl) %>%
 dplyr::mutate(mean = mean(mpg, na.rm = TRUE)) %>%
 summariseQuantile(mpg, probs = c(0, 0.2, 0.4, 0.6, 0.8, 1),
                   nameSuffix = "quant") %>%
 dplyr::collect()

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

---

summariseQuantile2 *Quantile calculation using dbplyr*

Description

This function provides DBMS independent syntax for quantile estimation. Some database systems do not have a quantile function. The SQL generated by summarizeQuantile2 should work on all supported database systems. This function can be added to a dplyr pipeline and adds an additional query to the input. No computation is triggered by summarizeQuantile2 if the input is a tbl reference to a database table.

Usage

```
summariseQuantile2(.data, x, probs, nameSuffix = "{x}")
```

Arguments

.data	lazy data frame backed by a database query created by dplyr::tbl().
x	A string vector of column names whose sample quantiles are wanted.
probs	A numeric vector of probabilities with values in [0,1].
nameSuffix	A single character character string, evaluated by glue::glue() that is appended to numerical quantile value as a column name part.

Details

Implemented quantiles estimation algorithm returns values analogous to quantile{stats} with argument type = 1. See discussion in Hyndman and Fan (1996). Results differ from PERCENTILE_CONT natively implemented in various DBMS, where returned values are equal to quantile{stats} with default argument type = 7

[Experimental]

Value

A lazy query with quantile calculation added. The result (after computation) will have one row per combination of grouping variables and one column for every variable/quantile combination. (see examples)

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
mtcars_tbl <- dplyr::copy_to(con, mtcars, name = "tmp", overwrite = TRUE, temporary = TRUE)

# quantiles for a single column
mtcars_tbl %>%
  dplyr::group_by(cyl) %>%
  dplyr::mutate(mean = mean(mpg, na.rm = TRUE)) %>%
 summariseQuantile2("mpg", probs = c(0, 0.2, 0.4, 0.6, 0.8, 1),  nameSuffix = "quant") %>%
  dplyr::collect()

#> cyl  p0_quant p20_quant p40_quant p60_quant p80_quant p100_quant
#>   6      17.8      18.1      19.2      21        21         21.4
#>   8      10.4      13.3      15        15.5      17.3       19.2
#>   4      21.4      22.8      24.4      27.3      30.4       33.9

# multiple columns
mtcars_tbl %>%
  dplyr::group_by(cyl) %>%
  dplyr::mutate(mean = mean(mpg, na.rm = TRUE)) %>%
 summariseQuantile2(c("mpg", "hp", "wt"), probs = c(0.2, 0.8),  nameSuffix = "{x}_quant") %>%
  dplyr::collect()
```

```
#>  cyl p20_mpg_quant p80_mpg_quant p20_hp_quant p80_hp_quant p20_wt_quant p80_wt_quant
#>    4          22.8          30.4           65           97         1.84         2.78
#>    6          18.1          21            110          123         2.77         3.44
#>    8          13.3          17.3          175          245         3.44         5.25

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

`tblGroup` *CDM table selection helper*

Description

The OMOP CDM tables are grouped together and the `tblGroup` function allows users to easily create a CDM reference including one or more table groups.

Usage

```
tblGroup(group, cdmVersion = "5.3")
```

Arguments

`group`	A character vector of CDM table groups: "vocab", "clinical", "all", "default", "derived".
`cdmVersion`	The OMOP CDM version whose table groups should be returned. Defaults to "5.3" for backwards compatibility.

Details

The "default" table group is meant to capture the most commonly used set of CDM tables. Currently the "default" group is: person, observation_period, visit_occurrence, visit_detail, condition_occurrence, drug_exposure, procedure_occurrence, device_exposure, measurement, observation, death, note, note_nlp, specimen, fact_relationship, location, care_site, provider, payer_plan_period, cost, drug_era, dose_era, condition_era, concept, vocabulary, concept_relationship, concept_ancestor, concept_synonym, drug_strength

Value

A character vector of CDM tables names in the groups

Examples

```
## Not run:
con <- DBI::dbConnect(RPostgres::Postgres(),
                      dbname = "cdm",
                      host = "localhost",
                      user = "postgres",
                      password = Sys.getenv("PASSWORD"))

cdm <- cdmFromCon(con, cdmName = "test", cdmSchema = "public") %>%
  cdmSelectTbl(tblGroup("vocab"))

## End(Not run)
```

`version` *Get the CDM version*

Description

Extract the CDM version attribute from a cdm_reference object

Usage

```
version(cdm)
```

Arguments

`cdm`	A cdm object

Value

"5.3", "5.4" or "5.5"

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir())
cdm <- cdmFromCon(con, cdmSchema = "main", writeSchema = "main")
version(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

Index

appendPermanent, 3
asDate, 4

benchmarkCDMConnector, 5

cdmCommentContents, 5
cdmCon, 6
cdmDisconnect.db_cdm, 7
cdmFlatten, *6*, 8
cdmFromCohortSet, 9
cdmFromCon, 10
cdmSample, 13
cdmSubset, 14
cdmSubsetCohort, 15
cdmTrimVocabulary, 16
cdmWriteSchema, 17
cohdSimilarConcepts, 17
computeDataHashByTable, 19
computeQuery, 20
copyCdmTo, *16*, 21

dateadd, 21
datediff, 22
datepart, 23
dbms, 24
dbSource, 24
downloadEunomiaData, 25
dropTable.db_cdm, 26

eunomiaDir, 26
eunomiaIsAvailable, 28
exampleDatasets, 28

generateCohortSet, 29
generateConceptCohortSet, 30

inSchema, 32
insertTable(), *32, 33*
insertTableSpark, 32

listTables, 33

readCohortSet, 34
requireEunomia, 34

snapshot, 35
summariseQuantile, 35
summariseQuantile2, 36

tblGroup, 38

version, 39