Package 'CDMConnector'

November 13, 2024

Title Connect to an OMOP Common Data Model

Version 1.6.0

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

RoxygenNote 7.3.2

Depends R (>= 4.0)

Imports dplyr, dbplyr (>= 2.5.0), DBI (>= 0.3.0), checkmate, cli, purrr, rlang, tidyselect, readr, glue, waldo, methods, withr, lifecycle, jsonlite, stringr, stringi, fs, generics, tidyr, omopgenerics (>= 0.1.2)

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

Enhances arrow

Config/testthat/edition 3

Config/testthat/parallel false

VignetteBuilder knitr

Collate 'CDMConnector-package.R' 'Eunomia.R' 'benchmarkCDMConnector.R' 'cdm.R' 'cdmSubset.R' 'cdm_from_environment.R' 'cohortTransformations.R' 'cohort_ddl.R' 'compute.R' 'copy_cdm_to.R' 'dateadd.R' 'dbSource.R'

'reexports-omopgenerics.R' 'generateCohortSet.R'
'generateConceptCohortSet.R' 'summariseQuantile.R' 'utils.R'
'validate.R' 'zzz-deprecated.R'

NeedsCompilation no

Author Adam Black [aut, cre] (<https://orcid.org/0000-0001-5576-8701>),
Artem Gorbachev [aut],
Edward Burn [aut],
Marti Catala Sabate [aut]

Maintainer Adam Black <black@ohdsi.org>

Repository CRAN

Date/Publication 2024-11-13 10:00:02 UTC

Contents

`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)

append_permanent(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 = eunomia_dir())
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)

as_date(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)
```

`assert_tables` *Assert that tables exist in a cdm object*

Description

A cdm object is a list of references to a subset of tables in the OMOP Common Data Model. If you write a function that accepts a cdm object as a parameter `assert_tables`/`assertTables` will help you check that the tables you need are in the cdm object, have the correct columns/fields, and (optionally) are not empty.

Usage

```
assert_tables(cdm, tables, empty.ok = FALSE, add = NULL)

assertTables(cdm, tables, empty.ok = FALSE, add = NULL)
```

Arguments

`cdm`	A cdm object
`tables`	A character vector of table names to check.
`empty.ok`	Should an empty table (0 rows) be considered an error? TRUE or FALSE (default)
`add`	An optional AssertCollection created by `checkmate::makeAssertCollection()` that errors should be added to.

Value

Invisibly returns the cdm object

Examples

```
## Not run:
# Use assertTables inside a function to check that tables exist
countDrugsByGender <- function(cdm) {
  assertTables(cdm, tables = c("person", "drug_era"), empty.ok = FALSE)

  cdm$person %>%
    dplyr::inner_join(cdm$drug_era, by = "person_id") %>%
    dplyr::count(.data$gender_concept_id, .data$drug_concept_id) %>%
    dplyr::collect()
}

library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomia_dir())
cdm <- cdm_from_con(con)

countDrugsByGender(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)


## End(Not run)
```

---

assert_write_schema *Assert that cdm has a writable schema*

Description

A cdm object can optionally contain a single schema in a database with write access. assert_write_schema checks that the cdm contains the "write_schema" attribute and tests that local dataframes can be written to tables in this schema.

Usage

```
assert_write_schema(cdm, add = NULL)

assertWriteSchema(cdm, add = NULL)
```

Arguments

cdm	A cdm object
add	An optional AssertCollection created by `checkmate::makeAssertCollection()` that errors should be added to.

Value

Invisibly returns the cdm object

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(), eunomia_dir())
cdm <- cdm_from_con(con, cdm_schema = "main", write_schema = "main")
benchmarkCDMConnector(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

cdmCon *Get underlying database connection*

Description

Get underlying database connection

Usage

```
cdmCon(cdm)
```

Arguments

cdm	A cdm reference object created by cdm_from_con

Value

A reference to the database containing tables in the cdm reference

Examples

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

cdm <- cdm_from_con(con = con, cdm_name = "Eunomia",
                    cdm_schema =  "main", write_schema = "main")

cdmCon(cdm)

DBI::dbDisconnect(con)

## End(Not run)
```

---

cdmDisconnect *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

```
cdmDisconnect(cdm)

cdm_disconnect(cdm)
```

Arguments

cdm	cdm reference

---

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", "drug", "procedure"),
  includeConceptName = TRUE
)

cdm_flatten(
  cdm,
  domain = c("condition", "drug", "procedure"),
  include_concept_name = TRUE
)
```

Arguments

cdm	A cdm_reference object
domain	Domains to include. Must be a subset of "condition", "drug", "procedure", "measurement", "visit", "death", "observation".
include_concept_name, includeConceptName	Should concept_name and type_concept_name be include in the output table? TRUE (default) or FALSE

Details

[Experimental]

Value

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

Examples

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

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

cdm <- cdm_from_con(con, cdm_schema = "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)
```

---

cdmName *Get the CDM name*

Description

Extract the CDM name attribute from a cdm_reference object

Usage

```
cdmName(cdm)

cdm_name(cdm)
```

Arguments

cdm	A cdm object

Value

The name of the CDM as a character string

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con, cdm_schema = "main", write_schema = "main")
cdmName(cdm)
#> [1] "eunomia"

DBI::dbDisconnect(con, shutdown = TRUE)

## 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")

cdm_sample(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.

Details

[Experimental]

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(), eunomia_dir())

cdm <- cdm_from_con(con, cdm_schema = "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)

cdm_subset(cdm, person_id)
```

Arguments

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

Details

[Experimental]

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(), eunomia_dir())

cdm <- cdm_from_con(con, cdm_schema = "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)

cdm_subset_cohort(
  cdm,
  cohort_table = "cohort",
  cohort_id = NULL,
  verbose = FALSE
)
```

Arguments

`cdm`	A cdm_reference object
`cohortTable`, `cohort_table`	The name of a cohort table in the cdm reference
`cohortId`, `cohort_id`	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)

Details

[Experimental]

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(), eunomia_dir())

cdm <- cdm_from_con(con, cdm_schema = "main", write_schema = "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)
```

`cdmWriteSchema` *Get cdm write schema*

Description

Get cdm write schema

Usage

```
cdmWriteSchema(cdm)
```

Arguments

cdm	A cdm reference object created by cdm_from_con

Value

The database write schema

Examples

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

cdm <- cdm_from_con(con = con, cdm_name = "Eunomia",
                    cdm_schema =  "main", write_schema = "main")

cdmWriteSchema(cdm)

DBI::dbDisconnect(con)

## End(Not run)
```

---

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

Description

Create a CDM reference object from a database connection

Usage

```
cdm_from_con(
  con,
  cdm_schema,
  write_schema,
  cohort_tables = NULL,
  cdm_version = "5.3",
  cdm_name = NULL,
  achilles_schema = NULL,
  .soft_validation = FALSE,
  write_prefix = NULL
)

cdmFromCon(
  con,
```

```
  cdmSchema,
  writeSchema,
  cohortTables = NULL,
  cdmVersion = "5.3",
  cdmName = NULL,
  achillesSchema = NULL,
  .softValidation = FALSE,
  writePrefix = NULL
)
```

Arguments

con	A DBI database connection to a database where an OMOP CDM v5.4 or v5.3 instance is located.
cdm_schema, cdmSchema	The schema where the OMOP CDM tables are located. Defaults to NULL.
write_schema, writeSchema	An optional schema in the CDM database that the user has write access to.
cohort_tables, cohortTables	A character vector listing the cohort table names to be included in the CDM object.
cdm_version, cdmVersion	The version of the OMOP CDM: "5.3" (default), "5.4", "auto". "auto" attempts to automatically determine the cdm version using heuristics. Cohort tables must be in the write_schema.
cdm_name, cdmName	The name of the CDM. If NULL (default) the cdm_source_name . field in the CDM_SOURCE table will be used.
achilles_schema, achillesSchema	An optional schema in the CDM database that contains achilles tables.
.soft_validation, .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.
write_prefix, 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

cdm_from_con / 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 `write_prefix`. This is a short character string that will be added to any tables created in the `write_schema` 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(), eunomia_dir())

# minimal example
cdm <- cdm_from_con(con,
                    cdm_schema = "main",
                    write_schema = "scratch")

# write prefix is optional but recommended if write_schema is shared
cdm <- cdm_from_con(con,
                    cdm_schema = "main",
                    write_schema = "scratch",
                    write_prefix = "tmp_")

# Some database systems use catalogs or compound schemas.
# These can be specified as follows:
cdm <- cdm_from_con(con,
                    cdm_schema = "catalog.main",
                    write_schema = "catalog.scratch",
                    write_prefix = "tmp_")

cdm <- cdm_from_con(con,
                    cdm_schema = c("my_catalog", "main"),
                    write_schema = c("my_catalog", "scratch"),
                    write_prefix = "tmp_")

cdm <- cdm_from_con(con,
                    cdm_schema = c(catalog = "my_catalog", schema = "main"),
                    write_schema = c(catalog = "my_catalog", schema = "scratch"),
                    write_prefix = "tmp_")

 DBI::dbDisconnect(con)

## End(Not run)
```

`cdm_from_environment` *Create a CDM object from a pre-defined set of environment variables*

Description

This function is intended to be used with the Darwin execution engine. The execution engine runs OHDSI studies in a pre-defined runtime environment and makes several environment variables available for connecting to a CDM database. Programmer writing code to run on the execution engine and simply use `cdm <- cdm_from_environment()` to create a cdm reference object to use for their analysis and the database connection and cdm object should be automatically created. This obviates the need for site specific code for connecting to the database and creating the cdm reference object.

Usage

```
cdm_from_environment(write_prefix = "")
```

Arguments

`write_prefix` (string) An optional prefix to use for all tables written to the CDM.

Details

The environment variables used by this function and provided by the execution engine are listed below.

- DBMS_TYPE: one of "postgresql", "sql server", "redshift", "duckdb", "snowflake".
- DATA_SOURCE_NAME: a free text name for the CDM given by the person running the study.
- CDM_VERSION: one of "5.3", "5.4".
- DBMS_CATALOG: The database catalog. Important primarily for compound schema names used in SQL Server and Snowflake.
- DBMS_SERVER: The database server URL.
- DBMS_NAME: The database name used for creating the connection.
- DBMS_PORT: The database port number.
- DBMS_USERNAME: The database username needed to authenticate.
- DBMS_PASSWORD: The database password needed to authenticate.
- CDM_SCHEMA: The schema name where the OMOP CDM is located in the database.
- WRITE_SCHEMA: The shema where the user has write access and tables will be created during study execution.

Value

A cdm_reference object

Examples

```
## Not run:

library(CDMConnector)

# This will only work in an evironment where the proper variables are present.
cdm <- cdm_from_environment()

# Proceed with analysis using the cdm object.

# Close the database connection when done.
cdm_disconnect(cdm)

## End(Not run)
```

`cdm_from_files` *Create a CDM reference from a folder containing parquet, csv, or feather files*

Description

Create a CDM reference from a folder containing parquet, csv, or feather files

Usage

```
cdm_from_files(
  path,
  format = "auto",
  cdm_version = "5.3",
  cdm_name = NULL,
  as_data_frame = TRUE
)

cdmFromFiles(
  path,
  format = "auto",
  cdmVersion = "5.3",
  cdmName = NULL,
  asDataFrame = TRUE
)
```

Arguments

`path`	A folder where an OMOP CDM v5.4 instance is located.
`format`	What is the file format to be read in? Must be "auto" (default), "parquet", "csv", "feather".
`cdm_version, cdmVersion`	The version of the cdm (5.3 or 5.4)

cdm_name, cdmName
: A name to use for the cdm.

as_data_frame, asDataFrame
: TRUE (default) will read files into R as dataframes. FALSE will read files into R as Arrow Datasets.

Value

A list of dplyr database table references pointing to CDM tables

---

cdm_from_tables *Create a cdm object from local tables*

Description

Create a cdm object from local tables

Usage

```
cdm_from_tables(tables, cdm_name, cohort_tables = list(), cdm_version = NULL)
```

Arguments

tables	List of tables to be part of the cdm object.
cdm_name	Name of the cdm object.
cohort_tables	List of tables that contains cohort, cohort_set and cohort_attrition can be provided as attributes.
cdm_version	Version of the cdm_reference

Value

A cdm_reference object.

Examples

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

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("2025-12-31"),
  period_type_concept_id = 0
)
```

```
cdm <- cdm_from_tables(
  tables = list("person" = person, "observation_period" = observation_period),
  cdm_name = "test"
)

## End(Not run)
```

`cdm_select_tbl` *Select a subset of tables in a cdm reference object*

Description

This function uses syntax similar to `dplyr::select` and can be used to subset a cdm reference object to a specific tables

Usage

```
cdm_select_tbl(cdm, ...)
```

Arguments

`cdm`	A cdm reference object created by `cdm_from_con`
`...`	One or more table names of the tables of the `cdm` object. `tidyselect` is supported, see `dplyr::select()` for details on the semantics.

Value

A cdm reference object containing the selected tables

Examples

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

cdm <- cdm_from_con(con, "main")

cdm_select_tbl(cdm, person)
cdm_select_tbl(cdm, person, observation_period)
cdm_select_tbl(cdm, tbl_group("vocab"))
cdm_select_tbl(cdm, "person")

DBI::dbDisconnect(con)

## End(Not run)
```

cohortAttrition *Get attrition table from a cohort_table object*

Description

Get attrition table from a cohort_table object

Usage

```
cohortAttrition(x)

cohort_attrition(x)
```

Arguments

x	A cohort_table object

cohortSet *Get cohort settings from a cohort_table object*

Description

Get cohort settings from a cohort_table object

Usage

```
cohortSet(x)

cohort_set(x)
```

Arguments

x	A cohort_table object

cohort_count *Get cohort counts from a generated_cohort_set object.*

Description

Get cohort counts from a generated_cohort_set object.

Usage

```
cohort_count(cohort)
```

Arguments

cohort	A generated_cohort_set object.

Value

A table with the counts.

Examples

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

con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con = con, cdm_schema = "main", write_schema = "main")
cdm <- generateConceptCohortSet(
  cdm = cdm, conceptSet = list(pharyngitis = 4112343), name = "new_cohort"
)
cohort_count(cdm$new_cohort)

## End(Not run)
```

cohort_erafy *Collapse cohort records within a certain number of days*

Description

Collapse cohort records within a certain number of days

Usage

```
cohort_erafy(x, gap)

cohortErafy(x, gap)
```

Arguments

x	A generated cohort set
gap	When two cohort records are 'gap' days apart or less the periods will be collapsed into a single record

Value

A lazy query on a generated cohort set

---

cohort_union *Union all cohorts in a cohort set with cohorts in a second cohort set*

Description

Union all cohorts in a cohort set with cohorts in a second cohort set

Usage

```
cohort_union(x, y)

cohortUnion(x, y)
```

Arguments

x	A tbl reference to a cohort table with one or more generated cohorts
y	A tbl reference to a cohort table with one generated cohort

Value

A lazy query that when executed will resolve to a new cohort table with one the same cohort_definitions_ids in x resulting from the union of all cohorts in x with the single cohort in y cohort table

---

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,
  ...
)

compute_query(
  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 = eunomia_dir())
cdm <- cdm_from_con(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)
```

`copy_cdm_to` *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. copy_cdm_to 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

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

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

Arguments

`con`	A DBI datbase 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)

Details

[Experimental]

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 = eunomia_dir())
cdm <- cdm_from_con(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. You must have read and write access to it.

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
)

download_eunomia_data(
  dataset_name = "GiBleed",
  cdm_version = "5.3",
  path_to_data = Sys.getenv("EUNOMIA_DATA_FOLDER"),
  overwrite = FALSE
)
```

Arguments

`overwrite`	Control whether the existing archive file will be overwritten should it already exist.
`dataset_name, 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
`cdm_version, 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) or '5.4'.
`path_to_data, 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.

Value

Invisibly returns the destination if the download was successful.

Examples

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

## End(Not run)
```

`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")
)

eunomia_dir(
  dataset_name = "GiBleed",
  cdm_version = "5.3",
  database_file = tempfile(fileext = ".duckdb")
)
```

Arguments

datasetName, dataset_name
: 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"

cdmVersion, cdm_version
: The OMOP CDM version. Must be "5.3" or "5.4".

databaseFile, database_file
: 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 and 5.4 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/

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(), eunomia_dir())
 cdm <- cdm_from_con(con, "main", "main")
```

```
cdmDisconnect(cdm)

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

# Currently the only 5.4 dataset is synpuf-1k
con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir("synpuf-1k", "5.4"))
cdm <- cdm_from_con(con, "main", "main", achilles_schema = "main")
cdmDisconnect(cdm)


## End(Not run)
```

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

Description

Has the Eunomia dataset been cached?

Usage

```
eunomia_is_available(dataset_name = "GiBleed", cdm_version = "5.3")

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

Arguments

`dataset_name`, `datasetName`
: Name of the Eunomia dataset to check. Defaults to "GiBleed".

`cdm_version`, `cdmVersion`
: Version of the Eunomia dataset to check. Must be "5.3" or "5.4".

Value

TRUE if the eunomia example dataset is available and FASLE otherwise

exampleDatasets *List the available example CDM datasets*

Description

List the available example CDM datasets

Usage

```
exampleDatasets()

example_datasets()
```

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 <- cdm_from_con(con)

## End(Not run)
```

generateCohortSet *Generate a cohort set on a cdm object*

Description

A "chort_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

Each of the three attributes are tidy tables. The implementation of this object is experimental and user feedback is welcome.

[Experimental]

One key design principle is that cohort_table objects are created once and can persist across analysis execution but should not be modified after creation. While it is possible to modify a cohort_table object doing so will invalidate it and it's attributes may no longer be accurate.

Usage

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

generate_cohort_set(
  cdm,
  cohort_set,
  name = "cohort",
  compute_attrition = TRUE,
  overwrite = TRUE
)
```

Arguments

`cdm`	A cdm reference created by CDMConnector. write_schema must be specified.
`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.
`overwrite`	Should the cohort table be overwritten if it already exists? TRUE (default) or FALSE
`cohort_set`, `cohortSet`	Can be a cohortSet object created with `readCohortSet()`
`compute_attrition`, `computeAttrition`	Should attrition be computed? TRUE (default) or FALSE

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con,
                    cdm_schema = "main",
                    write_schema = "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
)

generate_concept_cohort_set(
  cdm,
  concept_set = NULL,
  name = "cohort",
  limit = "first",
```

```
  required_observation = c(0, 0),
  end = "observation_period_end_date",
  subset_cohort = NULL,
  subset_cohort_id = NULL,
  overwrite = TRUE
)
```

Arguments

cdm	A cdm reference object created by CDMConnector::cdmFromCon or CDMConnector::cdm_from_con
conceptSet, concept_set	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, required_observation	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, subset_cohort	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, subset_cohort_id	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 schemas*

Description

This is similar to dbplyr::in_schema but has been tested across multiple database platforms. It only exists to work around some of the limitations of dbplyr::in_schema.

Usage

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

in_schema(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

intersect_cohorts *Intersect all cohorts in a single cohort table*

Description

Intersect all cohorts in a single cohort table

Usage

```
intersect_cohorts(x, cohort_definition_id = 1L)

intersectCohorts(x, cohort_definition_id = 1L)
```

Arguments

x	A tbl reference to a cohort table
cohort_definition_id	A number to use for the new cohort_definition_id
	[Superseded]

Value

A lazy query that when executed will resolve to a new cohort table with one cohort_definition_id resulting from the intersection of all cohorts in the original cohort table

---

list_tables *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

```
list_tables(con, schema = NULL)

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 = eunomia_dir())
listTables(con, schema = "main")

## End(Not run)
```

---

new_generated_cohort_set *Constructor for cohort_table objects*

Description

[Superseded]

Usage

```
new_generated_cohort_set(
  cohort_ref,
  cohort_set_ref = NULL,
  cohort_attrition_ref = NULL,
  cohort_count_ref = NULL,
  overwrite
)

newGeneratedCohortSet(
  cohortRef,
  cohortSetRef = NULL,
  cohortAttritionRef = NULL,
  cohortCountRef = NULL,
  overwrite
)
```

Arguments

`cohort_ref`, `cohortRef`	A `tbl_sql` object that points to a remote cohort table with the following first four columns: cohort_definition_id, subject_id, cohort_start_date, cohort_end_date. Additional columns are optional.
`cohort_set_ref`, `cohortSetRef`	A `tbl_sql` object that points to a remote table with the following first two columns: cohort_definition_id, cohort_name. Additional columns are optional. cohort_definition_id should be a primary key on this table and uniquely identify rows.
`cohort_attrition_ref`, `cohortAttritionRef`	A `tbl_sql` object that points to an attrition table in a remote database with the first column being cohort_definition_id.
`cohort_count_ref`, `cohortCountRef`	A `tbl_sql` object that points to a cohort_count table in a remote database with columns cohort_definition_id, cohort_entries, cohort_subjects.
`overwrite`	Should tables be overwritten if they already exist? TRUE or FALSE (default)

Details

Please use `omopgenerics::newCohortTable()` instead.

This constructor function is to be used by analytic package developers to create `cohort_table` objects.

A `cohort_table` is a set of person-time from an OMOP CDM database. A `cohort_table` can be represented by a table with three columns: subject_id, cohort_start_date, cohort_end_date. Subject_id is the same as person_id in the OMOP CDM. A `cohort_table` is a collection of one or more `cohort_table` and can be represented as a table with four columns: cohort_definition_id, subject_id, cohort_start_date, cohort_end_date.

This constructor function defines the `cohort_table` object in R.

The object is an extension of a tbl_sql object defined in dplyr. This is a lazy database query that points to a cohort table in the database with at least the columns cohort_definition_id, subject_id, cohort_start_date, cohort_end_date. The table could optionally have more columns as well.

In addition the cohort_table object has three optional attributes. These are: cohort_set, cohort_attrition, cohort_count. Each of these attributes is also a lazy SQL query (tbl_sql) that points to a table in a database and is described below.

cohort_set:

cohort_set is a table with one row per cohort_definition_id. The first two columns of the cohort_set table are: cohort_definition_id, and cohort_name. Additional columns can be added. The cohort_set table is meant to store metadata about the cohort definition. Since this table is required it will be created if it it is not supplied.

cohort_attrition:

cohort_attrition is an optional table that stores attrition information recorded during the cohort generation process such as how many persons were dropped at each step of inclusion rule application. The first column of this table should be cohort_definition_id but all other columns currently have no constraints.

cohort_count:

cohort_count is a option attribute table that records the number of records and the number of unique persons in each cohort in a cohort_table. It is derived metadata that can be re-derived as long as cohort_set, the complete list of cohorts in the set, is available. Column names of cohort_count are: cohort_definition_id, number_records, number_subjects. This table is required for cohort_table objects and will be created if not supplied.

Value

A cohort_table object that is a tbl_sql reference to a cohort table in the write_schema of an OMOP CDM

Examples

```
## Not run:
 # This function is for developers who are creating cohort_table
 # objects in their packages. The function should accept a cdm_reference
 # object as the first argument and return a cdm_reference object with the
 # cohort table added. The second argument should be `name` which will be
 # the prefix for the database tables, the name of the cohort table in the
 # database and the name of the cohort table in the cdm object.
 # Other optional arguments can be added after the first two.

 generateCustomCohort <- function(cdm, name, ...) {

   # accept a cdm_reference object as input
   checkmate::assertClass(cdm, "cdm_reference")
   con <- attr(cdm, "dbcon")

   # Create the tables in the database however you like
   # All the tables should be prefixed with `name`
```

```
  # The cohort table should be called `name` in the database

  # Create the dplyr table references
  cohort_ref <- dplyr::tbl(con, name)
  cohort_set <- dplyr::tbl(con, paste0(name, "_set"))
  cohort_attrition_ref <- dplyr::tbl(con, paste0(name, "_attrition"))
  cohort_count_ref <- dplyr::tbl(con, paste0(name, "_count"))

  # add to the cdm
  cdm[[name]] <- cohort_ref

  # create the generated cohort set object using the constructor
  cdm[[name]] <- new_generated_cohort_set(
     cdm[[name]],
     cohort_set_ref = cohort_set_ref,
     cohort_attrition_ref = cohort_attrition_ref,
     cohort_count_ref = cohort_count_ref)

  return(cdm)
}

## End(Not run)
```

`read_cohort_set` *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. 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.

Usage

```
read_cohort_set(path)

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.

recordCohortAttrition *Add attrition reason to a cohort_table object*

Description

Update the cohort attrition table with new counts and a reason for attrition.

Usage

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

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

Arguments

cohort	A generated cohort set
reason	The reason for attrition as a character string
cohortId	Cohort definition id of the cohort you want to update the attrition

Value

The cohort object with the attributes created or updated.

[Experimental]

Examples

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

con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con = con, cdm_schema = "main", write_schema = "main")
cdm <- generateConceptCohortSet(
  cdm = cdm, conceptSet = list(pharyngitis = 4112343), name = "new_cohort"
)

settings(cdm$new_cohort)
cohortCount(cdm$new_cohort)
cohortAttrition(cdm$new_cohort)

cdm$new_cohort <- cdm$new_cohort %>%
  filter(cohort_start_date >= as.Date("2010-01-01"))

cdm$new_cohort <- updateCohortAttributes(
  cohort = cdm$new_cohort, reason = "Only events after 2010"
)

settings(cdm$new_cohort)
```

```
cohortCount(cdm$new_cohort)
cohortAttrition(cdm$new_cohort)

## End(Not run)
```

`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")

require_eunomia(dataset_name = "GiBleed", cdm_version = "5.3")
```

Arguments

`dataset_name`, `datasetName`
: Name of the Eunomia dataset to check. Defaults to "GiBleed".

`cdm_version`, `cdmVersion`
: Version of the Eunomia dataset to check. Must be "5.3" or "5.4".

Value

Path to eunomia database.

`snapshot` *Extract CDM metadata*

Description

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

Usage

```
snapshot(cdm)
```

Arguments

`cdm` A cdm object

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(), eunomia_dir())
cdm <- cdm_from_con(con, "main")
snapshot(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

---

stow *Collect a list of lazy queries and save the results as files*

Description

Collect a list of lazy queries and save the results as files

Usage

```
stow(cdm, path, format = "parquet")
```

Arguments

cdm	A cdm object
path	A folder to save the cdm object to
format	The file format to use: "parquet" (default), "csv", "feather" or "duckdb".

Value

Invisibly returns the cdm input

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = eunomia_dir())
vocab <- cdm_from_con(con, "main") %>%
  cdm_select_tbl("concept", "concept_ancestor")
stow(vocab, here::here("vocab_tables"))
DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

summarise_quantile *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

```
summarise_quantile(.data, x = NULL, probs, name_suffix = "value")

summarize_quantile(.data, x = NULL, probs, name_suffix = "value")

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

summarizeQuantile(.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].
`name_suffix`, `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)) %>%
 summarise_quantile(mpg, probs = c(0, 0.2, 0.4, 0.6, 0.8, 1),
                    name_suffix = "quant") %>%
 dplyr::collect()

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

`tbl_group` *CDM table selection helper*

Description

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

Usage

```
tbl_group(group)

tblGroup(group)
```

Arguments

`group`	A character vector of CDM table groups: "vocab", "clinical", "all", "default", "derived".

Details

" alt" alt

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 <- cdm_from_con(con, cdm_name = "test", cdm_schema = "public") %>%
  cdm_select_tbl(tbl_group("vocab"))

## End(Not run)
```

`union_cohorts` *Union all cohorts in a single cohort table*

Description

Union all cohorts in a single cohort table

Usage

```
union_cohorts(x, cohort_definition_id = 1L)

unionCohorts(x, cohort_definition_id = 1L)
```

Arguments

`x`	A tbl reference to a cohort table
`cohort_definition_id`	A number to use for the new cohort_definition_id **[Superseded]**

Value

A lazy query that when executed will resolve to a new cohort table with one cohort_definition_id resulting from the union of all cohorts in the original cohort table

uniqueTableName *Create a unique table name for temp tables*

Description

Create a unique table name for temp tables

Usage

```
uniqueTableName()

unique_table_name()
```

Value

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

validate_cdm *Validation report for a CDM*

Description

Print a short validation report for a cdm object. The validation includes checking that column names are correct and that no tables are empty. A short report is printed to the console. This function is meant for interactive use.

Usage

```
validate_cdm(cdm)

validateCdm(cdm)
```

Arguments

cdm	A cdm reference object.

Value

Invisibly returns the cdm input

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con, cdm_schema = "main")
validate_cdm(cdm)
DBI::dbDisconnect(con)

## 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" or "5.4"

Examples

```
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con, cdm_schema = "main", write_schema = "main")
version(cdm)

DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

Index