

# Package ‘CDMConnector’

January 10, 2026

**Title** Connect to an OMOP Common Data Model

**Version** 2.3.0

**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** 7.3.3

**Depends** R (>= 4.1.0)

**Imports** 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 (>= 1.2.0)

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

**Enhances** arrow

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**VignetteBuilder** knitr

**NeedsCompilation** no

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

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

**Maintainer** Adam Black <black@ohdsi.org>

**Repository** CRAN

**Date/Publication** 2026-01-10 07:52:52 UTC

## Contents

appendPermanent . . . . .	3
asDate . . . . .	4
benchmarkCDMConnector . . . . .	4
cdmCon . . . . .	5
cdmDisconnect.db_cdm . . . . .	6
cdmFlatten . . . . .	6
cdmFromCon . . . . .	8
cdmSample . . . . .	10
cdmSubset . . . . .	11
cdmSubsetCohort . . . . .	12
cdmWriteSchema . . . . .	13
computeDataHashByTable . . . . .	14
computeQuery . . . . .	15
copyCdmTo . . . . .	16
dateadd . . . . .	17
datediff . . . . .	17
datepart . . . . .	18
dbms . . . . .	19
dbSource . . . . .	20
downloadEunomiaData . . . . .	20
dropTable.db_cdm . . . . .	21
eunomiaDir . . . . .	21
eunomiaIsAvailable . . . . .	23
exampleDatasets . . . . .	23
generateCohortSet . . . . .	24
generateConceptCohortSet . . . . .	25
inSchema . . . . .	27
listTables . . . . .	27
readCohortSet . . . . .	28
requireEunomia . . . . .	28
snapshot . . . . .	29
summariseQuantile . . . . .	30
summariseQuantile2 . . . . .	31
tblGroup . . . . .	32
version . . . . .	33

**Index**

**34**

---

appendPermanent	<i>Run a dplyr query and add the result set to an existing</i>
-----------------	--

---

**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	<i>as.Date dbplyr translation wrapper</i>
--------	---

---

### 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)
```

---

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, dropWriteSchema = FALSE, ...)
```

### Arguments

cdm	cdm reference
dropWriteSchema	Whether to drop tables in the writeSchema
...	Not used

---

```
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)
```

---

 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,
  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 v5.4 or v5.3 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.
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", 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	<i>Subset a cdm to the individuals in one or more cohorts</i>
-----------------	---

---

## 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)

```

---

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)
```

---

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**

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

**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	<i>Add days or years to a date in a dplyr query</i>
---------	---

---

**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	<i>Compute the difference between two days</i>
----------	--

---

**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	<i>Get the database management system (dbms) from a cdm_reference or DBI connection</i>
------	---

---

**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	<i>Create a source for a cdm in a database.</i>
----------	---

---

**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	<i>Download Eunomia data files</i>
---------------------	------------------------------------

---

**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 <a href="https://github.com/darwin-eu/EunomiaDatasets">https://github.com/darwin-eu/EunomiaDatasets</a> . 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) or '5.4'.
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	<i>Drop table from a database backed cdm object</i>
------------------	---

---

**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	<i>Create a copy of an example OMOP CDM dataset</i>
------------	---

---

**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"
cdmVersion	The OMOP CDM version. Must be "5.3" or "5.4".
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://synthetichealth.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/>

Setup: To use the eunomiaDir function please set the EUNOMIA\_DATA\_FOLDER in your .Renv 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" or "5.4".

### 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	<i>Generate a cohort set on a cdm object</i>
-------------------	--

---

**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

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
)
```



**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 <ul style="list-style-type: none"> <li>• "first" will include only the first occurrence of any event in the concept set in the cohort.</li> <li>• "all" will include all occurrences of the events defined by the concept set in the cohort.</li> </ul>
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? <ul style="list-style-type: none"> <li>• "observation_period_end_date" (default): The earliest observation_period_end_date after the event start date</li> <li>• numeric scalar: A fixed number of days from the event start date</li> <li>• "event_end_date": The event end date. If the event end date is not populated then the event start date will be used</li> </ul>
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	<i>Helper for working with compound schemas</i>
----------	---

---

**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)
```

**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 <code>dbms(connection)</code>

**Value**

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

---

listTables	<i>List tables in a schema</i>
------------	--------------------------------

---

**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 <code>NULL</code> , returns <code>DBI::dbListTables(con)</code> .

**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	<i>Read a set of cohort definitions into R</i>
---------------	--

---

**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**

```
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.
------	---

---

requireEunomia	<i>Require eunomia to be available. The function makes sure that you can later create a eunomia database with eunomiaDir().</i>
----------------	---

---

**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" or "5.4".

**Value**

Path to eunomia database.

---

snapshot	<i>Extract CDM metadata</i>
----------	-----------------------------

---

**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

<code>.data</code>	lazy data frame backed by a database query.
<code>x</code>	column name whose sample quantiles are wanted.
<code>probs</code>	numeric vector of probabilities with values in [0,1].
<code>nameSuffix</code>	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

<code>.data</code>	lazy data frame backed by a database query created by <code>dplyr::tbl()</code> .
<code>x</code>	A string vector of column names whose sample quantiles are wanted.
<code>probs</code>	A numeric vector of probabilities with values in $[0,1]$ .
<code>nameSuffix</code>	A single character string, evaluated by <code>glue::glue()</code> 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)
```

## Arguments

<code>group</code>	A character vector of CDM table groups: "vocab", "clinical", "all", "default", "derived".
--------------------	---

## 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" or "5.4"

**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, 4

cdmCon, 5  
cdmDisconnect.db\_cdm, 6  
cdmFlatten, 6  
cdmFromCon, 8  
cdmSample, 10  
cdmSubset, 11  
cdmSubsetCohort, 12  
cdmWriteSchema, 13  
computeDataHashByTable, 14  
computeQuery, 15  
copyCdmTo, 16

dateadd, 17  
datediff, 17  
datepart, 18  
dbms, 19  
dbSource, 20  
downloadEunomiaData, 20  
dropTable.db\_cdm, 21

eunomiaDir, 21  
eunomiaIsAvailable, 23  
exampleDatasets, 23

generateCohortSet, 24  
generateConceptCohortSet, 25

inSchema, 27

listTables, 27

readCohortSet, 28  
requireEunomia, 28

snapshot, 29  
summariseQuantile, 30

summariseQuantile2, 31

tblGroup, 32

version, 33