

# Package: CohortGenerator (via r-universe)

October 1, 2024

**Type** Package

**Title** Cohort Generation for the OMOP Common Data Model

**Version** 0.11.2

**Date** 2024-09-30

**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 (>= 3.6.0), R6

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

**Suggests** CirceR (>= 1.1.1), Eunomia, 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>

**RoxygenNote** 7.3.1

**Encoding** UTF-8

**Language** en-US

**NeedsCompilation** no

**Author** Anthony Sena [aut, cre], Jamie Gilbert [aut], Gowtham Rao [aut], Martijn Schuemie [aut], Observational Health Data Science and Informatics [cph]

**Repository** CRAN

**Date/Publication** 2024-09-30 19:40:02 UTC

## Contents

addCohortSubsetDefinition . . . . .	3
checkAndFixCohortDefinitionSetDataTypes . . . . .	4
CohortSubsetDefinition . . . . .	4
CohortSubsetOperator . . . . .	7
computeChecksum . . . . .	8
createCohortSubset . . . . .	8
createCohortSubsetDefinition . . . . .	9
createCohortTables . . . . .	10
createDemographicSubset . . . . .	10
createEmptyCohortDefinitionSet . . . . .	11
createEmptyNegativeControlOutcomeCohortSet . . . . .	12
createLimitSubset . . . . .	12
createResultsDataModel . . . . .	13
createSubsetCohortWindow . . . . .	14
DemographicSubsetOperator . . . . .	14
dropCohortStatsTables . . . . .	16
exportCohortStatsTables . . . . .	17
generateCohortSet . . . . .	18
generateNegativeControlOutcomeCohorts . . . . .	20
getCohortCounts . . . . .	22
getCohortDefinitionSet . . . . .	23
getCohortInclusionRules . . . . .	24
getCohortStats . . . . .	25
getCohortTableNames . . . . .	26
getDataMigrator . . . . .	27
getRequiredTasks . . . . .	27
getResultsDataModelSpecifications . . . . .	28
getSubsetDefinitions . . . . .	28
insertInclusionRuleNames . . . . .	29
isCamelCase . . . . .	30
isCohortDefinitionSet . . . . .	30
isFormattedForDatabaseUpload . . . . .	31
isSnakeCase . . . . .	31
isTaskRequired . . . . .	32
LimitSubsetOperator . . . . .	32
migrateDataModel . . . . .	33
readCsv . . . . .	34
recordTasksDone . . . . .	35
runCohortGeneration . . . . .	35
sampleCohortDefinitionSet . . . . .	37
saveCohortDefinitionSet . . . . .	39
saveCohortSubsetDefinition . . . . .	40
saveIncremental . . . . .	41
SubsetCohortWindow . . . . .	41
SubsetOperator . . . . .	42
uploadResults . . . . .	44

<i>addCohortSubsetDefinition</i>	3
writeCsv . . . . .	45
<b>Index</b>	<b>46</b>

---

`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

---

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

**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\$new()
- CohortSubsetDefinition\$list()
- CohortSubsetDefinition\$json()
- CohortSubsetDefinition\$addSubsetOperator()
- CohortSubsetDefinition\$getSubsetQuery()
- CohortSubsetDefinition\$getSubsetCohortName()
- CohortSubsetDefinition\$setTargetOutputPairs()
- CohortSubsetDefinition\$jsonFileName()
- CohortSubsetDefinition\$clone()

**Method** new():*Usage:*

CohortSubsetDefinition\$new(definition = NULL)

*Arguments:*

definition json or list representation of object to List

**Method** toList(): List representation of object to JSON*Usage:*

CohortSubsetDefinition\$list()

**Method** toJSON(): json serialized representation of object add Subset Operator*Usage:*

CohortSubsetDefinition\$json()

**Method** 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

**Method** `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

**Method** `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

**Method** `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

**Method** `getJsonFileName()`:

*Usage:*

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

*Arguments:*

`subsetJsonFolder` path to folder to place file

**Method** `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 to List

### Super class

[CohortGenerator::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

startWindow The time window to use evaluating the subset cohort start relative to the target cohort

endWindow The time window to use evaluating the subset cohort end relative to the target cohort

### Methods

#### Public methods:

- [CohortSubsetOperator\\$toList\(\)](#)
- [CohortSubsetOperator\\$getAutoGeneratedName\(\)](#)
- [CohortSubsetOperator\\$clone\(\)](#)

**Method** [toList\(\)](#): List representation of object Get auto generated name

*Usage:*

[CohortSubsetOperator\\$toList\(\)](#)

**Method** [getAutoGeneratedName\(\)](#): name generated from subset operation properties

*Usage:*

[CohortSubsetOperator\\$getAutoGeneratedName\(\)](#)

*Returns:* character

**Method** [clone\(\)](#): The objects of this class are cloneable with this method.

*Usage:*

[CohortSubsetOperator\\$clone\(deep = FALSE\)](#)

*Arguments:*

deep Whether to make a deep clone.

---

computeChecksum	<i>Computes the checksum for a value</i>
-----------------	--

---

### 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

---

createCohortSubset	<i>A definition of subset functions to be applied to a set of cohorts</i>
--------------------	---

---

### Description

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

### Usage

```
createCohortSubset(
  name = NULL,
  cohortIds,
  cohortCombinationOperator,
  negate,
  startWindow,
  endWindow
)
```

### 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
startWindow	A SubsetCohortWindow that patients must fall inside (see createSubsetCohortWindow)
endWindow	A SubsetCohortWindow that patients must fall inside (see createSubsetCohortWindow)

**Value**

a CohortSubsetOperator instance

---

createCohortSubsetDefinition  
*Create Subset Definition*

---

**Description**

Create subset definition from subset objects

**Usage**

```
createCohortSubsetDefinition(
  name,
  definitionId,
  subsetOperators,
  identifierExpression = NULL,
  operatorNameConcatString = "",
  subsetCohortNameTemplate = ""
)
```

**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)
operatorNameConcatString	(optional) String to concatenate operator names together when outputting resulting cohort name
subsetCohortNameTemplate	(optional) SqlRender string template for formatting names of resulting subset cohorts Can use the variables @baseCohortName, @subsetDefinitionName and @operatorNames. This is applied when adding the subset definition to a cohort definition set.

---

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 <a href="#">createConnectionDetails</a> function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the <a href="#">connect</a> 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 <a href="#">getCohortTableNames</a> 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.

---

createDemographicSubset  
                                   *Create createDemographicSubset Subset*

---

### Description

Create createDemographicSubset Subset

**Usage**

```
createDemographicSubset(
  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. <a href="https://athena.ohdsi.org/search-terms/terms?standardConcept=Standard&amp;domain=Gender&amp;page=">https://athena.ohdsi.org/search-terms/terms?standardConcept=Standard&amp;domain=Gender&amp;page=</a> 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

---

```
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
```

---

**Description**

Subset cohorts using specified limit criteria

**Usage**

```
createLimitSubset(
  name = NULL,
  priorTime = 0,
  followUpTime = 0,
  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)

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)

---

```
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.

---

createSubsetCohortWindow

*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

createSubsetCohortWindow(startDay, endDay, targetAnchor)

### Arguments

startDay	The start day for the window
endDay	The end day for the window
targetAnchor	To anchor using the target cohort's start date or end date

### Value

a SubsetCohortWindow instance

---

DemographicSubsetOperator

*Demographic Subset Operator*

---

### Description

Operators for subsetting a cohort by demographic criteria

### Value

char vector Get auto generated name

### Super class

[CohortGenerator::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\(\)](#)

**Method** `toList()`: List representation of object Map gender concepts to names

*Usage:*

```
DemographicSubsetOperator$toList()
```

**Method** `mapGenderConceptsToNames()`:

*Usage:*

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

*Arguments:*

mapping optional list of mappings for concept id to nouns

**Method** `getAutoGeneratedName()`: name generated from subset operation properties

*Usage:*

```
DemographicSubsetOperator$getAutoGeneratedName()
```

*Returns:* character

**Method** `toJSON()`: json serialized representation of object

*Usage:*

```
DemographicSubsetOperator$toJSON()
```

**Method** `isEqualTo()`: Compare Subset to another

*Usage:*

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

*Arguments:*

criteria DemographicSubsetOperator instance

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

*Usage:*

```
DemographicSubsetOperator$getGender()
```

**Method** getRace(): Race getter - used when constructing SQL to default NULL to an empty string

*Usage:*

```
DemographicSubsetOperator$getRace()
```

**Method** getEthnicity(): Ethnicity getter - used when constructing SQL to default NULL to an empty string

*Usage:*

```
DemographicSubsetOperator$getEthnicity()
```

**Method** 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 <a href="#">createConnectionDetails</a> function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the <a href="#">connect</a> 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 <a href="#">getCohortTableNames</a> 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,
  fileNameInSnakeCase = FALSE,
  incremental = FALSE,
  databaseId = NULL,
  minCellCount = 5,
  cohortDefinitionSet = NULL,
  tablePrefix = ""
)
```

## Arguments

connectionDetails	An object of type connectionDetails as created using the <a href="#">createConnectionDetails</a> function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the <a href="#">connect</a> 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 <a href="#">getCohortTableNames</a> 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 camel-Case? 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
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: <b>cohortId</b> The unique integer identifier of the cohort <b>cohortName</b> The cohort's name <b>sql</b> The OHDSI-SQL used to generate the cohort Optionally, this data frame may contain: <b>json</b> The Circe JSON representation of the cohort
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.

**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,
  cohortTable = getCohortTableNames()$cohortTable,
  negativeControlOutcomeCohortSet,
  occurrenceType = "all",
  incremental = FALSE,
  incrementalFolder = NULL,
  detectOnDescendants = FALSE
)
```

**Arguments**

connectionDetails	An object of type connectionDetails as created using the <a href="#">createConnectionDetails</a> function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the <a href="#">connect</a> 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'.
cohortTable	Name of the cohort table.
negativeControlOutcomeCohortSet	The negativeControlOutcomeCohortSet argument must be a data frame with the following columns: <b>cohortId</b> The unique integer identifier of the cohort <b>cohortName</b> The cohort's name <b>outcomeConceptId</b> 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.
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	<i>Count the cohort(s)</i>
-----------------	----------------------------

---

### 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 <a href="#">createConnectionDetails</a> function in the DatabaseConnector package. Can be left NULL if connection is provided.
connection	An object of type connection as created using the <a href="#">connect</a> 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: <b>cohortId</b> The unique integer identifier of the cohort <b>cohortName</b> The cohort's name <b>sql</b> The OHDSI-SQL used to generate the cohort

Optionally, this data frame may contain:

<b>json</b>	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/",
  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
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	<i>Get Cohort Inclusion Stats Table Data</i>
----------------	--

---

## Description

This function returns a data frame of the data in the Cohort Inclusion Tables. Results are organized in to a list with 5 different data frames:

- cohortInclusionTable
- cohortInclusionResultTable
- cohortInclusionStatsTable
- cohortSummaryStatsTable
- cohortCensorStatsTable

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"),
  cohortTableNames = getCohortTableNames()
)
```

## Arguments

- |                      |  |
|----------------------|--|
| connectionDetails    | An object of type connectionDetails as created using the <a href="#">createConnectionDetails</a> function in the DatabaseConnector package. Can be left NULL if connection is provided.  |
| connection           | An object of type connection as created using the <a href="#">connect</a> 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". 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 <a href="#">getCohortTableNames</a> for more details.

---

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")
)
```

## 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.

**Value**

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

---

getDataMigrator	<i>Get database migrations instance</i>
-----------------	---

---

**Description**

Returns ResultModelManager DataMigrationsManager 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

---

getRequiredTasks	<i>Get a list of tasks required when running in incremental mode</i>
------------------	--

---

**Description**

This function will attempt to check the recordKeepingFile to determine if a list of operations have completed by comparing the keys passed into the function with the checksum supplied

**Usage**

getRequiredTasks(..., checksum, recordKeepingFile)

**Arguments**

... Parameter values used to identify the key in the incremental record keeping file  
checksum The checksum representing the operation to check  
recordKeepingFile A file path to a CSV file containing the record keeping information.

**Value**

Returns a list of outstanding tasks based on inspecting the full contents of the record keeping file

---

```
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

---

 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 includes 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	<i>Used to check if a string is in lower camel case</i>
-------------	---

---

**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	<i>Is the data.frame a cohort definition set?</i>
-----------------------	---

---

**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**

<code>x</code>	A data frame
<code>warn</code>	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**

<code>x</code>	The string to evaluate
----------------	------------------------

**Value**

TRUE if the string is in snake case

---

isTaskRequired	<i>Is a task required when running in incremental mode</i>
----------------	--

---

### Description

This function will attempt to check the recordKeepingFile to determine if an individual operation has completed by comparing the keys passed into the function with the checksum supplied

### Usage

```
isTaskRequired(..., checksum, recordKeepingFile, verbose = TRUE)
```

### Arguments

...	Parameter values used to identify the key in the incremental record keeping file
checksum	The checksum representing the operation to check
recordKeepingFile	A file path to a CSV file containing the record keeping information.
verbose	When TRUE, this function will output if a particular operation has completed based on inspecting the recordKeepingFile.

### Value

Returns TRUE if the operation has completed according to the contents of the record keeping file.

---

LimitSubsetOperator	<i>Limit Subset Operator</i>
---------------------	------------------------------

---

### Description

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

Get auto generated name

### Super class

[CohortGenerator::SubsetOperator](#) -> LimitSubsetOperator



**Active bindings**

priorTime minimum washout time in days

followUpTime minimum required follow up 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\(\)](#)

**Method** `getAutoGeneratedName()`: name generated from subset operation properties

*Usage:*

```
LimitSubsetOperator$getAutoGeneratedName()
```

*Returns:* character To List

**Method** `toList()`: List representation of object

*Usage:*

```
LimitSubsetOperator$toList()
```

**Method** `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 postgres 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\_")

---

<code>readCsv</code>	<i>Used to read a .csv file</i>
----------------------	---------------------------------

---

**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](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

---

recordTasksDone	<i>Record a task as complete</i>
-----------------	----------------------------------

---

**Description**

This function will record a task as completed in the recordKeepingFile

**Usage**

```
recordTasksDone(..., checksum, recordKeepingFile, incremental = TRUE)
```

**Arguments**

...	Parameter values used to identify the key in the incremental record keeping file
checksum	The checksum representing the operation to check
recordKeepingFile	A file path to a CSV file containing the record keeping information.
incremental	When TRUE, this function will record tasks otherwise it will return without attempting to perform any action

---

runCohortGeneration	<i>Run a cohort generation and export results</i>
---------------------	---

---

**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 be 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.

### 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

Subsampled cohorts can be sampled, as with any other subset form. However, subsampling a sampled cohort is not recommended and not currently supported at this time. In the case where  $n > \text{cohort count}$  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.

**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/",
  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

	<b>sql</b>	The OHDSI-SQL used to generate the cohort Optionally, this data frame may contain:
	<b>json</b>	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
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



---

saveIncremental	<i>Used in incremental mode to save values to a file</i>
-----------------	--

---

### Description

When running in incremental mode, we may need to update results in a CSV file. This function will replace the data in fileName based on the key parameters

### Usage

```
saveIncremental(data, fileName, ...)
```

### Arguments

data	The data to record in the file
fileName	A CSV holding results in the same structure as the data parameter
...	Parameter values used to identify the key in the results file

---

SubsetCohortWindow	<i>Time Window For Cohort Subset Operator</i>
--------------------	---

---

### 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

### Methods

#### Public methods:

- [SubsetCohortWindow\\$toList\(\)](#)
- [SubsetCohortWindow\\$toJSON\(\)](#)
- [SubsetCohortWindow\\$isEqualTo\(\)](#)
- [SubsetCohortWindow\\$clone\(\)](#)

**Method** toList(): List representation of object To JSON

*Usage:*

```
SubsetCohortWindow$toList()
```

**Method** toJSON(): json serialized representation of object Is Equal to

*Usage:*

SubsetCohortWindow\$toJSON()

**Method** isEqualTo(): Compare SubsetCohortWindow to another

*Usage:*

SubsetCohortWindow\$isEqualTo(criteria)

*Arguments:*

criteria SubsetCohortWindow instance

**Method** 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\\$clone\(\)](#)

### Method new():

*Usage:*

SubsetOperator\$new(definition = NULL)

*Arguments:*

definition json character or list - definition of subset operator

*Returns:* instance of object Class Name

**Method** classname(): Class name of object Get auto generated name

*Usage:*

SubsetOperator\$classname()

**Method** getAutoGeneratedName(): Not intended to be used - should be implemented in subclasses Return query builder instance

*Usage:*

SubsetOperator\$getAutoGeneratedName()

**Method** 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)

**Method** publicFields(): Publicly settable fields of object Is Equal to

*Usage:*

SubsetOperator\$publicFields()

**Method** 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

**Method** toList(): convert to List representation To Json

*Usage:*

SubsetOperator\$toList()

**Method** toJSON(): convert to json serialized representation

*Usage:*

SubsetOperator\$toJSON()

*Returns:* list representation of object as json character

**Method** 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	<i>Upload results to the database server.</i>
---------------	---

---

**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 <a href="#">createConnectionDetails</a> 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	<i>Used to write a .csv file</i>
----------	----------------------------------

---

### 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\\_between](https://ohdsi.github.io/Hades/codeStyle.html#Interfacing_between)

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 camel-Case 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

addCohortSubsetDefinition, 3

checkAndFixCohortDefinitionSetDataTypes, 4

CohortGenerator::SubsetOperator, 7, 14, 32

CohortSubsetDefinition, 4

CohortSubsetOperator, 7

computeChecksum, 8

connect, 10, 16, 17, 19, 21, 22, 25, 29, 38

createCohortSubset, 8

createCohortSubsetDefinition, 9

createCohortTables, 10, 26

createConnectionDetails, 10, 16, 17, 19, 21, 22, 25, 29, 36, 38, 44

createDemographicSubset, 10

createEmptyCohortDefinitionSet, 11

createEmptyNegativeControlOutcomeCohortSet, 12

createLimitSubset, 12

createResultsDataModel, 13, 44

createSubsetCohortWindow, 14

DemographicSubsetOperator, 14

dropCohortStatsTables, 16

exportCohortStatsTables, 17

generateCohortSet, 18

generateNegativeControlOutcomeCohorts, 20

getCohortCounts, 22

getCohortDefinitionSet, 23

getCohortInclusionRules, 24

getCohortStats, 25

getCohortTableNames, 10, 17–19, 26, 26, 36, 38

getDataMigrator, 27

getRequiredTasks, 27

getResultsDataModelSpecifications, 28

getSubsetDefinitions, 28

insertInclusionRuleNames, 29

isCamelCase, 30

isCohortDefinitionSet, 30

isFormattedForDatabaseUpload, 31

isSnakeCase, 31

isTaskRequired, 32

LimitSubsetOperator, 32

migrateDataModel, 33

readCsv, 34

recordTasksDone, 35

runCohortGeneration, 35

sampleCohortDefinitionSet, 37

saveCohortDefinitionSet, 39

saveCohortSubsetDefinition, 40

saveIncremental, 41

SubsetCohortWindow, 41

SubsetOperator, 42

uploadResults, 44

writeCsv, 45