Running Achilles on Your CDM

Ajit Londhe

2018-10-06

Contents

1 Introduction **2**

2 General Approach **2**
2.1 SQL Only Mode 2
2.2 Logging 2
2.3 Preparation for running Achilles 3

3 Achilles Parameters (Both Modes) **3**
3.1 Staging Table Prefix 3
3.2 Source Name 3
3.3 Create Table 4
3.4 Limiting the Analyses 4
3.5 Cost Analyses 4
3.6 Small Cell Count 4
3.7 Drop Scratch Tables 4
3.8 Concept Hierarchy 4
3.9 Create Indices 4
3.10 Return Value 4

4 Running Achilles: Single-Threaded Mode **4**

5 Running Achilles: Multi-Threaded Mode **5**

6 Achilles Heel Parameters (Both Modes) **5**
6.1 Staging Table Prefix 5
6.2 Drop Scratch Tables 5
6.3 Thresholds 5

7 Running Achilles Heel: Single-Threaded Mode **6**

8 Running Achilles Heel: Multi-Threaded Mode **6**

9 Post-Processing **6**
9.1 Creating the Concept Hierarchy: Single-Threaded Mode 6
9.2 Creating the Concept Hierarchy: Multi-Threaded Mode 7
9.3 Creating Indices 7
9.4 Dropping All Staging Tables (Multi-threaded only) 7

10 Examining the Results **7**
10.1 Fetch a data frame with the Heel Results 8
10.2 View the Heel Results in HTML 8
10.3 Using AchillesWeb 8
10.4 Using Atlas Data Sources 9

11 Acknowledgments **9**

1 Introduction

In this vignette we cover how to run the Achilles package on your Common Data Model (CDM) database in order to characterize the dataset and run data quality (DQ) checks. The characterizations and DQ results can help you learn more about your dataset's features and limitations, and can then be consumed graphically using AchillesWeb or Atlas Data Sources.

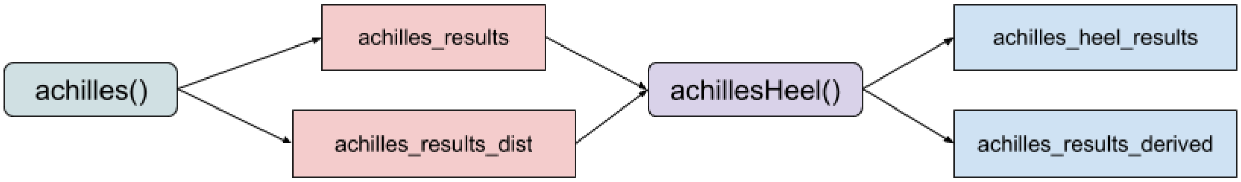
It is a best practice for all OHDSI sites to run Achilles on their CDM datasets to ensure researchers can examine study feasibility and contextualize study results.

2 General Approach

The Achilles package consists of:

1. The **achilles** function runs a set of SQL scripts to characterize the domains and concepts of the CDM.
2. The **achillesHeel** function uses the results of the achilles function to run a set of DQ scripts to evaluate the conformance and feasibility of your dataset.
3. The **createConceptHierarchy** function creates a table that summarizes all of the OMOP Vocabulary concepts. This is only necessary if using Atlas Data Sources.
4. The **createIndices** function creates table indices for the achilles tables, which can help improve query performance.
5. The **validateSchema** function compares your CDM schema against the OMOP CDM specification.
6. The **getAnalysisDetails** function provides descriptions about the full set of Achilles analyses.
7. The **dropAllScratchTables** function is useful only for multi-threaded mode. It can clear any leftover staging tables.
8. The **exportToJson** function can be used to export all Achilles results to JSON files, which is necessary for using AchillesWeb.
9. The **addDataSource** function can point a data source's JSON files to the AchillesWeb application.

The Achilles package should be run sequentially. That is, **achilles** should be run first to generate the achilles_results and achilles_results_dist tables, and then optionally, **achillesHeel** should be run next to generate the achilles_heel_results and achilles_results_derived tables.



2.1 SQL Only Mode

In most Achilles functions, you can specify `sqlOnly = TRUE` in order to produce the SQL without executing it, which can be useful if you'd like to examine the SQL closely or debug something. The SQL files are stored in the `outputFolder`.

2.2 Logging

File and console logging is enabled across most Achilles functions. The status of each step is logged into files in the `outputFolder`. You can review the files in a common text editor, or use the Shiny Application from the `ParallelLogger` package to view them more interactively.

```
ParallelLogger::launchLogViewer(logFileName = "output/log_achilles.txt")
```

2.3 Preparation for running Achilles

In order to run the package, you will need to determine if you'd like the Achilles tables and staging tables to be stored in schemas that are separate from your CDM's schema (recommended), or within the same schema as the CDM.

2.3.1 Multi-Threaded vs Single-Threaded

As the **achilles** and most of the **achillesHeel** functions can run independently, we have added a multi-threaded mode to allow for more than 1 SQL script to execute at a time. This is particularly useful for massively parallel processing (MPP) platforms such as Amazon Redshift and Microsoft PDW. It may not be beneficial for traditional SQL platforms, so only use the multi-threaded mode if confident it can be useful.

Further, while multiple threads can help performance in MPP platforms, there can be diminishing returns as the cluster has a finite number of concurrency slots to handle the queries. A rule of thumb: most likely you should not use more than 10.

In the multi-threaded mode, all scripts produce permanent staging tables, whereas in the single-threaded mode, the scripts produce temporary staging tables. In both, the staging tables are merged to produce the final Achilles tables.

2.3.2 Validate your CDM schema

Before you run **achilles**, it may be useful to verify that your CDM schema conforms to the OMOP CDM specification. This can quickly identify issues with your CDM ahead of time. Refer to the Common Data Model repo to identify which version of the CDM you intend to validate against.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

validateSchema(connectionDetails = connectionDetails,
               cdmDatabaseSchema = "cdm",
               resultsDatabaseSchema = "results",
               cdmVersion = 5.3,
               runCostAnalysis = TRUE,
               outputFolder = "output",
               sqlOnly = FALSE)
```

3 Achilles Parameters (Both Modes)

The following sub-sections describe the optional parameters in **achilles** that can be configured, regardless of whether you run the function in single- or multi-threaded mode.

3.1 Staging Table Prefix

To keep the staging tables organized, the **achilles** function will use a table prefix of "tmpach" by default, but you can choose a different one using the `tempAchillesPrefix` parameter. This is useful for database platforms like Oracle, which limit the length of table names.

3.2 Source Name

The `sourceName` parameter is used to assign the name of the dataset to the Achilles results. It is used in the Dashboard page in AchillesWeb and Atlas Data Sources. If you set this to `NULL`, the **achilles** function will try to obtain the source name from the CDM_SOURCE table.

3.3 Create Table

The `createTable` parameter, when set to `TRUE`, drops any existing Achilles results tables and builds new ones. If set to `FALSE`, these tables will persist, and the **achilles** function will just insert new data to them.

3.4 Limiting the Analyses

By default, the **achilles** function runs all analyses detailed in the `getAnalysisDetails` function. However, it may be useful to focus on a subset of analyses rather than running the whole set. This can be accomplished by specifying analysis Ids in the `analysisIds` parameter.

3.5 Cost Analyses

By default, the **achilles** function does not run analyses on the COST table(s), as they can be very time-consuming, and are not critical to most OHDSI studies. However, you can choose to run these analyses by setting `runCostAnalysis` to `TRUE`.

3.6 Small Cell Count

To avoid patient identifiability, you can establish the minimum cell size that should be kept in the Achilles tables. Cells with small counts (less than or equal to the value of the `smallCellCount` parameter) are deleted. By default, this is set to 5. However, set to NULL if you don't want any deletions.

3.7 Drop Scratch Tables

See the Post-Processing section to read about how to run this step separately

This parameter is only necessary if running in multi-threaded mode

The `dropScratchTables` parameter, if set to `TRUE` will drop all staging tables created during the execution of **achilles** in multi-threaded mode.

3.8 Concept Hierarchy

See the Post-Processing section to read about how to run this step separately

This table is only necessary if using Atlas Data Sources to consume Achilles results

The `conceptHierarchy` parameter, if set to `TRUE` will result in the concpet_hierarchy table to be created in the results schema.

3.9 Create Indices

See the Post-Processing section to read about how to run this step separately

3.10 Return Value

When running **achilles**, the return value, if you assign a variable to the function call, is a list object in which metadata about the execution and all of the SQL scripts executed are attributes.

4 Running Achilles: Single-Threaded Mode

In single-threaded mode, there is no need to set a `scratchDatabaseSchema`, as temporary tables will be used.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

achilles(connectionDetails = connectionDetails,
         cdmDatabaseSchema = "cdm",
         resultsDatabaseSchema = "results",
         vocabDatabaseSchema = "vocab",
         sourceName = "Synpuf",
         cdmVersion = 5.3,
         numThreads = 1)
```

5 Running Achilles: Multi-Threaded Mode

In multi-threaded mode, you need to specify `scratchDatabaseSchema` and use > 1 for `numThreads`.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

achilles(connectionDetails = connectionDetails,
         cdmDatabaseSchema = "cdm",
         resultsDatabaseSchema = "results",
         scratchDatabaseSchema = "scratch",
         vocabDatabaseSchema = "vocab",
         sourceName = "Synpuf",
         cdmVersion = 5.3,
         numThreads = 5)
```

6 Achilles Heel Parameters (Both Modes)

6.1 Staging Table Prefix

To keep the staging tables organized, the **achillesHeel** function will use a table prefix of "tmpheel" by default, but you can choose a different one using the `tempHeelPrefix` parameter. This is useful for database platforms like Oracle, which limit the length of table names.

6.2 Drop Scratch Tables

See the Post-Processing section to read about how to run this step separately

This parameter is only necessary if running in multi-threaded mode

The `dropScratchTables` parameter, if set to `TRUE` will drop all staging tables created during the execution of **achillesHeel** in multi-threaded mode.

6.3 Thresholds

The `ThresholdAgeWarning`, `ThresholdOutpatientVisitPerc`, and `ThresholdMinimalPtMeasDxRx` parameters can be used to configure DQ thresholds in **achillesHeel**.

- `ThresholdAgeWarning` refers to the maximum age to allow in the dataset; by default, this is 125 years of age.
- `ThresholdOutpatientVisitPerc` refers to the maximum percentage of outpatient visits allowed among all visits. This is by default set to 0.43.

- `ThresholdMinimalPtMeasDxRx` refers to the minimum percentage required of patients with at least 1 measurement, 1 condition, and 1 drug exposure. This is by default set to 20.5%.

7 Running Achilles Heel: Single-Threaded Mode

In single-threaded mode, there is no need to set a `scratchDatabaseSchema`, as temporary tables will be used.

```r
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

achillesHeel(connectionDetails = connectionDetails,
             cdmDatabaseSchema = "cdm",
             resultsDatabaseSchema = "results",
             vocabDatabaseSchema = "vocab",
             cdmVersion = 5.3,
             numThreads = 1,
             outputFolder = "output")
```

8 Running Achilles Heel: Multi-Threaded Mode

In multi-threaded mode, you need to specify `scratchDatabaseSchema` and use > 1 for `numThreads`.

```r
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")
achillesHeel(connectionDetails = connectionDetails,
             cdmDatabaseSchema = "cdm",
             resultsDatabaseSchema = "results",
             vocabDatabaseSchema = "vocab",
             cdmVersion = 5.3,
             numThreads = 5,
             outputFolder = "output",
             scratchDatabaseSchema = "scratch")
```

9 Post-Processing

9.1 Creating the Concept Hierarchy: Single-Threaded Mode

This table is only necessary if using Atlas Data Sources to consume Achilles results

In single-threaded mode, there is no need to set a `scratchDatabaseSchema`, as temporary tables will be used.

```r
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

createConceptHierarchy(connectionDetails = connectionDetails,
                       resultsDatabaseSchema = "results",
                       vocabDatabaseSchema = "vocab",
                       outputFolder = "output")
```

9.2 Creating the Concept Hierarchy: Multi-Threaded Mode

This table is only necessary if using Atlas Data Sources to consume Achilles results

In multi-threaded mode, you need to specify `scratchDatabaseSchema` and use > 1 for `numThreads`.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

createConceptHierarchy(connectionDetails = connectionDetails,
                       resultsDatabaseSchema = "results",
                       vocabDatabaseSchema = "vocab",
                       outputFolder = "output",
                       scratchDatabaseSchema = "scratch",
                       numThreads = 3)
```

9.3 Creating Indices

Not necessary for Amazon Redshift or IBM Netezza

To improve query performance of the Achilles results tables, run the **createIndices** function.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

createIndices(connectionDetails = connectionDetails,
              resultsDatabaseSchema = "results",
              outputFolder = "output")
```

9.4 Dropping All Staging Tables (Multi-threaded only)

If the **achilles** or **achillesHeel** execution has errors, or if you did not enable this step in the call to these functions, use the `dropAllScratchTables` function.

The `tableTypes` parameter can be used to specify which batch of tables to drop ("achilles", "heel", "concept_hierarchy"). Setting this to `NULL` will drop them all.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

dropAllScratchTables(connectionDetails = connectionDetails,
                     scratchDatabaseSchema = "scratch", numThreads = 5)
```

10 Examining the Results

To view the Heel results, you can use the fetchAchillesHeelResults() function or the launchHeelResultsViewer() function. The former produces an R data frame that you can then export to various formats. The latter launches a Shiny application that renders the results in an easy to consume HTML file that can be viewed in your internet browser.

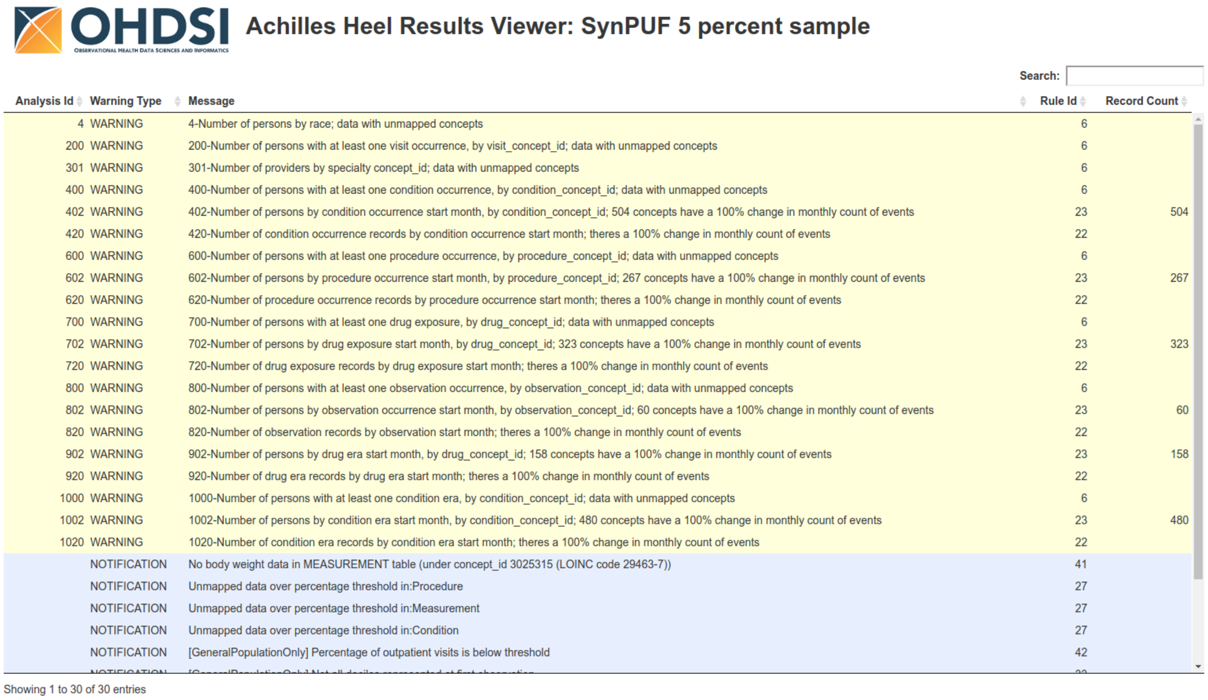
10.1 Fetch a data frame with the Heel Results

```r
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

fetchAchillesHeelResults(connectionDetails = connectionDetails,
                         resultsDatabaseSchema = "results")
```

10.2 View the Heel Results in HTML

The Heel Results Shiny Application can be useful in interacting with the Heel Results.



Achilles Heel Results Viewer: SynPUF 5 percent sample

Search:

Analysis Id	Warning Type	Message	Rule Id	Record Count
4	WARNING	4-Number of persons by race; data with unmapped concepts	6	
200	WARNING	200-Number of persons with at least one visit occurrence, by visit_concept_id; data with unmapped concepts	6	
301	WARNING	301-Number of providers by specialty concept_id; data with unmapped concepts	6	
400	WARNING	400-Number of persons with at least one condition occurrence, by condition_concept_id; data with unmapped concepts	6	
402	WARNING	402-Number of persons by condition occurrence start month, by condition_concept_id; 504 concepts have a 100% change in monthly count of events	23	504
420	WARNING	420-Number of condition occurrence records by condition occurrence start month; theres a 100% change in monthly count of events	22	
600	WARNING	600-Number of persons with at least one procedure occurrence, by procedure_concept_id; data with unmapped concepts	6	
602	WARNING	602-Number of persons by procedure occurrence start month, by procedure_concept_id; 267 concepts have a 100% change in monthly count of events	23	267
620	WARNING	620-Number of procedure occurrence records by procedure occurrence start month; theres a 100% change in monthly count of events	22	
700	WARNING	700-Number of persons with at least one drug exposure, by drug_concept_id; data with unmapped concepts	6	
702	WARNING	702-Number of persons by drug exposure start month, by drug_concept_id; 323 concepts have a 100% change in monthly count of events	23	323
720	WARNING	720-Number of drug exposure records by drug exposure start month; theres a 100% change in monthly count of events	22	
800	WARNING	800-Number of persons with at least one observation occurrence, by observation_concept_id; data with unmapped concepts	6	
802	WARNING	802-Number of persons by observation occurrence start month, by observation_concept_id; 60 concepts have a 100% change in monthly count of events	23	60
820	WARNING	820-Number of observation records by observation start month; theres a 100% change in monthly count of events	22	
902	WARNING	902-Number of persons by drug era start month, by drug_concept_id; 158 concepts have a 100% change in monthly count of events	23	158
920	WARNING	920-Number of drug era records by drug era start month; theres a 100% change in monthly count of events	22	
1000	WARNING	1000-Number of persons with at least one condition era, by condition_concept_id; data with unmapped concepts	6	
1002	WARNING	1002-Number of persons by condition era start month, by condition_concept_id; 480 concepts have a 100% change in monthly count of events	23	480
1020	WARNING	1020-Number of condition era records by condition era start month; theres a 100% change in monthly count of events	22	
	NOTIFICATION	No body weight data in MEASUREMENT table (under concept_id 3025315 (LOINC code 29463-7))	41	
	NOTIFICATION	Unmapped data over percentage threshold in:Procedure	27	
	NOTIFICATION	Unmapped data over percentage threshold in:Measurement	27	
	NOTIFICATION	Unmapped data over percentage threshold in:Condition	27	
	NOTIFICATION	[GeneralPopulationOnly] Percentage of outpatient visits is below threshold	42	
	NOTIFICATION	[GeneralPopulationOnly] Not all deciles represented at first observation	[illegible]	

Showing 1 to 30 of 30 entries

```r
connectionDetails <- DatabaseConnector::createConnectionDetails(dbms = "postgresql",
                                                                server = "localhost/synpuf",
                                                                user = "cdm_user",
                                                                password = "cdm_password")
launchHeelResultsViewer(connectionDetails = connectionDetails,
                        cdmDatabaseSchema = "cdm",
                        resultsDatabaseSchema = "results",
                        outputFolder = "output")
```

10.3 Using AchillesWeb

AchillesWeb is a lightweight web application that can be used to consume the Achilles results graphically. It is no longer updated, as development priorities have shifted towards Atlas Data Sources, but AchillesWeb can still be utilized. To connect Achilles results to AchillesWeb, the Achilles results need to be exported to JSON files and the AchillesWeb JSON file needs to point to those JSON files.

10.3.1 Exporting to JSON

The `exportToJson` function can export all of the Achilles results to JSON files that AchillesWeb can consume.

- The `compressIntoOnFile` parameter, if set to `TRUE`, will compress all of the files into one zip file for easier portability.
- The `reports` parameter can be used to export specific reports rather than all. Use the `showReportTypes` function to see all possible reports.

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/synpuf",
                                             user = "cdm_user",
                                             password = "cdm_password")

exportToJson(connectionDetails = connectionDetails,
             cdmDatabaseSchema = "cdm",
             resultsDatabaseSchema = "results",
             outputPath = "output",
             vocabDatabaseSchema = "vocab")
```

10.3.2 Adding Data Sources to AchillesWeb

AchillesWeb relies upon a JSON file to point to CDM Achilles result JSON files.

```
addDataSource(jsonFolderPath = "output",
              dataSourcePath = "achillesWeb")
```

10.4 Using Atlas Data Sources

If the Achilles results tables and the concept_hierarchy table are present in the results schema, you can point Atlas to this schema to have the results appear graphically in Atlas Data Sources.

11 Acknowledgments

Considerable work has been dedicated to provide the `Achilles` package.

```
citation("Achilles")
```

```
#>
#> To cite package 'Achilles' in publications use:
#>
#>   Patrick Ryan, Martijn Schuemie, Vojtech Huser, Chris Knoll, Ajit
#>   Londhe and Taha Abdul-Basser (2018). Achilles: Creates
#>   Descriptive Statistics Summary for an Entire OMOP CDM Instance.
#>   R package version 1.6.3.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {Achilles: Creates Descriptive Statistics Summary for an Entire OMOP CDM Instance},
#>     author = {Patrick Ryan and Martijn Schuemie and Vojtech Huser and Chris Knoll and Ajit Londhe and
#>     year = {2018},
#>     note = {R package version 1.6.3},
#>   }
```