Package 'CohortGenerator'

July 10, 2026

Type Package

Title Cohort Generation for the OMOP Common Data Model

Version 1.1.1

Date 2026-07-09

Maintainer Anthony Sena <sena@ohdsi.org>

Description Generate cohorts and subsets using an Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) Database. Cohorts are defined using 'CIRCE' (<https://github.com/ohdsi/circe-be>) or SQL compatible with 'SqlRender' (<https://github.com/OHDSI/SqlRender>).

Depends DatabaseConnector (>= 5.0.0), R (>= 4.1.0), R6

Imports checkmate, digest, dplyr, lubridate, methods, ParallelLogger (>= 3.0.0), readr (>= 2.1.0), rlang, jsonlite, ResultModelManager (>= 0.6.0), SqlRender (>= 1.11.1), stringi (>= 1.7.6), tibble

Suggests CirceR (>= 1.1.1), duckdb, Eunomia, ggplot2, knitr, rmarkdown, testthat, withr, zip

License Apache License

VignetteBuilder knitr

URL https://ohdsi.github.io/CohortGenerator/, https://github.com/OHDSI/CohortGenerator

BugReports https://github.com/OHDSI/CohortGenerator/issues

Encoding UTF-8

Language en-US

LazyData true

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Anthony Sena [aut, cre], Jamie Gilbert [aut], Gowtham Rao [aut],

Freddy Avila Cruz [aut],
Martijn Schuemie [aut],
Observational Health Data Science and Informatics [cph]

Repository CRAN

Date/Publication 2026-07-10 18:50:02 UTC

Contents

addCohortSubsetDefinition . . . 3
addCohortTemplateDefintion . . . 4
addExcludeOnIndexSubsetDefinition . . . 5
addIndicationSubsetDefinition . . . 7
addRestrictionSubsetDefinition . . . 9
addSqlCohortDefinition . . . 11
addUnionCohortDefinition . . . 13
checkAndFixCohortDefinitionSetDataTypes . . . 13
CohortSubsetDefinition . . . 14
CohortSubsetOperator . . . 16
CohortTemplateDefinition . . . 18
computeChecksum . . . 20
computeCohortAttrition . . . 21
createAtcCohortTemplateDefinition . . . 22
createCohortSubset . . . 23
createCohortSubsetDefinition . . . 23
createCohortSubsetOperator . . . 24
createCohortTables . . . 25
createCohortTemplateDefintion . . . 26
createDemographicSubset . . . 26
createDemographicSubsetOperator . . . 27
createEmptyCohortDefinitionSet . . . 28
createEmptyNegativeControlOutcomeCohortSet . . . 28
createLimitSubset . . . 29
createLimitSubsetOperator . . . 29
createResultsDataModel . . . 30
createRxNormCohortTemplateDefinition . . . 31
createSnomedCohortTemplateDefinition . . . 32
createSubsetCohortWindow . . . 33
createUnionCohortTemplate . . . 34
DemographicSubsetOperator . . . 34
dropCohortStatsTables . . . 36
exportCohortStatsTables . . . 37
exportCohortSubsetStatsTables . . . 38
generateCohortSet . . . 40
generateNegativeControlOutcomeCohorts . . . 41
getCohortCounts . . . 43
getCohortDefinitionSet . . . 44
getCohortInclusionRules . . . 45

getCohortStats . . . 46
getCohortTableNames . . . 47
getCohortValidationCounts . . . 48
getDataMigrator . . . 50
getExcludeOnIndexSubsetDefinitionIds . . . 50
getIndicationSubsetDefinitionIds . . . 51
getLastGeneratedCohortChecksums . . . 51
getRestrictionSubsetDefinitionIds . . . 52
getResultsDataModelSpecifications . . . 53
getSubsetDefinitions . . . 53
getTemplateDefinitions . . . 54
insertInclusionRuleNames . . . 54
isCamelCase . . . 55
isCohortDefinitionSet . . . 56
isFormattedForDatabaseUpload . . . 56
isSnakeCase . . . 57
LimitSubsetOperator . . . 57
migrateDataModel . . . 58
omopCdmDrugExposure . . . 59
omopCdmPerson . . . 59
readCsv . . . 60
runCohortGeneration . . . 61
sampleCohortDefinitionSet . . . 63
saveCohortDefinitionSet . . . 65
saveCohortSubsetDefinition . . . 66
SubsetCohortWindow . . . 66
SubsetOperator . . . 67
uploadResults . . . 69
writeCsv . . . 70

Index **72**

`addCohortSubsetDefinition` *Add cohort subset definition to a cohort definition set*

Description

Given a subset definition and cohort definition set, this function returns a modified cohortDefinitionSet That contains cohorts that's have parent's contained within the base cohortDefinitionSet

Also adds the columns subsetParent and isSubset that denote if the cohort is a subset and what the parent definition is.

Usage

```
addCohortSubsetDefinition(
  cohortDefinitionSet,
  cohortSubsetDefintion,
  targetCohortIds = NULL,
  overwriteExisting = FALSE
)
```

Arguments

cohortDefinitionSet	data.frame that conforms to CohortDefinitionSet
cohortSubsetDefintion	CohortSubsetDefinition instance
targetCohortIds	Cohort ids to apply subset definition to. If not set, subset definition is applied to all base cohorts in set (i.e. those that are not defined by subsetOperators). Applying to cohorts that are already subsets is permitted, however, this should be done with care and identifiers must be specified manually
overwriteExisting	Overwrite existing subset definition of the same definitionId if present

---

addCohortTemplateDefintion

Add Cohort template definition to cohort set

Description

Adds a cohort template definition to an existing cohort definition set or creates one if none provided

Usage

```
addCohortTemplateDefintion(
  cohortDefinitionSet = createEmptyCohortDefinitionSet(),
  cohortTemplateDefintion
)
```

Arguments

cohortDefinitionSet	The `cohortDefinitionSet` argument must be a data frame with the following columns: **cohortId** The unique integer identifier of the cohort **cohortName** The cohort's name **sql** The OHDSI-SQL used to generate the cohort Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`cohortTemplateDefintion`

An instance of CohortTemplateDefinition (or subclass). See [@seealso [createCohortTemplateDefinition()]].

`addExcludeOnIndexSubsetDefinition`

Add exclude on index subset definition

Description

The purpose of this subset recipe is to exclude all individuals if their index aligns with the specified exclusion cohort ids. If the index date of the exclusionCohortIds aligns with the targetCohortIds (or it lies within some relative window of the target cohort start date) then they will be excluded from the resulting sub population.

This may be used in situations where an outcome cohort may contain individuals treated for a target medication, complicating calculation of incidence rates.

Usage

```
addExcludeOnIndexSubsetDefinition(
  cohortDefinitionSet,
  subsetDefinitionName,
  subsetCohortNameTemplate = "@baseCohortName - @subsetDefinitionName",
  targetCohortIds,
  exclusionCohortIds,
  exclusionWindow = 0,
  subsetDefinitionId,
  cohortCombinationOperator = "any"
)
```

Arguments

`cohortDefinitionSet`

The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`subsetDefinitionName`

name of the subset definition (used in resulting cohort definitions)

`subsetCohortNameTemplate`

template string format for naming resulting cohorts

`targetCohortIds`
: Set of integer cohort IDs. Must be within the cohort definition set.

`exclusionCohortIds`
: cohort ids to exclude members of target from

`exclusionWindow`
: Days Default is 0 (target index date). by changing this you can adjust the period around target index for which you would exclude members.

`subsetDefinitionId`
: Unique integer Id of the subset definition

`cohortCombinationOperator`
: Logic for multiple indication cohort IDs: any (default) or all.

Examples

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

initialSet <- getCohortDefinitionSet(
  settingsFileName = "testdata/name/Cohorts.csv",
  jsonFolder = "testdata/name/cohorts",
  sqlFolder = "testdata/name/sql/sql_server",
  cohortFileNameFormat = "%s",
  cohortFileNameValue = c("cohortName"),
  packageName = "CohortGenerator",
  verbose = FALSE
)

print(initialSet[, c("cohortId", "cohortName")])

# Subset cohorts 1 & 2 by an "indication" cohort 3:
res <- addExcludeOnIndexSubsetDefinition(
  cohortDefinitionSet = initialSet,
  targetCohortIds = c(1, 2),
  exclusionCohortIds = c(3),
  subsetDefinitionId = 20,
  subsetDefinitioName = "Exclude on index if in cohort 3"
)

print(res[, c("cohortId", "cohortName", "subsetParent", "subsetDefinitionId", "isSubset")])

# Get all subset definitions that were created using the addExcludeOnIndexSubsetDefinition:
subsetDefinitionId <- getExcludeOnIndexSubsetDefinitionIds(res)

# Filter the cohortDefinitionSet to those cohorts defined using an exclusion subset definition:
newCohorts <- res |>
  dplyr::filter(subsetDefinitionId == subsetDefinitionId) |>
  dplyr::select(cohortId, cohortName, subsetParent, isSubset)
print(newCohorts)

## End(Not run)
```

`addIndicationSubsetDefinition` *Add Indication Subset Definition*

Description

Utility pattern for creating an indication subset from a set of target cohorts. The approach applies this subset definition to an exposure (target cohort) or set of exposures (multiple target cohorts), requiring the individual to have a history of the indication cohort overlapping the start of the first exposure. The first exposure must have the 'requiredPriorObservationTime' and 'requiredFollowUpTime'. If specified, the first exposure must also fall within the 'studyStartDate' and 'studyEndDate' and also meet the age and gender criteria.

Additionally, the R attribute of "indicationSubsetDefinitions" is attached to the cohort definition set. This can be obtained by calling 'getIndicationSubsetDefinitionIds', which should return the set of subset definition ids that are associated with indications.

Usage

```
addIndicationSubsetDefinition(
  cohortDefinitionSet,
  targetCohortIds,
  indicationCohortIds,
  subsetDefinitionId,
  subsetDefinitionName,
  subsetCohortNameTemplate = "@baseCohortName - @subsetDefinitionName",
  cohortCombinationOperator = "any",
  lookbackWindowStart = -99999,
  lookbackWindowEnd = 0,
  lookForwardWindowStart = 0,
  lookForwardWindowEnd = 99999,
  genderConceptIds = NULL,
  ageMin = NULL,
  ageMax = NULL,
  studyStartDate = NULL,
  studyEndDate = NULL,
  requiredPriorObservationTime = 365,
  requiredFollowUpTime = 1
)
```

Arguments

`cohortDefinitionSet`

The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`targetCohortIds`
: Set of integer cohort IDs. Must be within the cohort definition set.

`indicationCohortIds`
: Set of integer cohort IDs. Must be within the cohort definition set.

`subsetDefinitionId`
: Unique integer Id of the subset definition

`subsetDefinitionName`
: name of the subset definition (used in resulting cohort definitions)

`subsetCohortNameTemplate`
: template string format for naming resulting cohorts

`cohortCombinationOperator`
: Logic for multiple indication cohort IDs: any (default) or all.

`lookbackWindowStart`
: Start of lookback period.

`lookbackWindowEnd`
: End of lookback period.

`lookForwardWindowStart`
: When the indication can end relative to index; default is 0.

`lookForwardWindowEnd`
: When the indication can end relative to index; default is 9999.

`genderConceptIds`
: Gender concepts to require

`ageMin` Minimum age at target index.

`ageMax` Maximum age at target index.

`studyStartDate` Exclude patients with index prior to this date (format "%Y%m%d").

`studyEndDate` Exclude patients with index after this date (format "%Y%m%d").

`requiredPriorObservationTime`
: Observation time prior to index; default 365.

`requiredFollowUpTime`
: Observation time after index; default 1.

Examples

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

initialSet <- getCohortDefinitionSet(
  settingsFileName = "testdata/name/Cohorts.csv",
  jsonFolder = "testdata/name/cohorts",
  sqlFolder = "testdata/name/sql/sql_server",
  cohortFileNameFormat = "%s",
  cohortFileNameValue = c("cohortName"),
```

```
  packageName = "CohortGenerator",
  verbose = FALSE
)

print(initialSet[, c("cohortId", "cohortName")])

# Subset cohorts 1 & 2 by an "indication" cohort 3:
res <- addIndicationSubsetDefinition(
  cohortDefinitionSet = initialSet,
  targetCohortIds = c(1, 2),
  indicationCohortIds = c(3),
  subsetDefinitionId = 10
)

print(res[, c("cohortId", "cohortName", "subsetParent", "subsetDefinitionId", "isSubset")])

# Get all subset definitions that were created using the addIndicationSubsetDefinition:
subsetDefinitionId <- getIndicationSubsetDefinitionIds(res)

# Filter the cohortDefinitionSet to those cohorts defined using an indication subset definition:
newCohorts <- res |>
  dplyr::filter(subsetDefinitionId == subsetDefinitionId) |>
  dplyr::select(cohortId, cohortName, subsetParent, isSubset)
print(newCohorts)

## End(Not run)
```

`addRestrictionSubsetDefinition` *Add Restriction Subset Definition*

Description

Utility pattern for creating cohort subset definitions as a standard approach for indicated drugs. Restriction subset definitions are twins of indication definitions. They should apply the same core properties to a base exposure cohort (i.e. study dates, required prior observation time, ages, gender) as indications but, crucially, they do not require history of any prior condition(s).

This is useful in the context of comparing drug exposure + indication population, to population as a whole.

The preferred use of this function is to create this in conjunction with the target population.

Usage

```
addRestrictionSubsetDefinition(
  cohortDefinitionSet,
  targetCohortIds,
  subsetDefinitionId,
```

```
  subsetDefinitionName,
  subsetCohortNameTemplate = "@baseCohortName - @subsetDefinitionName",
  genderConceptIds = NULL,
  ageMin = NULL,
  ageMax = NULL,
  studyStartDate = NULL,
  studyEndDate = NULL,
  requiredPriorObservationTime = 365,
  requiredFollowUpTime = 1
)
```

Arguments

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  sql The OHDSI-SQL used to generate the cohort

  Optionally, this data frame may contain:

  json The Circe JSON representation of the cohort

`targetCohortIds`
: Set of integer cohort IDs. Must be within the cohort definition set.

`subsetDefinitionId`
: Unique integer Id of the subset definition

`subsetDefinitionName`
: name of the subset definition (used in resulting cohort definitions)

`subsetCohortNameTemplate`
: template string format for naming resulting cohorts

`genderConceptIds`
: Gender concepts to require

`ageMin`
: Minimum age at target index.

`ageMax`
: Maximum age at target index.

`studyStartDate`
: Exclude patients with index prior to this date (format "%Y%m%d").

`studyEndDate`
: Exclude patients with index after this date (format "%Y%m%d").

`requiredPriorObservationTime`
: Observation time prior to index; default 365.

`requiredFollowUpTime`
: Observation time after index; default 1.

Examples

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

```
initialSet <- getCohortDefinitionSet(
  settingsFileName = "testdata/name/Cohorts.csv",
  jsonFolder = "testdata/name/cohorts",
  sqlFolder = "testdata/name/sql/sql_server",
  cohortFileNameFormat = "%s",
  cohortFileNameValue = c("cohortName"),
  packageName = "CohortGenerator",
  verbose = FALSE
)

print(initialSet[, c("cohortId", "cohortName")])

# Restrinct to first occurrence of cohort
res <- addRestrictionSubsetDefinition(
  cohortDefinitionSet = initialSet,
  targetCohortIds = c(1, 2),
  subsetDefinitionId = 20
)

print(res[, c("cohortId", "cohortName", "subsetParent", "subsetDefinitionId", "isSubset")])

# Get all subset definitions that were created using the addRestrictionSubsetDefinition:
subsetDefinitionId <- getRestrictionSubsetDefinitionIds(res)

# Filter the cohortDefinitionSet to those cohorts defined using an restriction subset definition:
newCohorts <- res |>
  dplyr::filter(subsetDefinitionId == subsetDefinitionId) |>
  dplyr::select(cohortId, cohortName, subsetParent, isSubset)
print(newCohorts)

## End(Not run)
```

`addSqlCohortDefinition` *Add an sql cohort definition*

Description

This is useful in cases where it is difficult or impossible to define a cohort in Circe. This utility should be used sparingly, but is convenient non-the-less. Note that no checks on this definition occur and, in principle, any sql can be executed. Incremental execution and logging will work. This should also be compatible with other OHDSI packages that use standard cohort tables.

All cohorts should result in standard cohort tables which have the columns:

* cohort_definition_id, * subject_id, * cohort_start_date, * cohort_end_date

As these are requirements of cohorts.

The sql parameters: cohort_table, cohort_database_schema, cdm_database_schema and vocabulary_database_schema should not be specified in the arguments to this function. These cohorts can be serialized with saveCohortDefinitionSet and shared so should not include data source specific content.

Usage

```
addSqlCohortDefinition(
  cohortDefinitionSet,
  sql,
  cohortId,
  cohortName,
  tanslateSql = TRUE,
  json = NULL,
  ...
)
```

Arguments

cohortDefinitionSet	The cohortDefinitionSet argument must be a data frame with the following columns: **cohortId** The unique integer identifier of the cohort **cohortName** The cohort's name **sql** The OHDSI-SQL used to generate the cohort Optionally, this data frame may contain: **json** The Circe JSON representation of the cohort
sql	OHDSI SqlRender-compatible sql
cohortId	Id of cohort to add. Must be unique in the cohort definition set
cohortName	Name of the cohort to add
tanslateSql	perform translation on the sql. This is ignored if the sql has already been translated with the sql render function.
json	optional json parameters
...	arguments for the sql. Note that this does not need to include cohort_table, cohort_database_schema, cdm_database_schema or vocabulary_database_schema

Examples

```
sql <- "INSERT INTO @cohort_database_schema.@cohort_table
             (cohort_definition_id, subject_id, cohort_start_date, cohort_end_date)
        SELECT 1 as cohort_definition_id,
               person_id as subject_id,
               drug_era_start_date as cohort_start_date,
               drug_era_end_data as cohort_end_date
        FROM @cdm_database_schema.drug_era de
     INNER JOIN @vocabulary_database_schema.concept c on de.drug_concept_id = c.concept_id
        -- Find any matches of drugs named 'asprin' in the drug concept table
        WHERE lower(c.concept_name) like '%asprin%'; "

cohortDefinitionSet <- createEmptyCohortDefinitionSet() |>
  addSqlCohortDefinition(sql = sql, cohortId = 1, cohortName = "my asprin cohort")
```

addUnionCohortDefinition *Add union cohort definition to cohort definition set*

Description

This utility function adds the union of any two or more cohort ids to the cohort definition set with a new id and name.

If a name parameter is not provided this will be auto generated as the union of the provided cohort id

Usage

```
addUnionCohortDefinition(
  cohortDefinitionSet,
  cohortIds,
  cohortName,
  unionCohortId
)
```

Arguments

cohortDefinitionSet	cohort definition set
cohortIds	A vector of 'cohort_definition_id' values for the input cohorts.
cohortName	The Name of the resulting cohort
unionCohortId	The 'cohort_definition_id' for the resulting union cohort.

checkAndFixCohortDefinitionSetDataTypes *Check if a cohort definition set is using the proper data types*

Description

This function checks a data.frame to verify it holds the expected format for a cohortDefinitionSet's data types and can optionally fix data types that do not match the specification.

Usage

```
checkAndFixCohortDefinitionSetDataTypes(
  x,
  fixDataTypes = TRUE,
  emitWarning = FALSE
)
```

Arguments

x	The cohortDefinitionSet data.frame to check
fixDataTypes	When TRUE, this function will attempt to fix the data types to match the specification. @seealso [createEmptyCohortDefinitionSet()].
emitWarning	When TRUE, this function will emit warning messages when problems are encountered.

Value

Returns a list() of the following form:

list(dataTypesMatch = TRUE/FALSE, x = data.frame())

dataTypesMatch == TRUE when the supplied data.frame x matches the cohortDefinitionSet specification's data types.

If fixDataTypes == TRUE, x will hold the original data from x with the data types corrected. Otherwise x will hold the original value passed to this function.

CohortSubsetDefinition

Cohort Subset Definition

Description

Set of subset definitions pretty in print

Active bindings

`targetOutputPairs` list of pairs of integers - (targetCohortId, outputCohortId)

`subsetOperators` list of subset operations

`name` name of definition

`subsetCohortNameTemplate` template string for formatting resulting cohort names

`operatorNameConcatString` string used when concatenating operator names together

`definitionId` numeric definition id

`identifierExpression` expression that can be evaluated from

Methods

Public methods:

- `CohortSubsetDefinition$print()`
- `CohortSubsetDefinition$new()`
- `CohortSubsetDefinition$toList()`
- `CohortSubsetDefinition$toJSON()`
- `CohortSubsetDefinition$addSubsetOperator()`

- CohortSubsetDefinition$getSubsetQuery()
- CohortSubsetDefinition$getSubsetCohortName()
- CohortSubsetDefinition$setTargetOutputPairs()
- CohortSubsetDefinition$getJsonFileName()
- CohortSubsetDefinition$clone()

`CohortSubsetDefinition$print()`:

Usage:

```
CohortSubsetDefinition$print(...)
```

Arguments:

`...` further arguments passed to or from other methods.

`CohortSubsetDefinition$new()`:

Usage:

```
CohortSubsetDefinition$new(definition = NULL)
```

Arguments:

`definition` json or list representation of object to List

`CohortSubsetDefinition$toList()`: List representation of object to JSON

Usage:

```
CohortSubsetDefinition$toList()
```

`CohortSubsetDefinition$toJSON()`: json serialized representation of object add Subset Operator

Usage:

```
CohortSubsetDefinition$toJSON()
```

`CohortSubsetDefinition$addSubsetOperator()`: add subset to class - checks if equivalent id is present Will throw an error if a matching ID is found but reference object is different

Usage:

```
CohortSubsetDefinition$addSubsetOperator(subsetOperator)
```

Arguments:

`subsetOperator` a SubsetOperator instance

`overwrite` if a subset operator of the same ID is present, replace it with a new definition get query for a given target output pair

`CohortSubsetDefinition$getSubsetQuery()`: Returns vector of join, logic, having statements returned by subset operations

Usage:

```
CohortSubsetDefinition$getSubsetQuery(targetOutputPair)
```

Arguments:

`targetOutputPair` Target output pair Get name of an output cohort

`CohortSubsetDefinition$getSubsetCohortName()`:

Usage:

```
CohortSubsetDefinition$getSubsetCohortName(
  cohortDefinitionSet,
  targetOutputPair
)
```

Arguments:

`cohortDefinitionSet` Cohort definition set containing base names

`targetOutputPair` Target output pair Set the targetOutputPairs to be added to a cohort definition set

`CohortSubsetDefinition$setTargetOutputPairs()`:

Usage:

```
CohortSubsetDefinition$setTargetOutputPairs(targetIds)
```

Arguments:

`targetIds` list of cohort ids to apply subsetting operations to Get json file name for subset definition in folder

`CohortSubsetDefinition$getJsonFileName()`:

Usage:

```
CohortSubsetDefinition$getJsonFileName(
  subsetJsonFolder = "inst/cohort_subset_definitions/"
)
```

Arguments:

`subsetJsonFolder` path to folder to place file

`CohortSubsetDefinition$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CohortSubsetDefinition$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CohortSubsetOperator` *Cohort Subset Operator*

Description

A subset of type cohort - subset a population to only those contained within defined cohort

Super class

`SubsetOperator` -> `CohortSubsetOperator`

Active bindings

`cohortIds` Integer ids of cohorts to subset to

`cohortCombinationOperator` How to combine the cohorts

`negate` Inverse the subset rule? TRUE will take the patients NOT in the subset

`windows` list of time windows to use when evaluating the subset cohort relative to the target cohort

Methods

Public methods:

- `CohortSubsetOperator$new()`
- `CohortSubsetOperator$toList()`
- `CohortSubsetOperator$getAutoGeneratedName()`
- `CohortSubsetOperator$clone()`

`CohortSubsetOperator$new()`:

Usage:

```
CohortSubsetOperator$new(definition = NULL)
```

Arguments:

`definition` json character or list - definition of subset operator

Returns: instance of object to List

`CohortSubsetOperator$toList()`: List representation of object Get auto generated name

Usage:

```
CohortSubsetOperator$toList()
```

`CohortSubsetOperator$getAutoGeneratedName()`: name generated from subset operation properties

Usage:

```
CohortSubsetOperator$getAutoGeneratedName()
```

Returns: character

`CohortSubsetOperator$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CohortSubsetOperator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CohortTemplateDefinition` *Class for automating the creation of bulk cohorts*

Description

This class provides a framework for automating the creation of bulk cohorts by defining template SQL queries and associated callbacks to execute them. This is useful when defining lots of exposure or outcomes for cohorts that are very general in nature. For example, all RxNorm ingredient cohorts, all ATC ingredient cohorts or all SNOMED condition occurrences with > x diagnosis codes.

These cohorts can then be subsetted with common cohort subset operations such as limiting to specific age, gender, or observation criteria, should this be excluded from the cohort definition. However, when applying operations in bulk it may be more efficient to include such definitions within the template sql itself.

This approach is also useful for cohorts that can not based on ATLAS/CirceDefinitions alone.

CURRENTLY NOT SUPPORTED: * Saving definitions that use runtime arguments on a per cdm basis. This creates a challenge for running the same cohort across different databases. Furthermore, saving information within the CDM schema in a shared OHDSI study is not desirable.

Active bindings

`name` name for this template definition that describes the cohorts it creation

`sqlArgs` optional arguments for sql

`templateSql` sql template

`translateSql` translate the sql for different platforms

`references` data.frame of name/id references for cohort template that aligns with cohort set

Methods

Public methods:

- `CohortTemplateDefinition$new()`
- `CohortTemplateDefinition$executeTemplateSql()`
- `CohortTemplateDefinition$getTemplateReferences()`
- `CohortTemplateDefinition$getName()`
- `CohortTemplateDefinition$getId()`
- `CohortTemplateDefinition$getChecksum()`
- `CohortTemplateDefinition$toList()`
- `CohortTemplateDefinition$toJson()`
- `CohortTemplateDefinition$saveTemplate()`
- `CohortTemplateDefinition$clone()`

`CohortTemplateDefinition$new()`:

Usage:

```
CohortTemplateDefinition$new(settings)
```

Arguments:

`settings` Settings of object to load seealso createCohortTemplateDefinition To alter the execution, override this function in a subclass. This translates and executes the sql of the cohort definition Note that calling this function will generate the cohorts but will not do so in an incremental manner. Checksums and timestamps will, however, be added to the generation table ever want to call this function outside of a testing environment. It is best practice to always use the standard runCohortGeneration/generateCohortSet pipeline to ensure validity of execution steps.

`CohortTemplateDefinition$executeTemplateSql()`:

Usage:

```
CohortTemplateDefinition$executeTemplateSql(
  connection,
  cohortDatabaseSchema,
  cdmDatabaseSchema,
  cohortTableNames,
  vocabularyDatabaseSchema = cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema")
)
```

Arguments:

`connection` An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`cohortDatabaseSchema` Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortDatabaseSchema` Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cdmDatabaseSchema` Schema name where your patient-level data in OMOP CDM format resides. Note that for SQL Server, this should include both the database and schema name, for example 'cdm_data.dbo'.

`cohortTableNames` The names of the cohort tables. See `getCohortTableNames` for more details.

`vocabularyDatabaseSchema` vocabulary database schema

`tempEmulationSchema` cdm temp emulation schema get template references data.frame

`CohortTemplateDefinition$getTemplateReferences()`: Returns data.frame of references get the name of the definition

Usage:

```
CohortTemplateDefinition$getTemplateReferences()
```

`CohortTemplateDefinition$getName()`: Name field get the generated id of the template definition

Usage:

```
CohortTemplateDefinition$getName()
```

`CohortTemplateDefinition$getId()`: this is not the cohort ids and is based off of the checksum of the template definition get checksum

Usage:

```
CohortTemplateDefinition$getId()
```

`CohortTemplateDefinition$getChecksum()`: Get the hash of the definition (generated when class is instantiated) to list

Usage:

```
CohortTemplateDefinition$getChecksum()
```

`CohortTemplateDefinition$toList()`: Used for serializing the definition to json

Usage:

```
CohortTemplateDefinition$toList()
```

`CohortTemplateDefinition$toJson()`: json serialized form of the template definition save to disk

Usage:

```
CohortTemplateDefinition$toJson()
```

`CohortTemplateDefinition$saveTemplate()`: Save object to specified json path

Usage:

```
CohortTemplateDefinition$saveTemplate(filePath)
```

Arguments:

`filePath` File path to save json serialized from

`CohortTemplateDefinition$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CohortTemplateDefinition$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`computeChecksum` *Computes the checksum for a value*

Description

This is used as part of the incremental operations to hash a value to store in a record keeping file. This function leverages the md5 hash from the digest package

Usage

```
computeChecksum(val)
```

Arguments

val	The value to hash. It is converted to a character to perform the hash.

Value

Returns a string containing the checksum

---

computeCohortAttrition

Compute cohort attrition from inclusion rule statistics

Description

Computes a sequential attrition table using the inclusion rule statistics stored in the cohort statistics tables for Circe-based cohorts. For each cohort definition, we report a base cohort entry count (before inclusion rules) and then counts after applying the first k inclusion rules in sequence.

Inclusion rule satisfaction is encoded as a bit mask in `inclusionRuleMask`. For a rule sequence `i`, its bit value is `2^i`. A row with `inclusionRuleMask` equal to the sum of the bits indicates which rules were met. To compute the count after the first `k` rules, we require all first-`k` bits to be set by checking `bitwAnd(inclusionRuleMask, requiredMask) == requiredMask`, where `requiredMask = 2^k - 1`.

Attrition is computed separately for each `modeId` present in `cohortInclusionResult` (for example, person-level and event-level).

Usage

```
computeCohortAttrition(cohortInclusionResult, cohortInclusion)
```

Arguments

cohortInclusionResult	A data.frame containing inclusion rule masks and counts, typically from the `cohortInclusionResultTable` with camelCase column names. Required columns: `databaseId`, `cohortDefinitionId`, `inclusionRuleMask`, `modeId`, `personCount`. You can obtain this via `getCohortStats(..., outputTables = "cohortInclusionResultTable")` or by querying the cohort results schema table created when stats are generated.
cohortInclusion	A data.frame of inclusion rule metadata, typically from `cohortInclusionTable` with camelCase column names. Required columns: `cohortDefinitionId`, `ruleSequence`. You can obtain this via `getCohortStats(..., outputTables = "cohortInclusionTable")` or by querying the cohort results schema table created when stats are generated.

Value

A data.frame with the following columns:

- `databaseId`: Database identifier.
- `cohortDefinitionId`: Cohort definition identifier.
- `modeId`: The mode identifier from `cohortInclusionResult`.
- `cohortEntry`: 1 for the base cohort entry count, 0 for rule rows.
- `ruleSequence`: Inclusion rule sequence (-1 for base row).
- `personCount`: Count after applying rules.

`createAtcCohortTemplateDefinition`

Create ATC Cohort Template Definition

Description

Template cohort definition for all ATC level 4 class exposures. The cohortId = conceptId * 1000 + 4. The "identifierExpression" can be customized for uniqueness.

Usage

```
createAtcCohortTemplateDefinition(
  connection,
  identifierExpression = "CAST(concept_id as bigint) * 1000",
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema,
  nameSuffix = "",
  mergeIngredientEras = TRUE,
  priorObservationPeriod = 365,
  vocabularyDatabaseSchema = cdmDatabaseSchema
)
```

Arguments

`connection`	Database connection object
`identifierExpression`	An expression for setting the cohort id for the resulting cohort. Must produce unique ids
`cdmDatabaseSchema`	CDM database schema
`tempEmulationSchema`	Temporary emulation schema
`cohortDatabaseSchema`	Cohort database schema

`nameSuffix` A name suffix to use to add to the cohort names - this is useful if you're using multiple parameterized versions of this definition

`mergeIngredientEras`
(optional) Boolean indicating if different ingredients under the same ATC code should be merged

`priorObservationPeriod`
(optional) Required prior observation period for individuals

`vocabularyDatabaseSchema`
Vocabulary database schema

Value

A CohortTemplateDefinition instance

`createCohortSubset` *Create Cohort Subset Operator*

Description

Subset cohorts using specified limit criteria. deprecated This function is deprecated. Please use 'createCohortSubsetOperator()' instead.

Usage

```
createCohortSubset(...)
```

Arguments

`...` Arguments passed to the underlying operator.

`createCohortSubsetDefinition`
Create Subset Definition

Description

Create subset definition from subset objects

Usage

```
createCohortSubsetDefinition(
  name,
  definitionId,
  subsetOperators,
  identifierExpression = NULL,
  subsetCohortNameTemplate = "@baseCohortName - @subsetDefinitionName"
)
```

Arguments

name	Name of definition
definitionId	Definition identifier
subsetOperators	list of subsetOperator instances to apply
identifierExpression	Expression (or string that converts to expression) that returns an id for an output cohort the default is dplyr::expr(targetId * 1000 + definitionId)
subsetCohortNameTemplate	SqlRender string template for formatting names of resulting subset cohorts Can use the variables @baseCohortName and @subsetDefinitionName. This is applied when adding the subset definition to a cohort definition set.

---

createCohortSubsetOperator *A definition of subset functions to be applied to a set of cohorts*

Description

A definition of subset functions to be applied to a set of cohorts

Usage

```
createCohortSubsetOperator(
  name = NULL,
  cohortIds,
  cohortCombinationOperator,
  negate,
  windows = list(),
  startWindow = NULL,
  endWindow = NULL
)
```

Arguments

name	optional name of operator
cohortIds	integer - set of cohort ids to subset to
cohortCombinationOperator	"any" or "all" if using more than one cohort id allow a subject to be in any cohort or require that they are in all cohorts in specified windows
negate	The opposite of this definition - include patients who do NOT meet the specified criteria
windows	A list of time windows to use to evaluate subset cohorts in relation to the target cohorts. The logic is to always apply these windows with logical AND conditions. See [@seealso [createSubsetCohortWindow()]] for more details on how to create these windows.

startWindow	DEPRECATED: Use 'windows' instead.
endWindow	DEPRECATED: Use 'windows' instead.

Value

a CohortSubsetOperator instance

See Also

Other subsets: createDemographicSubsetOperator(), createLimitSubsetOperator()

---

createCohortTables *Create cohort tables*

Description

This function creates an empty cohort table and empty tables for cohort statistics.

Usage

```
createCohortTables(
  connectionDetails = NULL,
  connection = NULL,
  cohortDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  incremental = FALSE
)
```

Arguments

connectionDetails	An object of type connectionDetails as created using the createConnectionDetails function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the connect function in the DatabaseConnector package. Can be left NULL if connectionDetails is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.
cohortDatabaseSchema	Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.
cohortTableNames	The names of the cohort tables. See getCohortTableNames for more details.
incremental	When set to TRUE, this function will check to see if the cohortTableNames exists in the cohortDatabaseSchema and if they exist, it will skip creating the tables.

createCohortTemplateDefintion

Create Cohort Template Definition

Description

construct a cohort template definition

Usage

```
createCohortTemplateDefintion(
  name,
  templateSql,
  references,
  sqlArgs = list(),
  translateSql = TRUE
)
```

Arguments

name	A name for the template definition. This is not used in the checksum of the cohort
templateSql	Sql string that is used to generate the cohorts. This should be in OHDSI sql form, translatable to other db platforms.
references	This is a data frame that must contain cohortId and cohortName. Optionally, this can contain the columns sql and json as well. It must be bindable to a cohort definition set instance.
sqlArgs	Optional parameters for execution of the query - for example vocabulary schema These are arguments that should be passed to the sql. These are used in the checksum if using parameterized sql for different definitions (e.g. a definition requiring varying observation lengths. This is used to distinguish them) This should not include cdm/data source specific parameters such as the cohort table names, cdm database schema or vocabulary database schema. If the definition requires runtime specific arguments (e.g. non standard tables) this presents a problem for serializing and uniquely identifying template cohort definitions.
translateSql	to translate the sql or not.

createDemographicSubset

Create Demographic Subset Operator

Description

Subset cohorts using specified limit criteria. deprecated This function is deprecated. Please use ‘createDemographicSubsetOperator()‘ instead.

Usage

```
createDemographicSubset(...)
```

Arguments

...	Arguments passed to the underlying operator.

---

createDemographicSubsetOperator *Create createDemographicSubset Subset operator*

Description

Create createDemographicSubset Subset operator

Usage

```
createDemographicSubsetOperator(
  name = NULL,
  ageMin = 0,
  ageMax = 99999,
  gender = NULL,
  race = NULL,
  ethnicity = NULL
)
```

Arguments

name	Optional char name
ageMin	The minimum age
ageMax	The maximum age
gender	Gender demographics - concepts - 0, 8532, 8507, 0, "female", "male". Any string that is not "male" or "female" (case insensitive) is converted to gender concept 0. https://athena.ohdsi.org/search-terms/terms?standardConcept=Standard&domain=Gender&page= Specific concept ids not in this set can be used but are not explicitly validated
race	Race demographics - concept ID list
ethnicity	Ethnicity demographics - concept ID list

See Also

Other subsets: createCohortSubsetOperator(), createLimitSubsetOperator()

createEmptyCohortDefinitionSet

Create an empty cohort definition set

Description

This function creates an empty cohort set data.frame for use with `generateCohortSet`.

Usage

```
createEmptyCohortDefinitionSet(verbose = FALSE)
```

Arguments

`verbose`	When TRUE, descriptions of each field in the data.frame are returned

Value

Invisibly returns an empty cohort set data.frame

createEmptyNegativeControlOutcomeCohortSet

Create an empty negative control outcome cohort set

Description

This function creates an empty cohort set data.frame for use with `generateNegativeControlOutcomeCohorts`.

Usage

```
createEmptyNegativeControlOutcomeCohortSet(verbose = FALSE)
```

Arguments

`verbose`	When TRUE, descriptions of each field in the data.frame are returned

Value

Invisibly returns an empty negative control outcome cohort set data.frame

createLimitSubset *Create Limit Subset Operator*

Description

Subset cohorts using specified limit criteria. deprecated This function is deprecated. Please use ‘createLimitSubsetOperator()‘ instead.

Usage

```
createLimitSubset(...)
```

Arguments

...	Arguments passed to the underlying operator.

createLimitSubsetOperator *Create Limit Subset Operator*

Description

Subset cohorts using specified limit criteria

Usage

```
createLimitSubsetOperator(
  name = NULL,
  priorTime = 0,
  followUpTime = 0,
  minimumCohortDuration = 0,
  maximumCohortDuration = NULL,
  limitTo = "all",
  calendarStartDate = NULL,
  calendarEndDate = NULL
)
```

Arguments

name	Name of operation
priorTime	Required prior observation window (specified as a positive integer)
followUpTime	Required post observation window (specified as a positive integer)
minimumCohortDuration	Required cohort duration length (specified as a positive integer)

maximumCohortDuration
: Optional: maximum cohort duration length (specified as a positive integer), defaults to NULL

limitTo
: character one of: "firstEver" - only first entry in patient history "earliestRemaining" - only first entry after washout set by priorTime "latestRemaining" - the latest remaining after washout set by followUpTime "lastEver" - only last entry in patient history inside

  Note, when using firstEver and lastEver with follow up and washout, patients with events outside this will be censored. The "firstEver" and "lastEver" are applied first. The "earliestRemaining" and "latestRemaining" are applied after all other limit criteria are applied (i.e. after applying prior/post time and calendar time).

calendarStartDate
: End date to allow periods (e.g. 2020/1/1/)

calendarEndDate
: Start date to allow period (e.g. 2015/1/1)

See Also

Other subsets: `createCohortSubsetOperator()`, `createDemographicSubsetOperator()`

---

createResultsDataModel

Create the results data model tables on a database server.

Description

Create the results data model tables on a database server.

Usage

```
createResultsDataModel(
  connectionDetails = NULL,
  databaseSchema,
  tablePrefix = ""
)
```

Arguments

connectionDetails
: DatabaseConnector connectionDetails instance @seealso[DatabaseConnector::createConnectionDetails]

databaseSchema
: The schema on the server where the tables will be created.

tablePrefix
: (Optional) string to insert before table names for database table names

Details

Only PostgreSQL and SQLite servers are supported.

createRxNormCohortTemplateDefinition
Create Rx Norm Cohort Template Definition

Description

Template cohort definition for all RxNorm ingredients. This cohort will use the vocabulary tables to automatically generate a set of cohorts that have the cohortId = conceptId * 1000. The "identifierExpression" can be customized for uniqueness.

Usage

```
createRxNormCohortTemplateDefinition(
  connection,
  identifierExpression = "CAST(concept_id as bigint) * 1000",
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema,
  priorObservationPeriod = 365,
  nameSuffix = "",
  vocabularyDatabaseSchema = cdmDatabaseSchema
)
```

Arguments

connection	Database connection object
identifierExpression	An expression for setting the cohort id for the resulting cohort. Must produce unique ids
cdmDatabaseSchema	CDM database schema
tempEmulationSchema	Temporary emulation schema
cohortDatabaseSchema	Cohort database schema
priorObservationPeriod	(optional) Required prior observation period for individuals
nameSuffix	A name suffix to use to add to the cohort names - this is useful if you're using multiple parameterized versions of this definition
vocabularyDatabaseSchema	Vocabulary database schema

Value

A CohortTemplateDefinition instance

`createSnomedCohortTemplateDefinition` *Create SNOMED Cohort Template Definition*

Description

Template cohort definition for all OHDSI standard conditions. The cohortId = conceptId * 1000. The "identifierExpression" can be customized for uniqueness. This definition uses any valid SNOMED condition code and all its descendants.

Excluded terms include word patterns:
, ,

Cohorts are first event.

Usage

```
createSnomedCohortTemplateDefinition(
  connection,
  identifierExpression = "CAST(concept_id as bigint) * 1000",
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  priorObservationPeriod = 365,
  requireSecondDiagnosis = FALSE,
  nameSuffix = "",
  vocabularyDatabaseSchema = cdmDatabaseSchema
)
```

Arguments

`connection`	Database connection object
`identifierExpression`	An expression for setting the cohort id for the resulting cohort. Must produce unique ids
`cdmDatabaseSchema`	CDM database schema
`tempEmulationSchema`	Temporary emulation schema
`priorObservationPeriod`	(optional) Required prior observation period for individuals
`requireSecondDiagnosis`	(optional) Require more than one diagnosis code
`nameSuffix`	A name suffix to use to add to the cohort names - this is useful if you're using multiple parameterized versions of this definition
`vocabularyDatabaseSchema`	Vocabulary database schema

Value

A CohortTemplateDefinition instance

`createSubsetCohortWindow` *Create a relative time window for cohort subset operations*

Description

This function is used to create a relative time window for cohort subset operations. The cohort window allows you to define an interval of time relative to the target cohort's start/end date and the subset cohort's start/end date.

Usage

```
createSubsetCohortWindow(
  startDay,
  endDay,
  targetAnchor,
  subsetAnchor = NULL,
  negate = FALSE
)
```

Arguments

`startDay`	The start day for the time window
`endDay`	The end day for the time window
`targetAnchor`	To anchor using the target cohort's start date or end date. The parameter is specified as 'cohortStart' or 'cohortEnd'.
`subsetAnchor`	To anchor using the subset cohort's start date or end date. The parameter is specified as 'cohortStart' or 'cohortEnd'.
`negate`	Allows for negating a window, a way to detect that a subset does not occur relative to a target

Value

a SubsetCohortWindow instance

`createUnionCohortTemplate`

Create cohort template to union multiple cohorts

Description

This is a union between all cohorts within a specified set of ids. If an individual has multiple overlapping eras, they will be merged into a single time window.

Distinct eras will be mapped to the same cohort id but remain distinct. For example:

“‘ A: [———] B: [–] C: [——-] “‘ Becomes: “‘ A U B U C: [————] “‘

And “‘ A: [———] B: [——-] “‘ Becomes “‘ A U B: [———] [——-] “‘ It is never allowed to have multiple overlapping eras for the same individual within a cohort

Usage

```
createUnionCohortTemplate(cohortIds, cohortName, unionCohortId)
```

Arguments

`cohortIds`	A vector of ‘cohort_definition_id‘ values for the input cohorts.
`cohortName`	The Name of the resulting cohort
`unionCohortId`	The ‘cohort_definition_id‘ for the resulting union cohort.

`DemographicSubsetOperator`

Demographic Subset Operator

Description

Operators for subsetting a cohort by demographic criteria

Super class

`SubsetOperator` -> `DemographicSubsetOperator`

Active bindings

`ageMin` Int between 0 and 99999 - minimum age

`ageMax` Int between 0 and 99999 - maximum age

`gender` vector of gender concept IDs

`race` character string denoting race

`ethnicity` character string denoting ethnicity

Methods

Public methods:

- DemographicSubsetOperator$toList()
- DemographicSubsetOperator$mapGenderConceptsToNames()
- DemographicSubsetOperator$getAutoGeneratedName()
- DemographicSubsetOperator$toJSON()
- DemographicSubsetOperator$isEqualTo()
- DemographicSubsetOperator$getGender()
- DemographicSubsetOperator$getRace()
- DemographicSubsetOperator$getEthnicity()
- DemographicSubsetOperator$clone()

`DemographicSubsetOperator$toList()`: List representation of object Map gender concepts to names

Usage:

```
DemographicSubsetOperator$toList()
```

`DemographicSubsetOperator$mapGenderConceptsToNames()`:

Usage:

```
DemographicSubsetOperator$mapGenderConceptsToNames(
  mapping = list(`8507` = "males", `8532` = "females", `0` = "unknown gender")
)
```

Arguments:

`mapping` optional list of mappings for concept id to nouns

Returns: char vector Get auto generated name

`DemographicSubsetOperator$getAutoGeneratedName()`: name generated from subset operation properties

Usage:

```
DemographicSubsetOperator$getAutoGeneratedName()
```

Returns: character

`DemographicSubsetOperator$toJSON()`: json serialized representation of object

Usage:

```
DemographicSubsetOperator$toJSON()
```

`DemographicSubsetOperator$isEqualTo()`: Compare Subset to another

Usage:

```
DemographicSubsetOperator$isEqualTo(criteria)
```

Arguments:

`criteria` DemographicSubsetOperator instance

`DemographicSubsetOperator$getGender()`: Gender getter - used when constructing SQL to default NULL to an empty string

Usage:

```
DemographicSubsetOperator$getGender()
```

`DemographicSubsetOperator$getRace()`: Race getter - used when constructing SQL to default NULL to an empty string

Usage:

```
DemographicSubsetOperator$getRace()
```

`DemographicSubsetOperator$getEthnicity()`: Ethnicity getter - used when constructing SQL to default NULL to an empty string

Usage:

```
DemographicSubsetOperator$getEthnicity()
```

`DemographicSubsetOperator$clone()`: The objects of this class are cloneable with this method.

Usage:

```
DemographicSubsetOperator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`dropCohortStatsTables` *Drop cohort statistics tables*

Description

This function drops the cohort statistics tables.

Usage

```
dropCohortStatsTables(
  connectionDetails = NULL,
  connection = NULL,
  cohortDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  dropCohortTable = FALSE
)
```

Arguments

`connectionDetails`	An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.
`connection`	An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

cohortDatabaseSchema
: Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

cohortTableNames
: The names of the cohort tables. See `getCohortTableNames` for more details.

dropCohortTable
: Optionally drop cohort table in addition to stats tables (defaults to FALSE)

`exportCohortStatsTables` *Export the cohort statistics tables to the file system*

Description

This function retrieves the data from the cohort statistics tables and writes them to the inclusion statistics folder specified in the function call. NOTE: inclusion rule names are handled in one of two ways:

1. You can specify the cohortDefinitionSet parameter and the inclusion rule names will be extracted from the data.frame. 2. You can insert the inclusion rule names into the database using the insertInclusionRuleNames function of this package.

The first approach is preferred as to avoid the warning emitted.

Usage

```
exportCohortStatsTables(
  connectionDetails,
  connection = NULL,
  cohortDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  cohortStatisticsFolder,
  snakeCaseToCamelCase = TRUE,
  fileNamesInSnakeCase = FALSE,
  incremental = FALSE,
  databaseId = NULL,
  minCellCount = 5,
  cohortDefinitionSet = NULL,
  tablePrefix = ""
)
```

Arguments

connectionDetails
: An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.

`connection` An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`cohortDatabaseSchema`
Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortTableNames`
The names of the cohort tables. See `getCohortTableNames` for more details.

`cohortStatisticsFolder`
The path to the folder where the cohort statistics folder where the results will be written

`snakeCaseToCamelCase`
Should column names in the exported files convert from snake_case to camelCase? Default is FALSE

`fileNamesInSnakeCase`
Should the exported files use snake_case? Default is FALSE

`incremental` If `incremental = TRUE`, results are written to update values instead of overwriting an existing results (deprecated)

`databaseId` Optional - when specified, the databaseId will be added to the exported results

`minCellCount` To preserve privacy: the minimum number of subjects contributing to a count before it can be included in the results. If the count is below this threshold, it will be set to '-minCellCount'.

`cohortDefinitionSet`
The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`tablePrefix` Optional - allows to append a prefix to the exported file names.

`exportCohortSubsetStatsTables`
Export cohort subset statistics tables to the file system

Description

This function retrieves the data from the cohort subset statistics table and writes them to the subset statistics folder specified in the function call.

Usage

```
exportCohortSubsetStatsTables(
  connectionDetails,
  connection = NULL,
  cohortDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  cohortSubsetStatisticsFolder,
  snakeCaseToCamelCase = TRUE,
  fileNamesInSnakeCase = FALSE,
  databaseId = NULL,
  minCellCount = 5,
  tablePrefix = ""
)
```

Arguments

connectionDetails	An object of type connectionDetails as created using the createConnectionDetails function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the connect function in the DatabaseConnector package. Can be left NULL if connectionDetails is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.
cohortDatabaseSchema	Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.
cohortTableNames	The names of the cohort tables. See getCohortTableNames for more details.
cohortSubsetStatisticsFolder	The path to the folder where the cohort subset statistics results will be written.
snakeCaseToCamelCase	Should column names in the exported files convert from snake_case to camelCase? Default is FALSE
fileNamesInSnakeCase	Should the exported files use snake_case? Default is FALSE
databaseId	Optional - when specified, the databaseId will be added to the exported results
minCellCount	To preserve privacy: the minimum number of subjects contributing to a count before it can be included in the results. If the count is below this threshold, it will be set to '-minCellCount'.
tablePrefix	Optional - allows to append a prefix to the exported file names.

`generateCohortSet` *Generate a set of cohorts*

Description

This function generates a set of cohorts in the cohort table.

Usage

```
generateCohortSet(
  connectionDetails = NULL,
  connection = NULL,
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema = cdmDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  cohortDefinitionSet = NULL,
  stopOnError = TRUE,
  incremental = FALSE,
  incrementalFolder = NULL
)
```

Arguments

`connectionDetails`
: An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.

`connection`
: An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`cdmDatabaseSchema`
: Schema name where your patient-level data in OMOP CDM format resides. Note that for SQL Server, this should include both the database and schema name, for example 'cdm_data.dbo'.

`tempEmulationSchema`
: Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.

`cohortDatabaseSchema`
: Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortTableNames`
: The names of the cohort tables. See `getCohortTableNames` for more details.

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  sql The OHDSI-SQL used to generate the cohort

  Optionally, this data frame may contain:

  json The Circe JSON representation of the cohort

`stopOnError`
: If an error happens while generating one of the cohorts in the cohortDefinitionSet, should we stop processing the other cohorts? The default is TRUE; when set to FALSE, failures will be identified in the return value from this function.

`incremental`
: Create only cohorts that haven't been created before?

`incrementalFolder`
: If `incremental = TRUE`, specify a folder where records are kept of which definition has been executed. (deprecated)

Value

A data.frame consisting of the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

generationStatus The status of the generation task which may be one of the following:

- **COMPLETE** The generation completed successfully
- **FAILED** The generation failed (see logs for details)
- **SKIPPED** If using incremental == 'TRUE', this status indicates that the cohort's generation was skipped since it was previously completed.

startTime The start time of the cohort generation. If the generationStatus == 'SKIPPED', the startTime will be NA.

endTime The end time of the cohort generation. If the generationStatus == 'FAILED', the endTime will be the time of the failure. If the generationStatus == 'SKIPPED', endTime will be NA.

`generateNegativeControlOutcomeCohorts` *Generate a set of negative control outcome cohorts*

Description

This function generate a set of negative control outcome cohorts. For more information please see [Chapter 12 - Population Level Estimation](https://ohdsi.github.io/TheBookOfOhdsi/PopulationLevelEstimation.html) for more information how these cohorts are utilized in a study design.

Usage

```
generateNegativeControlOutcomeCohorts(
  connectionDetails = NULL,
  connection = NULL,
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema = cdmDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  cohortTable = cohortTableNames$cohortTable,
  negativeControlOutcomeCohortSet,
  occurrenceType = "all",
  incremental = FALSE,
  incrementalFolder = NULL,
  detectOnDescendants = FALSE
)
```

Arguments

`connectionDetails`
: An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.

`connection`
: An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`cdmDatabaseSchema`
: Schema name where your patient-level data in OMOP CDM format resides. Note that for SQL Server, this should include both the database and schema name, for example 'cdm_data.dbo'.

`tempEmulationSchema`
: Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.

`cohortDatabaseSchema`
: Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortTableNames`
: The names of the cohort tables. See `getCohortTableNames` for more details.

`cohortTable`
: Name of the cohort table.

`negativeControlOutcomeCohortSet`
: The `negativeControlOutcomeCohortSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  outcomeConceptId The concept_id in the condition domain to use for the negative control outcome.

occurrenceType The occurrenceType will detect either: the first time an outcomeConceptId occurs or all times the outcomeConceptId occurs for a person. Values accepted: 'all' or 'first'.

incremental Create only cohorts that haven't been created before?

incrementalFolder
If incremental = TRUE, specify a folder where records are kept of which definition has been executed. (deprecated)

detectOnDescendants
When set to TRUE, detectOnDescendants will use the vocabulary to find negative control outcomes using the outcomeConceptId and all descendants via the concept_ancestor table. When FALSE, only the exact outcomeConceptId will be used to detect the outcome.

Value

Invisibly returns an empty negative control outcome cohort set data.frame

---

getCohortCounts *Count the cohort(s)*

Description

Computes the subject and entry count per cohort. Note the cohortDefinitionSet parameter is optional - if you specify the cohortDefinitionSet, the cohort counts will be joined to the cohortDefinitionSet to include attributes like the cohortName.

Usage

```
getCohortCounts(
  connectionDetails = NULL,
  connection = NULL,
  cohortDatabaseSchema,
  cohortTable = "cohort",
  cohortIds = c(),
  cohortDefinitionSet = NULL,
  databaseId = NULL
)
```

Arguments

connectionDetails
An object of type connectionDetails as created using the createConnectionDetails function in the DatabaseConnector package. Can be left NULL if connection is provided.

connection An object of type connection as created using the connect function in the DatabaseConnector package. Can be left NULL if connectionDetails is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`cohortDatabaseSchema`
: Schema name where your cohort table resides. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortTable`
: The name of the cohort table.

`cohortIds`
: The cohort Id(s) used to reference the cohort in the cohort table. If left empty and no 'cohortDefinitionSet' argument is specified, all cohorts in the table will be included. If you specify the 'cohortIds' AND 'cohortDefinitionSet', the counts will reflect the 'cohortIds' from the 'cohortDefinitionSet'.

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  sql The OHDSI-SQL used to generate the cohort

  Optionally, this data frame may contain:

  json The Circe JSON representation of the cohort

`databaseId`
: Optional - when specified, the databaseId will be added to the exported results

Value

A data frame with cohort counts

`getCohortDefinitionSet` *Get a cohort definition set*

Description

This function supports the legacy way of retrieving a cohort definition set from the file system or in a package. This function supports the legacy way of storing a cohort definition set in a package with a CSV file, JSON files, and SQL files in the 'inst' folder.

Usage

```
getCohortDefinitionSet(
  settingsFileName = "Cohorts.csv",
  jsonFolder = "cohorts",
  sqlFolder = "sql/sql_server",
  cohortFileNameFormat = "%s",
  cohortFileNameValue = c("cohortId"),
  subsetJsonFolder = "inst/cohort_subset_definitions/",
  templateFolder = "inst/cohort_template_definitions/",
  packageName = NULL,
  warnOnMissingJson = TRUE,
  verbose = FALSE
)
```

Arguments

`settingsFileName`
: The name of the CSV file that will hold the cohort information including the cohortId and cohortName

`jsonFolder`
: The name of the folder that will hold the JSON representation of the cohort if it is available in the cohortDefinitionSet

`sqlFolder`
: The name of the folder that will hold the SQL representation of the cohort.

`cohortFileNameFormat`
: Defines the format string for naming the cohort JSON and SQL files. The format string follows the standard defined in the base sprintf function.

`cohortFileNameValue`
: Defines the columns in the cohortDefinitionSet to use in conjunction with the cohortFileNameFormat parameter.

`subsetJsonFolder`
: Defines the folder to store the subset JSON

`templateFolder`
: Defines the folder to store sql template cohorts that can be loaded as part of the definition JSON files are loaded into cohort definition set

`packageName`
: The name of the package containing the cohort definitions.

`warnOnMissingJson`
: Provide a warning if a .JSON file is not found for a cohort in the settings file

`verbose`
: When TRUE, extra logging messages are emitted

Value

Returns a cohort set data.frame

`getCohortInclusionRules` *Get Cohort Inclusion Rules from a cohort definition set*

Description

This function returns a data frame of the inclusion rules defined in a cohort definition set.

Usage

```
getCohortInclusionRules(cohortDefinitionSet)
```

Arguments

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`getCohortStats` *Get Cohort Inclusion Stats Table Data*

Description

This function returns a data frame of the data in the Cohort Inclusion Tables. Results are organized in to a list with 6 different data frames:

- cohortInclusionTable
- cohortInclusionResultTable
- cohortInclusionStatsTable
- cohortSummaryStatsTable
- cohortCensorStatsTable
- cohortAttritionTable

These can be optionally specified with the `outputTables`. See `exportCohortStatsTables` function for saving data to csv.

Usage

```
getCohortStats(
  connectionDetails,
  connection = NULL,
  cohortDatabaseSchema,
  databaseId = NULL,
  snakeCaseToCamelCase = TRUE,
  outputTables = c("cohortInclusionTable", "cohortInclusionResultTable",
   "cohortInclusionStatsTable", "cohortInclusionStatsTable", "cohortSummaryStatsTable",
     "cohortCensorStatsTable", "cohortAttritionTable"),
  cohortTableNames = getCohortTableNames(),
  inclusionRules = NULL
)
```

Arguments

`connectionDetails`
: An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.

connection	An object of type connection as created using the connect function in the DatabaseConnector package. Can be left NULL if connectionDetails is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.
cohortDatabaseSchema	Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.
databaseId	Optional - when specified, the databaseId will be added to the exported results
snakeCaseToCamelCase	Convert column names from snake case to camel case.
outputTables	Character vector. One or more of "cohortInclusionTable", "cohortInclusionResultTable", "cohortInclusionStatsTable", "cohortInclusionStatsTable", "cohortSummaryStatsTable" or "cohortCensorStatsTable", "cohortAttritionTable". Output is limited to these tables. Cannot export, for, example, the cohort table. Defaults to all stats tables.
cohortTableNames	The names of the cohort tables. See getCohortTableNames for more details.
inclusionRules	A data.frame with inclusion rules from the cohortDefinitionSet used to generate the cohort stats obtained by running getCohortInclusionRules(cohortDefinitionSet) (Optional)

---

getCohortTableNames *Used to get a list of cohort table names to use when creating the cohort tables*

Description

This function creates a list of table names used by createCohortTables to specify the table names to create. Use this function to specify the names of the main cohort table and cohort statistics tables.

Usage

```
getCohortTableNames(
  cohortTable = "cohort",
  cohortSampleTable = cohortTable,
  cohortInclusionTable = paste0(cohortTable, "_inclusion"),
  cohortInclusionResultTable = paste0(cohortTable, "_inclusion_result"),
  cohortInclusionStatsTable = paste0(cohortTable, "_inclusion_stats"),
  cohortSummaryStatsTable = paste0(cohortTable, "_summary_stats"),
  cohortCensorStatsTable = paste0(cohortTable, "_censor_stats"),
  cohortSubsetAttritionTable = paste0(cohortTable, "_subset_attrition"),
  cohortChecksumTable = paste0(cohortTable, "_checksum")
)
```

Arguments

`cohortTable`	Name of the cohort table.
`cohortSampleTable`	Name of the cohort table for sampled cohorts (defaults to the same as the cohort table).
`cohortInclusionTable`	Name of the inclusion table, one of the tables for storing inclusion rule statistics.
`cohortInclusionResultTable`	Name of the inclusion result table, one of the tables for storing inclusion rule statistics.
`cohortInclusionStatsTable`	Name of the inclusion stats table, one of the tables for storing inclusion rule statistics.
`cohortSummaryStatsTable`	Name of the summary stats table, one of the tables for storing inclusion rule statistics.
`cohortCensorStatsTable`	Name of the censor stats table, one of the tables for storing inclusion rule statistics.
`cohortSubsetAttritionTable`	Name of the subset attrition table for storing subset operator attrition.
`cohortChecksumTable`	Stores the checksum of the cohort used and the time generation starts and ends

Value

A list of the table names as specified in the parameters to this function.

`getCohortValidationCounts` *Validate cohort*

Description

Using custom sql, it is possible to generate cohorts that are not technically definitions. Invalid cohorts include the following:

* Cohorts where individuals have multiple, overlapping eras * Cohorts that have start dates that occur after their end dates * Cohorts with duplicate entries for the same subject.

Additionally the count for cohorts that lie outside the observation period for individuals is added. However, due to valid reasons in cohort definitions (e.g. fixed cohort duration, data source context) this cannot be directly considered a pass/fail diagnostic in all contexts.

Note - this code cannot formally verify the validity of a cohort. There may be situations where the logic of a cohort definition only causes errors in certain circumstances. Furthermore, if cohort counts are 0 this check is unable to evaluate validity at all.

The returned data.frame counts the number of errors found for each cohort. In addition a boolean "valid" field is applied that is TRUE only in the case where all counts are 0.

Usage

```
getCohortValidationCounts(
  connectionDetails = NULL,
  connection = NULL,
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema = cdmDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  cohortIds = NULL
)
```

Arguments

connectionDetails	An object of type connectionDetails as created using the createConnectionDetails function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the connect function in the DatabaseConnector package. Can be left NULL if connectionDetails is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.
cdmDatabaseSchema	Schema name where your patient-level data in OMOP CDM format resides. Note that for SQL Server, this should include both the database and schema name, for example 'cdm_data.dbo'.
tempEmulationSchema	Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.
cohortDatabaseSchema	Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.
cohortTableNames	The names of the cohort tables. See getCohortTableNames for more details.
cohortIds	Ids of cohorts to validate

Value

a data.frame with the fields cohortId, overlappingErasCount, invalidDateCount, duplicateCount, outsideObservationCount

`getDataMigrator` *Get database migrations instance*

Description

Returns ResultModelManager DataMigrationManager instance.

Usage

```
getDataMigrator(connectionDetails, databaseSchema, tablePrefix = "")
```

Arguments

`connectionDetails`	DatabaseConnector connection details object
`databaseSchema`	String schema where database schema lives
`tablePrefix`	(Optional) Use if a table prefix is used before table names (e.g. "cg_")

Value

Instance of ResultModelManager::DataMigrationManager that has interface for converting existing data models

`getExcludeOnIndexSubsetDefinitionIds`
Get Exclude On Index Subset Definition Ids

Description

Get the exclusion on index subset definition ids from a cohort definition set (if any have been added) Useful if keeping track in a script with complex business logic around what a cohort definition is for

Usage

```
getExcludeOnIndexSubsetDefinitionIds(cohortDefinitionSet)
```

Arguments

`cohortDefinitionSet`

The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`getIndicationSubsetDefinitionIds`
Get Indication Subset Definition Ids

Description

Get the indication subset definition ids from a cohort definition set (if any have been added) Useful if keeping track in a script with complex business logic around what a cohort definition is for

Usage

```
getIndicationSubsetDefinitionIds(cohortDefinitionSet)
```

Arguments

`cohortDefinitionSet`
The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`getLastGeneratedCohortChecksums`
Get last generated cohort checksums

Description

This gets a log of the last checksum for each cohort id stored in the cohort_checksum table.

This should be used to audit cohort generation as (if generated with cohort_generator) cohorts should always have an end time in this table. The last end time will be the cohort that is in the cohort table (assuming no other manual modifications are made to the cohort table itself).

This can be used downstream of CohortGenerator to evaluate if cohorts are consistent with passed definitions.

Usage

```
getLastGeneratedCohortChecksums(
  connectionDetails = NULL,
  connection = NULL,
  cohortId = NULL,
  cohortDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  .checkTables = TRUE
)
```

Arguments

connectionDetails	An object of type connectionDetails as created using the createConnectionDetails function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the connect function in the DatabaseConnector package. Can be left NULL if connectionDetails is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.
cohortId	cohortId to check. If NULL, all cohorts will be returned.
cohortDatabaseSchema	Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.
cohortTableNames	The names of the cohort tables. See getCohortTableNames for more details.
.checkTables	used internally

---

getRestrictionSubsetDefinitionIds *Get Restriction Subset Definition Ids*

Description

Get the restriction subset definition ids from a cohort definition set (if any have been added) Useful if keeping track in a script with complex business logic around what a cohort definition is for

Usage

```
getRestrictionSubsetDefinitionIds(cohortDefinitionSet)
```

Arguments

cohortDefinitionSet

The cohortDefinitionSet argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

---

getResultsDataModelSpecifications *Get specifications for CohortGenerator results data model*

Description

Get specifications for CohortGenerator results data model

Usage

```
getResultsDataModelSpecifications()
```

Value

A tibble data frame object with specifications

---

getSubsetDefinitions *Get cohort subset definitions from a cohort definition set*

Description

Get the subset definitions (if any) applied to a cohort definition set. Note that these subset definitions are a copy of those applied to the cohort set. Modifying these definitions will not modify the base cohort set. To apply a modification, reapply the subset definition to the cohort definition set data.frame with addCohortSubsetDefinition with 'overwriteExisting = TRUE'.

Usage

```
getSubsetDefinitions(cohortDefinitionSet)
```

Arguments

cohortDefinitionSet

A valid cohortDefinitionSet

Value

list of cohort subset definitions or empty list

`getTemplateDefinitions` *Extract template definitions from a cohort definition set*

Description

Extract template definitions from a cohort definition set

Usage

```
getTemplateDefinitions(cohortDefinitionSet)
```

Arguments

`cohortDefinitionSet`

The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`insertInclusionRuleNames` *Used to insert the inclusion rule names from a cohort definition set when generating cohorts that include cohort statistics*

Description

This function will take a cohortDefinitionSet that inclusions the Circe JSON representation of each cohort, parse the InclusionRule property to obtain the inclusion rule name and sequence number and insert the values into the cohortInclusionTable. This function is only required when generating cohorts that include cohort statistics.

Usage

```
insertInclusionRuleNames(
  connectionDetails = NULL,
  connection = NULL,
  cohortDefinitionSet,
  cohortDatabaseSchema,
  cohortInclusionTable = getCohortTableNames()$cohortInclusionTable
)
```

Arguments

`connectionDetails`
: An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.

`connection`
: An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  sql The OHDSI-SQL used to generate the cohort

  Optionally, this data frame may contain:

  json The Circe JSON representation of the cohort

`cohortDatabaseSchema`
: Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortInclusionTable`
: Name of the inclusion table, one of the tables for storing inclusion rule statistics.

Value

A data frame containing the inclusion rules by cohort and sequence ID

`isCamelCase` *Used to check if a string is in lower camel case*

Description

This function is used check if a string conforms to the lower camel case format.

Usage

```
isCamelCase(x)
```

Arguments

`x`
: The string to evaluate

Value

TRUE if the string is in lower camel case

isCohortDefinitionSet *Is the data.frame a cohort definition set?*

Description

This function checks a data.frame to verify it holds the expected format for a cohortDefinitionSet.

Usage

```
isCohortDefinitionSet(x)
```

Arguments

x	The data.frame to check

Value

Returns TRUE if the input is a cohortDefinitionSet or returns FALSE with warnings on any violations

isFormattedForDatabaseUpload *Is the data.frame formatted for uploading to a database?*

Description

This function is used to check a data.frame to ensure all column names are in snake case format.

Usage

```
isFormattedForDatabaseUpload(x, warn = TRUE)
```

Arguments

x	A data frame
warn	When TRUE, display a warning of any columns are not in snake case format

Value

Returns TRUE if all columns are snake case format. If warn == TRUE, the function will emit a warning on the column names that are not in snake case format.

isSnakeCase *Used to check if a string is in snake case*

Description

This function is used check if a string conforms to the snake case format.

Usage

```
isSnakeCase(x)
```

Arguments

x	The string to evaluate

Value

TRUE if the string is in snake case

LimitSubsetOperator *Limit Subset Operator*

Description

operator to apply limiting subset operations (e.g. washout periods, calendar ranges or earliest entries)

Get auto generated name

Super class

SubsetOperator -> LimitSubsetOperator

Active bindings

priorTime minimum washout time in days

followUpTime minimum required follow up time in days

minimumCohortDuration minimum cohort duration time in days

maximumCohortDuration maximum cohort duration time in days

limitTo character one of: "firstEver" - only first entry in patient history "earliestRemaining" - only first entry after washout set by priorTime "latestRemaining" - the latest remaining after washout set by followUpTime "lastEver" - only last entry in patient history inside

Note, when using firstEver and lastEver with follow up and washout, patients with events outside this will be censored.

calendarStartDate The calendar start date for limiting by date

calendarEndDate The calendar end date for limiting by date

Methods

Public methods:

- `LimitSubsetOperator$getAutoGeneratedName()`
- `LimitSubsetOperator$toList()`
- `LimitSubsetOperator$clone()`

`LimitSubsetOperator$getAutoGeneratedName()`: name generated from subset operation properties

Usage:

```
LimitSubsetOperator$getAutoGeneratedName()
```

Returns: character To List

`LimitSubsetOperator$toList()`: List representation of object

Usage:

```
LimitSubsetOperator$toList()
```

`LimitSubsetOperator$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LimitSubsetOperator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`migrateDataModel` *Migrate Data model*

Description

Migrate data from current state to next state

It is strongly advised that you have a backup of all data (either sqlite files, a backup database (in the case you are using a PostgreSQL backend) or have kept the csv/zip files from your data generation.

Usage

```
migrateDataModel(connectionDetails, databaseSchema, tablePrefix = "")
```

Arguments

`connectionDetails`	DatabaseConnector connection details object
`databaseSchema`	String schema where database schema lives
`tablePrefix`	(Optional) Use if a table prefix is used before table names (e.g. "cg_")

omopCdmDrugExposure *OMOP CDM Drug Exposure Sample Data*

Description

A data set containing sample drug exposures for 2 drugs

Usage

```
omopCdmDrugExposure
```

Format

A data frame with 8 rows and 5 variables:

drug_exposure_id A unique identifier for the drug exposure

person_id An integer representing the patient

drug_concept_id An integer concept ID representing the drug concept

drug_exposure_start_date Drug start date

drug_exposure_end_date Drug end date

Source

Fictional data for demonstration.

omopCdmPerson *OMOP CDM Person Sample Data*

Description

A data set containing sample persons

Usage

```
omopCdmPerson
```

Format

A data frame with 12 rows and 5 variables:

person_id A unique identifier for the person

gender_concept_id An integer concept ID representing the person's gender

year_of_birth Year of birth

race_concept_id An integer concept ID representing the person's race

ethnicity_concept_id An integer concept ID representing the person's ethnicity

Source

Fictional data for demonstration.

`readCsv` *Used to read a .csv file*

Description

This function is used to centralize the function for reading .csv files across the HADES ecosystem. This function will automatically convert from snake_case in the file to camelCase in the data.frame returned as is the standard described in: https://ohdsi.github.io/Hades/codeStyle.html#Interfacing_between_R_and_SQL

Usage

```
readCsv(file, warnOnCaseMismatch = TRUE, colTypes = readr::cols())
```

Arguments

`file`	The .csv file to read.
`warnOnCaseMismatch`	When TRUE, raise a warning if column headings in the .csv are not in snake_case format
`colTypes`	Corresponds to the 'col_types' in the 'readr::read_csv' function. One of 'NULL', a [readr::cols()] specification, or a string. See 'vignette("readr")' for more details. If 'NULL', all column types will be inferred from 'guess_max' rows of the input, interspersed throughout the file. This is convenient (and fast), but not robust. If the guessed types are wrong, you'll need to increase 'guess_max' or supply the correct types yourself. Column specifications created by [list()] or [cols()] must contain one column specification for each column. Alternatively, you can use a compact string representation where each character represents one column: - c = character - i = integer - n = number - d = double - l = logical - f = factor - D = date - T = date time - t = time - ? = guess - _ or - = skip By default, reading a file without a column specification will print a message showing what 'readr' guessed they were. To remove this message, set 'show_col_types = FALSE' or set 'options(readr.show_col_types = FALSE)'.

Value

A tibble with the .csv contents

runCohortGeneration *Run a cohort generation and export results*

Description

Run a cohort generation and export results

Usage

```
runCohortGeneration(
  connectionDetails,
  cdmDatabaseSchema,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema = cdmDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  cohortDefinitionSet = NULL,
  negativeControlOutcomeCohortSet = NULL,
  occurrenceType = "all",
  detectOnDescendants = FALSE,
  stopOnError = TRUE,
  outputFolder,
  databaseId = 1,
  minCellCount = 5,
  incremental = FALSE,
  incrementalFolder = NULL
)
```

Arguments

`connectionDetails`
: An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package.

`cdmDatabaseSchema`
: Schema name where your patient-level data in OMOP CDM format resides. Note that for SQL Server, this should include both the database and schema name, for example 'cdm_data.dbo'.

`tempEmulationSchema`
: Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.

`cohortDatabaseSchema`
: Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`cohortTableNames`
: The names of the cohort tables. See `getCohortTableNames` for more details.

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  sql The OHDSI-SQL used to generate the cohort

  Optionally, this data frame may contain:

  json The Circe JSON representation of the cohort

`negativeControlOutcomeCohortSet`
: The `negativeControlOutcomeCohortSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  outcomeConceptId The concept_id in the condition domain to use for the negative control outcome.

`occurrenceType` For negative controls outcomes, the occurrenceType will detect either: the first time an outcomeConceptId occurs or all times the outcomeConceptId occurs for a person. Values accepted: 'all' or 'first'.

`detectOnDescendants`
: For negative controls outcomes, when set to TRUE, detectOnDescendants will use the vocabulary to find negative control outcomes using the outcomeConceptId and all descendants via the concept_ancestor table. When FALSE, only the exact outcomeConceptId will be used to detect the outcome.

`stopOnError` If an error happens while generating one of the cohorts in the cohortDefinitionSet, should we stop processing the other cohorts? The default is TRUE; when set to FALSE, failures will be identified in the return value from this function.

`outputFolder` Name of the folder where all the outputs will written to.

`databaseId` A unique ID for the database. This will be appended to most tables.

`minCellCount` To preserve privacy: the minimum number of subjects contributing to a count before it can be included in the results. If the count is below this threshold, it will be set to '-minCellCount'.

`incremental` Create only cohorts that haven't been created before?

`incrementalFolder`
: If `incremental = TRUE`, specify a folder where records are kept of which definition has been executed. (deprecated)

Details

Run a cohort generation for a set of cohorts and negative control outcomes. This function will also export the results of the run to the 'outputFolder'.

`sampleCohortDefinitionSet` *Sample Cohort Definition Set*

Description

Create 1 or more sample of size n of a cohort definition set

Subsetted cohorts can be sampled, as with any other subset form. However, subsetting a sampled cohort is not recommended and not currently supported at this time. In the case where n > cohort count the entire cohort is copied unmodified

As different databases have different forms of randomness, the random selection is computed in R, based on the count for each cohort. This is, therefore, db platform independent

Note, this function assumes cohorts have already been generated.

Lifecycle Note: This functionality is considered experimental and not intended for use inside analytic packages

Usage

```
sampleCohortDefinitionSet(
  cohortDefinitionSet,
  cohortIds = cohortDefinitionSet$cohortId,
  connectionDetails = NULL,
  connection = NULL,
  tempEmulationSchema = getOption("sqlRenderTempEmulationSchema"),
  cohortDatabaseSchema,
  outputDatabaseSchema = cohortDatabaseSchema,
  cohortTableNames = getCohortTableNames(),
  n = NULL,
  sampleFraction = NULL,
  seed = 64374,
  seedArgs = NULL,
  identifierExpression = "cohortId * 1000 + seed",
  incremental = FALSE,
  incrementalFolder = NULL
)
```

Arguments

`cohortDefinitionSet`

The `cohortDefinitionSet` argument must be a data frame with the following columns:

cohortId The unique integer identifier of the cohort

cohortName The cohort's name

sql The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

json The Circe JSON representation of the cohort

`cohortIds` Optional subset of cohortIds to generate. By default this function will sample all cohorts

`connectionDetails`
An object of type `connectionDetails` as created using the `createConnectionDetails` function in the DatabaseConnector package. Can be left NULL if `connection` is provided.

`connection` An object of type `connection` as created using the `connect` function in the DatabaseConnector package. Can be left NULL if `connectionDetails` is provided, in which case a new connection will be opened at the start of the function, and closed when the function finishes.

`tempEmulationSchema`
Some database platforms like Oracle and Impala do not truly support temp tables. To emulate temp tables, provide a schema with write privileges where temp tables can be created.

`cohortDatabaseSchema`
Schema name where your cohort tables reside. Note that for SQL Server, this should include both the database and schema name, for example 'scratch.dbo'.

`outputDatabaseSchema`
optional schema to output cohorts to (if different from cohortDatabaseSchema)

`cohortTableNames`
The names of the cohort tables. See `getCohortTableNames` for more details.

`n` Sample size. Ignored if sample fraction is set

`sampleFraction` Fraction of cohort to sample

`seed` Vector of seeds to give to the R pseudorandom number generator

`seedArgs` optional arguments to pass to set.seed

`identifierExpression`
Optional string R expression used to compute output cohort id. Can only use variables cohortId and seed. Default is "cohortId * 1000 + seed", which is substituted and evaluated

`incremental` Create only cohorts that haven't been created before?

`incrementalFolder`
If `incremental = TRUE`, specify a folder where records are kept of which definition has been executed. (deprecated)

Value

sampledCohortDefinitionSet - a data.frame like object that contains the resulting identifiers and modified names of cohorts

`saveCohortDefinitionSet` *Save the cohort definition set to the file system*

Description

This function saves a cohortDefinitionSet to the file system and provides options for specifying where to write the individual elements: the settings file will contain the cohort information as a CSV specified by the settingsFileName, the cohort JSON is written to the jsonFolder and the SQL is written to the sqlFolder. We also provide a way to specify the json/sql file name format using the cohortFileNameFormat and cohortFileNameValue parameters.

Usage

```
saveCohortDefinitionSet(
  cohortDefinitionSet,
  settingsFileName = "inst/Cohorts.csv",
  jsonFolder = "inst/cohorts",
  sqlFolder = "inst/sql/sql_server",
  cohortFileNameFormat = "%s",
  cohortFileNameValue = c("cohortId"),
  subsetJsonFolder = "inst/cohort_subset_definitions/",
  templateFolder = "inst/cohort_template_definitions/",
  verbose = FALSE
)
```

Arguments

`cohortDefinitionSet`
: The `cohortDefinitionSet` argument must be a data frame with the following columns:

  cohortId The unique integer identifier of the cohort

  cohortName The cohort's name

  sql The OHDSI-SQL used to generate the cohort

  Optionally, this data frame may contain:

  json The Circe JSON representation of the cohort

`settingsFileName`
: The name of the CSV file that will hold the cohort information including the cohortId and cohortName

`jsonFolder`
: The name of the folder that will hold the JSON representation of the cohort if it is available in the cohortDefinitionSet

`sqlFolder`
: The name of the folder that will hold the SQL representation of the cohort.

`cohortFileNameFormat`
: Defines the format string for naming the cohort JSON and SQL files. The format string follows the standard defined in the base sprintf function.

`cohortFileNameValue`
: Defines the columns in the cohortDefinitionSet to use in conjunction with the cohortFileNameFormat parameter.

`subsetJsonFolder`
: Defines the folder to store the subset JSON

`templateFolder`
: Defines the folder to store sql template cohorts that can be saved as part of the definition Sql will be copied to this location when ‘saveCohortDefinitionSet‘ is called.

`verbose`
: When TRUE, logging messages are emitted to indicate export progress.

---

saveCohortSubsetDefinition *Save cohort subset definitions to json*

Description

This is generally used as part of saveCohortDefinitionSet

Usage

```
saveCohortSubsetDefinition(
  subsetDefinition,
  subsetJsonFolder = "inst/cohort_subset_definitions/"
)
```

Arguments

`subsetDefinition`
: The subset definition object @seealso[CohortSubsetDefinition]

`subsetJsonFolder`
: Defines the folder to store the subset JSON

SubsetCohortWindow *Time Window For Cohort Subset Operator*

Description

Representation of a time window to use when subsetting a target cohort with a subset cohort

Active bindings

`startDay` Integer

`endDay` Integer

`targetAnchor` Boolean

`subsetAnchor` Boolean

`negate` Boolean

Methods

Public methods:

- SubsetCohortWindow$toList()
- SubsetCohortWindow$toJSON()
- SubsetCohortWindow$isEqualTo()
- SubsetCohortWindow$clone()

`SubsetCohortWindow$toList()`: List representation of object To JSON

Usage:

```
SubsetCohortWindow$toList()
```

`SubsetCohortWindow$toJSON()`: json serialized representation of object Is Equal to

Usage:

```
SubsetCohortWindow$toJSON()
```

`SubsetCohortWindow$isEqualTo()`: Compare SubsetCohortWindow to another

Usage:

```
SubsetCohortWindow$isEqualTo(criteria)
```

Arguments:

`criteria` SubsetCohortWindow instance

`SubsetCohortWindow$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SubsetCohortWindow$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

SubsetOperator *Abstract base class for subsets.*

Description

Abstract Base Class for subsets. Subsets should inherit from this and implement their own requirements.

Active bindings

`name` name of subset operation - should describe what the operation does e.g. "Males under the age of 18", "Exposed to Celecoxib"

Methods

Public methods:

- SubsetOperator$new()
- SubsetOperator$classname()
- SubsetOperator$getAutoGeneratedName()
- SubsetOperator$getQueryBuilder()
- SubsetOperator$publicFields()
- SubsetOperator$isEqualTo()
- SubsetOperator$toList()
- SubsetOperator$toJSON()
- SubsetOperator$print()
- SubsetOperator$clone()

`SubsetOperator$new()`:

Usage:

```
SubsetOperator$new(definition = NULL)
```

Arguments:

`definition` json character or list - definition of subset operator

Returns: instance of object Class Name

`SubsetOperator$classname()`: Class name of object Get auto generated name

Usage:

```
SubsetOperator$classname()
```

`SubsetOperator$getAutoGeneratedName()`: Not intended to be used - should be implemented in subclasses Return query builder instance

Usage:

```
SubsetOperator$getAutoGeneratedName()
```

`SubsetOperator$getQueryBuilder()`: Return query builder instance Public Fields

Usage:

```
SubsetOperator$getQueryBuilder(id)
```

Arguments:

`id` - integer that should be unique in the sql (e.g. increment it by one for each subset operation in set)

`SubsetOperator$publicFields()`: Publicly settable fields of object Is Equal to

Usage:

```
SubsetOperator$publicFields()
```

`SubsetOperator$isEqualTo()`: Compare Subsets - are they identical or not? Checks all fields and settings

Usage:

`SubsetOperator$isEqualTo(subsetOperatorB)`

Arguments:

`subsetOperatorB` A subset to test equivalence to To list

`SubsetOperator$toList()`: convert to List representation To Json

Usage:

`SubsetOperator$toList()`

`SubsetOperator$toJSON()`: convert to json serialized representation

Usage:

`SubsetOperator$toJSON()`

Returns: list representation of object as json character Pretty print

`SubsetOperator$print()`:

Usage:

`SubsetOperator$print(...)`

Arguments:

`...` further arguments passed to or from other methods.

`SubsetOperator$clone()`: The objects of this class are cloneable with this method.

Usage:

`SubsetOperator$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

CohortSubsetOperator

DemographicSubsetOperator

LimitSubsetOperator

`uploadResults` *Upload results to the database server.*

Description

Requires the results data model tables have been created using the `createResultsDataModel` function.

Usage

```
uploadResults(
  connectionDetails,
  schema,
  resultsFolder,
  forceOverWriteOfSpecifications = FALSE,
  purgeSiteDataBeforeUploading = TRUE,
  tablePrefix = "",
  ...
)
```

Arguments

connectionDetails	An object of type connectionDetails as created using the createConnectionDetails function in the DatabaseConnector package.
schema	The schema on the server where the tables have been created.
resultsFolder	The folder holding the results in .csv files
forceOverWriteOfSpecifications	If TRUE, specifications of the phenotypes, cohort definitions, and analysis will be overwritten if they already exist on the database. Only use this if these specifications have changed since the last upload.
purgeSiteDataBeforeUploading	If TRUE, before inserting data for a specific databaseId all the data for that site will be dropped. This assumes the resultsFolder file contains the full data for that data site.
tablePrefix	(Optional) string to insert before table names for database table names
...	See ResultModelManager::uploadResults

---

writeCsv *Used to write a .csv file*

Description

This function is used to centralize the function for writing .csv files across the HADES ecosystem. This function will automatically convert from camelCase in the data.frame to snake_case column names in the resulting .csv file as is the standard described in: https://ohdsi.github.io/Hades/codeStyle.html#Interfacing_betwe

This function may also raise warnings if the data is stored in a format that will not work with the HADES standard for uploading to a results database. Specifically file names should be in snake_case format, all column headings are in snake_case format and where possible the file name should not be plural. See isFormattedForDatabaseUpload for a helper function to check a data.frame for rules on the column names

Usage

```
writeCsv(
  x,
  file,
  append = FALSE,
  warnOnCaseMismatch = TRUE,
  warnOnFileNameCaseMismatch = TRUE,
  warnOnUploadRuleViolations = TRUE
)
```

Arguments

x	A data frame or tibble to write to disk.
file	The .csv file to write.
append	When TRUE, append the values of x to an existing file.
warnOnCaseMismatch	When TRUE, raise a warning if columns in the data.frame are NOT in camelCase format.
warnOnFileNameCaseMismatch	When TRUE, raise a warning if the file name specified is not in snake_case format.
warnOnUploadRuleViolations	When TRUE, this function will provide warning messages that may indicate if the data is stored in a format in the .csv that may cause problems when uploading to a database.

Value

Returns the input x invisibly.

Index

∗ **datasets**
omopCdmDrugExposure, 59
omopCdmPerson, 59
∗ **subsets**
createCohortSubsetOperator, 24
createDemographicSubsetOperator, 27
createLimitSubsetOperator, 29
∗ **utils**
computeChecksum, 20

addCohortSubsetDefinition, 3
addCohortTemplateDefintion, 4
addExcludeOnIndexSubsetDefinition, 5
addIndicationSubsetDefinition, 7
addRestrictionSubsetDefinition, 9
addSqlCohortDefinition, 11
addUnionCohortDefinition, 13

checkAndFixCohortDefinitionSetDataTypes, 13
CohortSubsetDefinition, 14
CohortSubsetOperator, 16
CohortTemplateDefinition, 18
computeChecksum, 20
computeCohortAttrition, 21
connect, *19*, *25*, *36*, *38–40*, *42, 43*, *47*, *49*, *52*, *55*, *64*
createAtcCohortTemplateDefinition, 22
createCohortSubset, 23
createCohortSubsetDefinition, 23
createCohortSubsetOperator, 24
createCohortSubsetOperator(), *27*, *30*
createCohortTables, 25, *47*
createCohortTemplateDefintion, 26
createConnectionDetails, *25*, *36, 37*, *39, 40*, *42, 43*, *46*, *49*, *52*, *55*, *61*, *64*, *70*
createDemographicSubset, 26
createDemographicSubsetOperator, 27
createDemographicSubsetOperator(), *25*, *30*
createEmptyCohortDefinitionSet, 28
createEmptyNegativeControlOutcomeCohortSet, 28
createLimitSubset, 29
createLimitSubsetOperator, 29
createLimitSubsetOperator(), *25*, *27*
createResultsDataModel, 30, *69*
createRxNormCohortTemplateDefinition, 31
createSnomedCohortTemplateDefinition, 32
createSubsetCohortWindow, 33
createUnionCohortTemplate, 34

DemographicSubsetOperator, 34
dropCohortStatsTables, 36

exportCohortStatsTables, 37
exportCohortSubsetStatsTables, 38

generateCohortSet, 40
generateNegativeControlOutcomeCohorts, 41
getCohortCounts, 43
getCohortDefinitionSet, 44
getCohortInclusionRules, 45
getCohortStats, 46
getCohortTableNames, *19*, *25*, *37–40*, *42*, *47*, 47, *49*, *52*, *61*, *64*
getCohortValidationCounts, 48
getDataMigrator, 50
getExcludeOnIndexSubsetDefinitionIds, 50
getIndicationSubsetDefinitionIds, 51
getLastGeneratedCohortChecksums, 51
getRestrictionSubsetDefinitionIds, 52
getResultsDataModelSpecifications, 53
getSubsetDefinitions, 53

getTemplateDefinitions, 54

insertInclusionRuleNames, 54
isCamelCase, 55
isCohortDefinitionSet, 56
isFormattedForDatabaseUpload, 56
isSnakeCase, 57

LimitSubsetOperator, 57

migrateDataModel, 58

omopCdmDrugExposure, 59
omopCdmPerson, 59

readCsv, 60
runCohortGeneration, 61

sampleCohortDefinitionSet, 63
saveCohortDefinitionSet, 65
saveCohortSubsetDefinition, 66
SubsetCohortWindow, 66
SubsetOperator, *16*, *34*, *57*, 67

uploadResults, 69

writeCsv, 70