

# Running Cohort Diagnostics

Gowtham Rao and James P. Gilbert

2021-12-02

## Contents

<b>1</b>	<b>Configuring the connection to the server</b>	<b>1</b>
<b>2</b>	<b>Running cohort diagnostics</b>	<b>2</b>
2.1	Loading cohort references from a package . . . . .	2
2.2	Loading cohort references from WebApi . . . . .	2
2.3	Generating cohorts . . . . .	2
2.4	Executing cohort diagnostics . . . . .	4
<b>3</b>	<b>Cohort Diagnostics Output</b>	<b>4</b>
3.1	Creating a pre merged file . . . . .	4

## 1 Configuring the connection to the server

We need to tell R how to connect to the server where the data are. `CohortDiagnostics` uses the `DatabaseConnector` package, which provides the `createConnectionDetails` function. Type `?createConnectionDetails` for the specific settings required for the various database management systems (DBMS). For example, one might connect to a PostgreSQL database using this code:

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/ohdsi",
                                             user = "joe",
                                             password = "supersecret")
```

For the purposes of this example, we will use the Eunomia test CDM package that is in an SQLite database stored locally.

```
connectionDetails <- Eunomia::getEunomiaConnectionDetails()

cdmDatabaseSchema <- "main"
tempEmulationSchema <- NULL
cohortDatabaseSchema <- "main"
cohortTable <- "cohort"
```

The last four lines define the `cdmDatabaseSchema`, `tempEmulationSchema`, `cohortDatabaseSchema`, and `cohortTable` variables. We'll use the `cdmDatabaseSchema` later to tell R where the data in CDM format live. The `tempEmulationSchema` is needed only for Oracle users, since Oracle does not support temporary tables. The `cohortDatabaseSchema`, and `cohortTable` specify where we want to instantiate our cohorts. Note that for Microsoft SQL Server, database schemas need to specify both the database and the schema, so for example `cdmDatabaseSchema <- "my_cdm_data.dbo"`.

## 2 Running cohort diagnostics

### 2.1 Loading cohort references from a package

The preferred usage of cohort diagnostics is through the use of a study package. This is a dedicated R package that can be installed on a system and run. The primary reason for this is due to reproducibility, cohort definitions and resources frequently change. However, a study package can be seen as a snapshot, frozen at the time of creation and incrementally updated.

For example, the cohort diagnostics package includes a set of cohort sql and json to run on the Eunomia data set.

```
library(CohortDiagnostics)
cohortDefinitionSet <- loadCohortsFromPackage(packageName = "CohortDiagnostics",
                                              cohortToFile = "settings/CohortsToCreateForTesting.json")
```

### 2.2 Loading cohort references from WebApi

It is often desirable to perform cohort diagnostics on definitions stored in an ATLAS instance. Though this is not the preferred way of running studies (and this is certainly not the preferred method for an OHDSI network study involving multiple sites) it is possible to load references into a data frame used by cohort diagnostics.

The following code demonstrates how to create a set of cohort references from ATLAS that can be used by cohort diagnostics:

```
# Set up url
baseUrl <- "https://atlas.hosting.com/WebAPI"
# list of cohort ids
cohortIds <- c(18345, 18346)

cohortDefinitionSet <- ROhdsiWebApi::exportCohortDefinitionSet(baseUrl = baseUrl,
                                                                cohortIds = cohortIds,
                                                                generateStats = TRUE)
```

Consult the `ROhdsiWebApi` documentation for details on authentication to your atlas instance.

### 2.3 Generating cohorts

Cohorts must be generated before cohort diagnostics can be run.

### 2.3.1 Using CohortGenerator to instantiate cohorts

For example,

```
cohortTableNames <- CohortGenerator::getCohortTableNames(cohortTable = cohortTable)

# Next create the tables on the database
CohortGenerator::createCohortTables(connectionDetails = connectionDetails,
                                   cohortTableNames = cohortTableNames,
                                   cohortDatabaseSchema = "main",
                                   incremental = FALSE)

# Generate the cohort set
CohortGenerator::generateCohortSet(connectionDetails= connectionDetails,
                                   cdmDatabaseSchema = cdmDatabaseSchema,
                                   cohortDatabaseSchema = cdmDatabaseSchema,
                                   cohortTableNames = cohortTableNames,
                                   cohortDefinitionSet = cohortDefinitionSet,
                                   incremental = FALSE)
```

**Note, that the above code will delete an existing table.** However, incremental mode can be used when setting the parameter `incremental = TRUE`.

The resulting cohort table should include the columns:

`cohort_definition_id`, `subject_id`, `cohort_start_date`, `cohort_end_date`

### 2.3.2 Inclusion rules

We have the option to also generate inclusion rule statistics while the cohorts are instantiated (recommended). If we want to do this, we need to provide a folder where the inclusion rule statistics will be stored for later use.

```
inclusionStatisticsFolder <- "inclusionStats"
```

We can then use the cohort generator package to create the inclusion rule statistics for our cohorts.

```
inclusionRuleNames <- CohortGenerator::insertInclusionRuleNames(connectionDetails = connectionDetails,
                                                             cohortDefinitionSet = cohortDefinitionSet,
                                                             cohortDatabaseSchema = cohortDatabaseSchema,
                                                             cohortInclusionTable = cohortTableNames)

CohortGenerator::exportCohortStatsTables(connectionDetails = connectionDetails,
                                         cohortDatabaseSchema = cdmDatabaseSchema,
                                         cohortTableNames = cohortTableNames,
                                         cohortStatisticsFolder = inclusionStatisticsFolder,
                                         incremental = FALSE)

CohortGenerator::dropCohortStatsTables(connectionDetails = connectionDetails,
                                       cohortDatabaseSchema = cohortDatabaseSchema,
                                       cohortTableNames = cohortTableNames)

CohortGenerator::dropCohortStatsTables(connectionDetails = connectionDetails,
                                       cohortDatabaseSchema = cohortDatabaseSchema,
                                       cohortTableNames = cohortTableNames)
```

This will calculate basic inclusion rule statistics and export the results to `inclusionStatisticsFolder` for later use in cohort diagnostics.

## 2.4 Executing cohort diagnostics

Once cohort definitions are loaded and cohort tables have been populated cohort diagnostics is ready to be executed.

First we set an export folder, this is where the results will be stored.

```
exportFolder <- "export"
```

Then we execute the function (using the default settings) as follows:

```
executeDiagnostics(cohortDefinitionSet,  
                  connectionDetails = connectionDetails,  
                  cohortTable = cohortTable,  
                  cohortDatabaseSchema = cohortDatabaseSchema,  
                  cdmDatabaseSchema = cdmDatabaseSchema,  
                  exportFolder = exportFolder,  
                  inclusionStatisticsFolder = inclusionStatisticsFolder,  
                  databaseId = "MyCdm",  
                  minCellCount = 5)
```

## 3 Cohort Diagnostics Output

Once the diagnostics have completed, a zip file will have been created in the specified export folder. This zip file can be shared between sites, as it does not contain patient-identifiable information. When unzipped, the zip file will contain several .csv files that maybe easily audited. Note that cell counts smaller than 5 have been removed, as specified using the `minCellCount` argument, to ensure non-identifiability.

### 3.1 Creating a pre merged file

Assuming you completed the steps described above for one or more databases, you should now have a set of zip files, one per database. Make sure to place all zip files in a single folder.

Optionally, we can pre-merge the zip files, so we can view results in the Shiny app:

```
preMergeDiagnosticsFiles(exportFolder)
```

This file can be used in the shiny app to explore results. See the vignette “Viewing results using Diagnostics Explorer” for more details.