Package ‘CDMConnector’

October 31, 2023

Title Connect to an OMOP Common Data Model
Version 1.2.0
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 7.2.3
Depends R (>= 4.0)
Imports dplyr, DBI (>= 0.3.0), checkmate, dbplyr (>= 2.4.0), cli, purr, rlang, tidyselect, readr, glue, waldo, methods, withr, lifecycle, jsonlite, stringr, stringi, fs, generics, tidyr, pool
Suggests arrow, SqlRender, rJava, covr, knitr, rmarkdown, duckdb, RSQLite, RPostgres, odbc, ggplot2, bigquery, testthat (>= 3.0.0), DatabaseConnector, lubridate, tibble, zeallot
Enhances CirceR, Capr
Config/testthat/edition 3
Config/testthat/parallel true
VignetteBuilder knitr
Additional_repositories https://OHDSI.github.io/drat
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
2023-10-31 16:40:17 UTC

### R topics documented:

<table>
<thead>
<tr>
<th>R topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>appendPermanent</td>
<td>3</td>
</tr>
<tr>
<td>asDate</td>
<td>4</td>
</tr>
<tr>
<td>assert_tables</td>
<td>5</td>
</tr>
<tr>
<td>assert_write_schema</td>
<td>6</td>
</tr>
<tr>
<td>cdmDisconnect</td>
<td>6</td>
</tr>
<tr>
<td>cdmFlatten</td>
<td>7</td>
</tr>
<tr>
<td>cdmName</td>
<td>8</td>
</tr>
<tr>
<td>cdmSample</td>
<td>9</td>
</tr>
<tr>
<td>cdmSubset</td>
<td>10</td>
</tr>
<tr>
<td>cdmSubsetCohort</td>
<td>11</td>
</tr>
<tr>
<td>cdm_from_con</td>
<td>13</td>
</tr>
<tr>
<td>cdm_from_files</td>
<td>14</td>
</tr>
<tr>
<td>cdm_select_tbl</td>
<td>15</td>
</tr>
<tr>
<td>cohortAttrition</td>
<td>16</td>
</tr>
<tr>
<td>cohortCount</td>
<td>16</td>
</tr>
<tr>
<td>cohortSet</td>
<td>17</td>
</tr>
<tr>
<td>cohort_erafy</td>
<td>17</td>
</tr>
<tr>
<td>cohort_union</td>
<td>18</td>
</tr>
<tr>
<td>collect.cdm_reference</td>
<td>18</td>
</tr>
<tr>
<td>computeQuery</td>
<td>19</td>
</tr>
<tr>
<td>copy_cdm_to</td>
<td>20</td>
</tr>
<tr>
<td>dateadd</td>
<td>21</td>
</tr>
<tr>
<td>datediff</td>
<td>22</td>
</tr>
<tr>
<td>datepart</td>
<td>22</td>
</tr>
<tr>
<td>dbms</td>
<td>23</td>
</tr>
<tr>
<td>downloadEunomiaData</td>
<td>24</td>
</tr>
<tr>
<td>dropTable</td>
<td>25</td>
</tr>
<tr>
<td>eunomiaDir</td>
<td>26</td>
</tr>
<tr>
<td>eunomia_is_available</td>
<td>27</td>
</tr>
<tr>
<td>exampleDatasets</td>
<td>28</td>
</tr>
<tr>
<td>generateCohortSet</td>
<td>28</td>
</tr>
<tr>
<td>generateConceptCohortSet</td>
<td>30</td>
</tr>
<tr>
<td>inSchema</td>
<td>31</td>
</tr>
<tr>
<td>intersect_concepts</td>
<td>32</td>
</tr>
<tr>
<td>list_tables</td>
<td>32</td>
</tr>
<tr>
<td>new_generated_cohort_set</td>
<td>33</td>
</tr>
<tr>
<td>print.cdm_reference</td>
<td>35</td>
</tr>
<tr>
<td>read_cohort_set</td>
<td>36</td>
</tr>
<tr>
<td>recordCohortAttrition</td>
<td>37</td>
</tr>
<tr>
<td>snapshot</td>
<td>38</td>
</tr>
<tr>
<td>stow</td>
<td>38</td>
</tr>
</tbody>
</table>
Run a dplyr query and add the result set to an existing table.

**Usage**

```r
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**

```r
## 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) %>%
```
computeQuery("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**

```r
## 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
  )))

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)
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdm</td>
<td>A cdm object</td>
</tr>
<tr>
<td>add</td>
<td>An optional AssertCollection created by checkmate::makeAssertCollection() that errors should be added to.</td>
</tr>
</tbody>
</table>

Value
Invisibly returns the cdm object

cdmDisconnect  

Disconnect the connection of the cdm object

Description
Disconnect the connection of the cdm object

Usage
cdmDisconnect(cdm)
cdm_disconnect(cdm)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdm</td>
<td>cdm reference</td>
</tr>
</tbody>
</table>
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

```r
cdmFlatten(
  cdm,
  domain = c("condition", "drug", "procedure"),
  includeConceptName = TRUE
)
```

```r
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

```r
# 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 x 8
### # A tibble: 213 × 8
### #> person_id observation_concept_id start_date end_date type_ concept_concept_id concept_concept_name domain observation_concept_name observation_value
### #> <dbl> <dbl> <date> <date> <dbl> <chr> <chr> <chr> <chr> <chr> <dbl> <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**

```r
cdmName(cdm)

cdm_name(cdm)
```

**Arguments**

```r
cdm
```

A cdm object

**Value**

The name of the CDM as a character string
### Examples

```r
## Not run:
library(CDMConnector)
con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
cdm <- cdm_from_con(con, "main")
cdmName(cdm)
#> [1] "Synthea synthetic health database"

cdm <- cdm_from_con(con, "main", cdm_name = "Example CDM")
cdmName(cdm)
#> [1] "Example CDM"

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

```r
cdmSample(cdm, n)
cdm_sample(cdm, n)
```

### Arguments

- **cdm**: A cdm_reference object
- **n**: Number of persons to include in the cdm

### Details

[Experimental]

### Value

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

```r
## 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)
```

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**

```r
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

```r
## 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

```r
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**: 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)

Details

[Experimental]

Value

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

Examples

```r
## 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")

cdmGibGbleed <- cdmSubsetCohort(cdm, cohortTable = "gibleed")
cdmGibGbleed$person %>%
  tally()
#> # Source: SQL [1 x 1]
#> # Database: DuckDB 0.6.1
#> n
#> <dbi>
#> 1 237
cdm$person %>%
  tally()
#> # Source: SQL [1 x 1]
#> # Database: DuckDB 0.6.1
#> n
```
# cdm_from_con

```r
#> <dbl>
#> 1 2694
```

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

```r
cdm_from_con(
  con,
  cdm_schema = NULL,
  write_schema = NULL,
  cohort_tables = NULL,
  cdm_version = "5.3",
  cdm_name = NULL
)
```

```r
cdmFromCon(
  con,
  cdmSchema = NULL,
  writeSchema = NULL,
  cohortTables = NULL,
  cdmVersion = "5.3",
  cdmName = 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_from_files

**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.

**Value**

A list of dplyr database table references pointing to CDM tables

---

**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**

```r
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_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 GeneratedCohortSet object*

**Description**

Get attrition table from a GeneratedCohortSet object

**Usage**

```r
cohortAttrition(x)
```

```r
cohort_attrition(x)
```

**Arguments**

- **x**
  
  A generatedCohortSet object

---

**cohortCount**

*Get cohort counts from a GeneratedCohortSet object*

**Description**

Get cohort counts from a GeneratedCohortSet object

**Usage**

```r
cohortCount(x)
```

```r
cohort_count(x)
```

**Arguments**

- **x**
  
  A generatedCohortSet object
**cohortSet**

*Get cohort settings from a GeneratedCohortSet object*

**Description**

Get cohort settings from a GeneratedCohortSet object

**Usage**

cohortSet(x)

cohort_set(x)

**Arguments**

x  
A generatedCohortSet object

---

**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

---

**collect.cdm_reference**  
*Bring a remote CDM reference into R*

**Description**

This function calls collect on a list of lazy queries and returns the result as a list of dataframes.

**Usage**

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

**Arguments**

- `x`: A cdm_reference object.  
- `...`: Not used. Included for compatibility.

**Value**

A cdm_reference object that is a list of R dataframes.
computeQuery

Examples

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

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

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 or FALSE (default) Ignored if temporary = TRUE.
- `...` Further arguments passed on the `dplyr::compute`
Value

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

Examples

```r
## 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**

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

```r
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)
**dateadd**

**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**

```r
## 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**

```r
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**

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

## End(Not run)

### Description

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

### Usage

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

```r
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.

- **datasetName, datasetName**
  The data set name as found on https://github.com/darwin-eu/EunomiaDatasets. The data set name corresponds to the folder with the data set ZIP files

- **cdmVersion, cdmVersion**
  The OMOP CDM version. This version will appear in the suffix of the data file, for example: synpuf_5.3.zip. Default: '5.3'

- **pathToData, 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.
dropTable

Value

Invisibly returns the destination if the download was successful.

Examples

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

dropTable

Drop tables from write_schema of a cdm object

Description

cdm objects can have zero or more cohort tables stored in a special schema where the user has write access. This function removes tables from a cdm’s write_schema

Usage

dropTable(cdm, name, verbose = FALSE)
drop_table(cdm, name, verbose = FALSE)

Arguments

cdm
A cdm reference

name
A character vector of tables in the cdm’s write_schema or a tidyselect specification of tables to drop. (e.g. starts_with("temp"), matches("study01"), etc.)

verbose
Print a message when dropping a table? TRUE or FALSE (default)

Value

Returns the cdm object with selected tables removed

Examples

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

# create two temporary tables in the remote database from a query with a common prefix
cdm$tmp_table <- cdm$concept %>%
dplyr::count(domain_id == "Drug") %>%
computeQuery("tmp_table", temporary = FALSE, schema = "main")
`cdm$tmp_table2 <- cdm$concept %>%
defer::count(domain_id == "Condition") %>%
computeQuery("tmp_table2", temporary = FALSE, schema = "main")`

```r
stringr::str_subset(DBI::dbListTables(con), "tmp")
#> [1] "tmp_table" "tmp_table2"
stringr::str_subset(names(cdm), "tmp")
#> [1] "tmp_table" "tmp_table2"
```

# drop tables with a common prefix
```r
cdm <- dropTable(cdm, name = dplyr::starts_with("tmp"))
```

```r
stringr::str_subset(DBI::dbListTables(con), "tmp")
# character(0)
stringr::str_subset(names(cdm), "tmp")
# character(0)
```

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

## End(Not run)

---

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

### Description

Creates a copy of a Eunomia database, 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

```r
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-"
eunomia_is_available

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_jointReplacement-10k", "synthea-
veteran_prostate_cancer-10k", "synthea-veterans-10k", "synthea-weight_loss-10k"
cdmVersion, cdm_version
  The OMOP CDM version. Currently only "5.3" is supported.
databaseFile, database_file
  The full path to the new copy of the example CDM dataset.

Value
  The file path to the new Eunomia dataset copy

Examples
  ## Not run:
  con <- DBI::dbConnect(duckdb::duckdb(), eunomiaDir("GiBleed"))
  DBI::dbDisconnect(con, shutdown = TRUE)
  ## 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 FALSE otherwise
exampleDatasets  
*List the available example CDM datasets*

**Description**

List the available example CDM datasets

**Usage**

```r
exampleDatasets()
example_datasets()
```

**Value**

A character vector with example CDM dataset identifiers

**Examples**

```r
## 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 "GeneratedCohortSet" 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 GeneratedCohortSet objects are created once and can persist across analysis execution but should not be modified after creation. While it is possible to modify a GeneratedCohortSet object doing so will invalidate it and its attributes may no longer be accurate.

Usage

generateCohortSet(
  cdm,
  cohortSet,
  name = "cohort",
  computeAttrition = TRUE,
  overwrite = FALSE
)

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

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.
overwrite Should the cohort table be overwritten if it already exists? TRUE or FALSE (default)
cohort_set, cohortSet Can be a cohortSet object created with readCohortSet(), a single Capr cohort definition, or a named list of Capr cohort definitions.
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"))
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 = "cohort",
  limit = "first",
  requiredObservation = c(0, 0),
  end = "observation_period_end_date",
  overwrite = FALSE
)

generate_concept_cohort_set(
  cdm,
  concept_set = NULL,
  name = "cohort",
  limit = "first",
  required_observation = c(0, 0),
  end = "observation_period_end_date",
  overwrite = FALSE
)
**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 Capr concept sets
- **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
- **overwrite**: Should the cohort table be overwritten if it already exists? TRUE or FALSE (default)

**Value**

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

---

**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**

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

```r
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)`
list_tables

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 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
new_generated_cohort_set

Value

A character vector of table names

Examples

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

## End(Not run)
```

new_generated_cohort_set

Constructor for GeneratedCohortSet objects

Description

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

Usage

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

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

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

A generatedCohort is a set of person-time from an OMOP CDM database. A generatedCohort 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 generatedCohortSet is a collection of one or more generatedCohorts 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 generatedCohortSet 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 generatedCohortSet 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 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 generatedCohortSet. 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 generatedCohortSet objects and will be created if not supplied.
Value

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

Examples

```r
## Not run:
# This function is for developers who are creating generatedCohortSet
# 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)
```
Description
Print a CDM reference object

Usage

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

Arguments

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

Value

Invisibly returns the input

---

**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

```r
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 GeneratedCohortSet 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"
)

cohortSet(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"
)

cohortSet(cdm$new_cohort)
cohortCount(cdm$new_coohort)
cohortAttrition(cdm$new_coohort)

## End(Not run)

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

```r
## 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

```r
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 \texttt{quantile(stats)} with argument type = 1. See discussion in Hyndman and Fan (1996). Results differ from \texttt{PERCENTILE_CONT} natively implemented in various DBMS, where returned values are equal to \texttt{quantile(stats)} with default argument type = 7.

Value

An object of the same type as `.data`

Examples

```r
## 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)
```
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

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

cdm <- cdm_from_con(con) %>%
  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

```r
union_cohorts(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]
**validate_cdm**

**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

```r
## 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**

```r
version(cdm)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdm</td>
<td>A <code>cdm</code> object</td>
</tr>
</tbody>
</table>

**Value**

"5.3" or "5.4"

**Examples**

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

DBI::dbDisconnect(con, shutdown = TRUE)

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

Subset a cdm reference object

### Usage

```r
## S3 method for class 'cdm_reference'
x[[i]]
```

### Arguments

- **x**: A cdm reference
- **i**: The name or index of the table to extract from the cdm object

### Value

A single cdm table reference
Subset a cdm reference object

**Description**

Subset a cdm reference object

**Usage**

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

**Arguments**

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

**Value**

A single cdm table reference
Index

[.cdm_reference, 44
[[.cdm_reference, 44
$.cdm_reference, 45

append_permmanent (appendPermanent), 3
appendPermanent, 3
as_date (asDate), 4
asDate, 4
assert_tables, 5
assert_write_schema, 6
assertTables (assert_tables), 5
assertWriteSchema
  (assert_write_schema), 6

cdm_disconnect (cdmDisconnect), 6
cdm_flatten (cdmFlatten), 7
cdm_from_con, 13
cdm_from_files, 14
cdm_name (cdmName), 8
cdm_sample (cdmSample), 9
cdm_select_tbl, 15
cdm_subset (cdmSubset), 10
cdm_subset_cohort (cdmSubsetCohort), 11
cdmDisconnect, 6
cdmFlatten, 7
cdmFromCon (cdm_from_con), 13
cdmFromFiles (cdm_from_files), 14
cdmName, 8
cdmSample, 9
cdmSubset, 10
cdmSubsetCohort, 11
cohort_attrition (cohortAttrition), 16
cohort_count (cohortCount), 16
cohort_erafy, 17
cohort_set (cohortSet), 17
cohort_union, 18
cohortAttrition, 16
cohortCount, 16
cohortErafy (cohort_erafy), 17
cohortSet, 17

cohortUnion (cohort_union), 18
collect.cdm_reference, 18
compute_query (computeQuery), 19
computeQuery, 19
copy_cdm_to, 20
copyCdmTo (copy_cdm_to), 20
dateadd, 21
datediff, 22
datepart, 22
dbms, 23
download_eunomia_data
  (downloadEunomiaData), 24
downloadEunomiaData, 24
drop_table (dropTable), 25
dropTable, 25
eunomia_dir (eunomiaDir), 26
eunomia_is_available, 27
eunomiaDir, 26
eunomiaIsAvailable
  (eunomia_is_available), 27
example_datasets (exampleDatasets), 28
exampleDatasets, 28
generate_cohort_set
  (generateCohortSet), 28
generate_concept_cohort_set
  (generateConceptCohortSet), 30
generateCohortSet, 28
generateConceptCohortSet, 30
in_schema (inSchema), 31
inSchema, 31
intersect_cohorts, 32
intersectCohorts (intersect_cohorts), 32
list_tables, 32
listTables (list_tables), 32
new_generated_cohort_set, 33
newGeneratedCohortSet
   (new_generated_cohort_set), 33

print.cdm_reference, 35

read_cohort_set, 36
readCohortSet (read_cohort_set), 36
record_cohort_attrition
   (recordCohortAttrition), 37
recordCohortAttrition, 37

snapshot, 38
stow, 38
summarise_quantile, 39
summariseQuantile (summarise_quantile),
   39
summarize_quantile
   (summarise_quantile), 39
summarizeQuantile (summarise_quantile),
   39

tbl_group, 40
tblGroup (tbl_group), 40

union_cohorts, 41
unionCohorts (union_cohorts), 41
unique_table_name (uniqueTableName), 42
uniqueTableName, 42

validate_cdm, 42
validateCdm (validate_cdm), 42
version, 43