27-05-2024

DK-DHR

LSR – STCM calculations

Annex 3

DANISH MEDICINES AGENCY

Indhold

Creation of Source To Concept Map for VNRs (May 23, 2024)

The stcm_vnr_all_dw.csv was created from multiple sources of VNR mappings to RxNorm/RxNorm extension.
Program: DW_preprocessing_usagi_maps_May24.R on *X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\DM_scripts_and_csv_files_May24*.
CSV file locations: *X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\DM_scripts_and_csv_files_May24\script3_input and /script3_output*.
Also located in *X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\05_Vocabularies\Drug mapping LMST May2024*.

STCMs/usagi map files included

- Stcm_vnr90
- Stcm_vnr90_usagi
- Stcm_mdwVNR
- Usagi_finVNR

Sources of mappings

- The 20240513_stcm_vnr90.csv file is the SourceToConceptMap.csv, which was one of the output files from the DrugMapping tool after running it with the 90% VNR list. The most detailed mappings from DrugMapping tool is to Clinical Drug concepts. *File location: X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\ DM_scripts_and_csv_files_May24\script3_input*

- Stcm_VNR_usagi.csv is one of the Usagi output files from DrugMapping in April 2024. The input to Usagi was the VNRs from drug mapping in April 2024 that were not mapped by the DrugMapping tool or were only mapped to Clinical Drug Form. *Original file: X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\05_Vocabularies\Drug mapping LMST April2024\20240410_Usagi_output\ STCM_VNR_usagi_approved_120424.xlsx*

- Stcm_mdwVNR is mappings received from Marcel de Wilde (see mail Re: Mapping of Danish registries 26-02-2024), which are from 2021. They were filtered to exclude concepts that were not valid anymore. They included also mappings to Clinical Drug Box and quantified drugs. *Original file: X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\05_Vocabularies\Drug Mapping Marcel\2021-01-04 01 DrugMapping.csv.*

- Usagi_fin_VNR. Received 9/2-2024 (see mail 'Re: OMOPing healthcare registries in the Nordics', from Anna Hammais). *Original file: X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\05_Vocabularies\Nordic_STCM\VNRfi.usagi.csv*

Preparation of mapping files

Section in program: Create a joined STCM_VNR.

- For 20240513_stcm_vnr90.csv only the records with 'Drug' as prefix in source_code were selected and VNR was derived from this column. Some additional records were then removed since the mappings seemed very uncertain: These could be identified from the 20240513_DrugMapping_output.csv file as they had length of SourceName<4 (because the unit showed up in this column by an unknown error). From the same file the records with too high source margin (>20%) could be identified for removal, however none were present.
- For Stcm_mdwVNR.csv the records for which VNR was mapped to concept_id=0 or concept_id=NA were removed. The input file had also been filtered for concept_ids that were no longer valid.
- For Usagi_fin_VNR.csv the records for which VNR was mapped to concept_id=0 or concept_id=NA were removed. Additionally, it was checked for invalid concept_ids by merging with Concept.csv and any was removed.

Combining the STCMs

1. The 20240514_stcm_vnr90.csv from DrugMapping was combined with the output from Usagi (stcm_VNR_usagi). In the DM tool there is the possibility to use a Manual Mappings file to overrule automatic mapping. This functionality was not used. Instead, VNRs in stcm_vnr90 were added to stcm_vnr90_usagi if not already present.
2. The combined map was appended with stcm_finVNR and stcm_mdwVNR and duplicate rows were removed (several present).
3. The concept_classes were determined by combining with Concept.csv on concept_id, and records were divided into 6 tables according to class: Branded Drug/Clinical Drug/Clinical Drug Form/Clinical Drug Comp/Ingredient and Precise Ingredient/ Others. 'Others' include Clinical Drug Box, Quant Clinical Drug, Quant Clinical Box. (correct next time: used opposite order).
4. Replicates of VNRs in the tables were present since there could be mappings to e.g. oral tablet as well as delayed release oral tablet. Duplicates were removed from each table by choosing slice 1 (=>our own mappings got highest priority).
5. The different tables were combined such that if any VNR was mapped to multiple concept_classes one was chosen according to the priority list: Branded Drug>Clinical Drug>Clinical Drug Form >Clinical Drug Component >Ingredient >Other. The 'Other' group was last on the list, since the method for calculating Quantity to Drug_exposure table is different in these cases.
6. The VNR column was converted to character type and leading zeros were added to get a length of 6.
7. The collated file was saved as *users\data\usagi_maps\stcm_vnr_dw.csv* on NGC cld055. It is also placed on the X-drive*: X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\ DM_scripts_and_csv_files_May24/stcm_vnr_all_dw.csv*.

LSR calculations

Program: LSR_quantity_calc_ClinicalDrugs_May24.R on *X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\DM_scripts_and_csv_files_May24*

Literature used:

PowerPoint Presentation (ohdsi.org)

Drug Domain abstract (ohdsi.org)

Calculation of Quantity (for multiplication with number of packages, APK)

1. Laegemiddeloplysninger from LSR was left joined with the stcm_vnr_all_dw.csv on VNR in order to get the matching drug_concept_ids. Only 'Branded Drug', 'Clinical Drug', 'Branded Drug Comp' and 'Clinical Drug Comp' concept_ids were added. Quantity is not going to be calculated for VNRs mapped to Ingredient level (mail from Maxim Moinat 12/2-2024: Re: Drug quantity calculations). Also calculation of quantity for quantified drugs was skipped, since the method for this is different. Thus only a smaller part of VNRs could get a quantity.
2. This table was joined with the OMOP Drug_strength table to add information on the strengths for the mapped-to drug concepts. This information defines the method for calculation of quantity.
3. Calculation methods for 4 different scenarios:
    a. In Drug_strength only the amount_value and amount_unit_concept_id is available and tells the strength as e.g. mg per tablet. SIZEUNIT in Laegemiddeloplysninger is ST, HT …(pieces, vials..) or similar indications of an item =>**Quantity =PACKSIZE**.
    b. Amount_value and amount_unit is still the only available information in Drug_strength. SIZEUNIT is MG or RG (ug) => **Quantity=PACKSIZE*factor/amount**_value, where factor converts from SIZEUNIT to amount_unit_concept_id. This will give the number of e.g. tablets.
    c. In Drug_strength, the numerator_value, numerator_concept_id and denominator_unit concept_id is available (no denominator value). It tells the strength as e.g. X mg/ml. The SIZEUNIT corresponds to the denominator_unit (it is e.g. ul) =>**Quantity=PACKSIZE*factor**, where factor converts from SIZEUNIT to denominator_unit_concept_id.
    d. Again the the numerator_value, numerator_concept_id and denominator_unit concept_id is available. The SIZEUNIT corresponds to the numerator value (it is e.g. X g) =>Quantity=**PACKSIZE*factor/numerator_value**.
    e. There are more options, but only these scenarios were considered in this round of calculations.

NOTE 1: In the calculations used for the deadline May 2024, no quantity was calculated if the drug_concept_id had both of numerator_unit_concept_id and denominator_unit_concept_id equal to 8576 (milligram), since it was uncertain if PACKSIZE (with SIZEUNIT=G or MG) would correspond to numerator or denominator. However, for next upload this can in some cases be determined from VOLTYPECODE (if PACKSIZE=VOLUME):

'GP' (g (præparat)) => PACKSIZE is denominator (scenario 2a), and SIZEUNIT=G is thus compared with denominator_unit_concept_id. Q=PACKSIZE*F.
'GA' (g (aktivt stof))=> PACKSIZE is numerator (scenario 2b), and SIZEUNIT=G is thus compared with numerator_unit_concept_id. Q=PACKSIZE*F/numerator_value
NOTE 2: Also if SIZEUNIT is 'G' and numerator_unit/denominator is mg/ml it is uncertain if PACKSIZE is numerator or denominator (it could mean X gram of a creme or X gram of the active substance). For the May 2024 deadline, no quantity was calculated in this situation.

Calculations of Duration for End_date in Drug_exposure

If VOLTYPECODE is DW, DDK…(number of defined daily doses) the duration can be found as VOLUME*APK (or VOLAPK in Epikur). However, duration can also be found from STRNUM*PACKSIZE/DDD, where DDD is the defined daily dose e.g. in mg for a specific ATC code (this list was obtained from ATCDDD - Order (fhi.no)) This method was used for VNRs where all necessary information was available.

1. The conversion factors needed for calculating amount in mg from STRNUM and PACKSIZE were found for the different combinations of STRUNIT and SIZEUNIT.
2. Amount in mg was calculated as STRNUM*PACKSIZE* factor.
3. The DDD list was filtered to include only DDD values with a unit of mg or g and with information on the route of administration. If there was a value in the Comment field the record was removed, since it often discriminated between duplicates (e.g. long duration vs short duration) and it was not possible to choose the correct one. If the DDD value was given in g, it was converted to mg.
4. The DDD list was joined with the VNR list on ATC code and route of administration (WHOADMVEJ in Laegemiddeloplysninger).
5. Duration1 was calculated as amount in mg divided by the DDD value.

The table with VNR, quant1 and duration1 was uploaded to the dk_registries_preprocessed_data DB. Notice: It is important to use lowercase for names of new variables, as Rabbit in a Hat converts everything to lowercase, and there needs to be agreement between RiaH and actual data.

NOTE: vnr in epikur is a character variable of length 6 (including leading zeros) => Make sure that vnr in lsr_calculations is similar in format.

Method for creating the input files to EHDEN Drug Mapping tool

(For internal use, to remember how it was done)

GitHub - EHDEN/DrugMapping: Tool for mapping drugs in a source vocabulary to RxNorm(Extension) concepts in the OMOP-CDM

Input data: The Darwin data (FSEID-00007025) are used.

Scripts: Besids on NGC, the scripts as well as input and output files are saved in X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\DM_scripts_and_csv_files

First set of input files

The files are created in the program LSR_DrugMapping_step1.Rmd followed by manual mappings of dose form and units. The files are used as input to the program LSR_DrugMapping_input_dw .R (where the final input files to DrugMapping tool are created).

Description of files

1. **The LSR90_for_DrugMapping_May24.csv** file contains the list of VNRs (6188) that cover 90% of records in epikur. Besides selected variables from laegemiddeloplysninger it also contains the vcount variable with the frequencies in epikur.
2. **LSR_units_for_DrugMapping_darwin.csv** contains manual mappings from STRUNIT (strength units) to OMOP concepts (UCUM vocabulary). They cover units used for the 90 % list of VNRs. A column with conversion factor from source unit to target unit is included, and the Freq column shows the counts of VNRs in Laegemiddeloplysninger with each unit.
3. **LSR_form_for_DrugMapping_darwin.csv** contains manual mapping from DOSFORM (dosage form) to OMOP standard concepts (RxNorm). They cover DOSFORM values used for the 90 % list of VNRs. A column prioritizes between concepts_ids if multiple are provided for one DOSFORM value (i.e. several suggested options). Priority = 0 means highest priority. The FormFreq columns shows the counts of VNRs in Laegemiddeloplysninger with each dose form.

File locations

X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\07_Documentation\DM_scripts_and_csv_files\script2_input

Creation of final DM input files

R program on cld056: /users*/home/DAC_folder/DrugMapping/ LSR_drugMapping_input_dw.R*

Input files to R program

Besides the files above, a list with names and strengths of each active ingredient for each DrugID is used.

The location of this file is: *X:\1-DIR\DAC\2_Process & Methodology\OMOP DAC EHDS\3. Ydelser\OMOP DK Registers\05_Vocabularies\List of DrugID LMST\Alle aktive substanser på alle drugid v2.xlsx.*

This list was obtained from LMST (David Alexander, dfg@dkma.dk).

The list can be joined with Laegemiddeloplysninger based on DrugID. Notice, there is a one-to-many relation between DrugID and VNR. The strength of ingredients is shown in vægt.eller.volumen column.

Creation of Generic Drug File

1. To improve DrugMapping output, STRNUM in the subset of Laegemiddeloplysninger was multiplied by 0.001 if strength unit was originally in micrograms to get the strength in mg (originally thought this was not necessary). This table was merged with the drug ingredient list based on matching DrugIDs.
2. Two indicator columns were made in order to subsequently create a new strength column. One indicator column tells if the VNR has more than one ingredient. Another shows if the same ingredient is repeated multiple times for a single VNR. A third added column contains the sum of the vægt.eller.volumen column over each VNR.
3. New Strength (STRNUM_NEW) and strength unit (STRUNIT_NEW) columns were created to get the strength of each ingredient.

- The original STRNUM was used whenever possible (i.e. when available for single-component drugs).
- If STRNUM was not available for single component drugs, the summed value of vægt.eller.volumen was used (summation was used, since a single-ingredient drug could appear with the same ingredient in multiple records with different concentrations -summing up to the full strength).
- For multi-ingredient drugs, vægt.eller.volumen from the drugID list was always used.

XX VNRs could not get a strength value (e.g. multi-component drugs without information in the ingredient list)

4. The final columns in Generic Drug File are:

New name	**Source variable**
SourceCode	VNR (from Laegemiddeloplysninger)
SourceName	Concatenate (PNAME, STRENG) (from Laegemiddeloplysninger)
SourceATCCode	ATC (from Laegemiddeloplysninger)
SourceFormulation	DOSFORM (from Laegemiddeloplysninger)
SourceCount	vcount (calculated frequency of the VNR in epikur)
IngredientCode	Substans (from ingredient list)
IngredientName	Substans (from ingredient list)
IngredientNameEnglish	Substans.EN (from ingredient list)
Dosage	STRNUM_NEW (from Laegemiddeloplysninger+ingredient list)
DosageUnit	STRUNIT_NEW (from Laegemiddeloplysninger+ingredient list)

Creation of Unit Mapping file

1. Calculate the frequency of VNRs in Epikur with each unit type->unit_counts.

2. The final columns in Unit Mapping file are:

New name	**Source variable**
SourceUnit	STRUNIT (from Laegemiddeloplysninger)
RecordCount	unit_counts (calculated from vcounts)
DrugCount	Freq (from LSR_units_for_DrugMapping.csv)
Factor	Factor (from LSR_units_for_DrugMapping.csv)

TargetUnit	Concept.code (from LSR_units_for_DrugMapping.csv)

Creation of Form Mapping file

1. Calculate the frequency of VNRs in Epikur with each dosage form->form_counts.
2. The final columns in Form Mapping file are:

New name	**Source variable**
DoseForm	DOSFORM (from Laegemiddeloplysninger)
RecordCount	form_counts (calculated from vcounts)
DrugCount	FormFreq (from LSR_form_for_DrugMapping.csv)
Priority	Priority (from LSR_form_for_DrugMapping.csv)
Concept_id	Concept (from LSR_form_for_DrugMapping.csv)
Concept_name	Concept.name (from LSR_form_for_DrugMapping.csv)

Creation of Ingredient English Translation file

1. It is based on the subset of Laegemiddeloplysninger merged with the ingredient list.
2. The final columns in Form Mapping file are:

New name	**Source variable**
IngredientName	Substans (from ingredient list)
IngredientNameEnglish	Substans.En (from ingredient list)
Comments	empty char column (NA)

It contains XX different ingredients for the 6188 VNRs.

Save as CSV files

This is a very critical step (!) since DrugMapping tool is sensitive to quotation marks etc.

1. First remove any comma in the string variables (SourceName, IngredientCode, IngredientName, IngredientNameEnglish).
2. Save as comma separated files with the options row.names=FALSE and quote=FALSE.

Drug mapping

-The next step is to use DrugMapping tool (the Windows installation on CLD055). See 'Documentation of Drug Mapping and Usagi'. The input files were initially transferred to the windows node by: scp */path/inputfile*.csv susbru@CLD055-0013:Desktop.

-Handling of the output files from DrugMapping is described in LSR_STCM_calculations.docx

Tables

Table 1. Calculations of Quantity and End_date in Drug_exposure table

Tables and input variables		Calculations		
Epikur (LSR)	**lsr_calculations**	**Quantity**	**End date**	**Priority of methods**
APK	quant1	quant1*APK		
Volapk, Eksd[1]			Eksd+Volapk-1	1
APK[2]	duration1		Eksd+duration1*APK-1	2
Eksd			Eksd+29	3

[1]Can be used if VOLTYPECODE in (DW, DWG, DDK…), i.e. number of defined daily doses

[2]Can be used if STRNUM, PACKSIZE and DDD values are available

Variables in Epikur

APK: number of packages

VOLAPK: APK*VOLUME

Eksd: expedition day (drug exposure start date)

LSR calculation: quant 1 (see details on also page 5)

Quant1 is calculated for VNRs mapped to Clinical Drugs, Branded drugs and Clinical drug components. See sheet Quant1 calculations.

The calculations are based on information in OMOP table Drug_strength and on PACKSIZE in Laegemiddeloplysninger

Quant1 could be number of tablets (where strength is given in Drug_strength)

LSR calculation: duration 1

Duration is calculated for all VNRs as amount in mg divided by DDD in mg/day.

Amount is calculated as STRNUM*PACKSIZE* conversion factor, where STRNUM is strength from laegemiddeloplysninger. See sheet Mg conversion.

DDD is external data with defined daily doses for ingredients based on ATC code. They are merged with VNR list on ATC code and route of administration.

DDD values are only used if route of administration matches with WHOADMVEJ in laegemiddeloplysninger and if the dose is reported as mg, g or µg.

Quality control

Comparison of duration1 with duration from Volapk (or VOLUME*APK in laegemiddeloplysninger) if both calculations are present.

About 300 VNRs had >10 % discrepancy between the two durations.

Results: There are some errors in Packsize in laegemiddeloplysninger.

Also, calculations for drugs with STRUNIT= RGT and MGT seems not useful, as there is high discrepancy with duration from VOLUME

Table 2. Calculations of Quant1

Scenario 1: Only amount_value and amount_unit_concept_id is present in Drug Strength

Scenario 1a: Packsize is number of items, which each has the strength mentioned in amount value

Amount_unit	Amount_value	Sizeunit	Packsize	Factor (F)	Quant1 = Packsize
mg (8576)	100	ST, HT, HS, AS	50	1	50
ug (9655)	500	ST, HT, HS, AS	50	1	50
unit (8510)	8	ST, HT, HS, AS	50	1	50

Scenario 1b: Packsize is the total number of mg=>number of items needs to be calculated

Amount_unit	Amount_value	Sizeunit	Packsize	Factor (F)	Quant1 = [Packsize*F] / amount_value
mg (8576)	100	MG	500	1	5
mg (8576)	100	RG	500000	0,001	5
mg (8576)	100	G	0,5	1000	5

Scenario 2: Only numerator value and unit and denominator_unit_concept_id is present

Scenario 2a: Sizeunit (e.g. L) is similar to denominator unit (ml)

numerator unit	numerator value	denominator unit	Sizeunit	Packsize	Factor (F)	Quant1 = Packsize*F
mg (8576)[1]	0,1	mg (8576)	mg	100	1	100
mg (8576)	0,1	mg (8576)	G	0,1	1000	100
mg (8576)	0,1	mg (8576)	KG	0,0001	1000000	100
mg (8576)	10	ml(8587)	ml	5	1	5
mg (8576)	10	ml(8587)	LT	0,005	1000	5
mg (8576)	200	act(45744809)	DO(S)	5	1	5
U (8510)	0,4	ml(8587)	ml	5	1	5

[1]For mg/mg it is just assumed that Packsize is the denominator

Scenario 2c: Drug_strength strength is MG/Act

numerator unit	numerator value	denominator unit	Sizeunit	Packsize	Factor (F)	Quant1 = [Packsize*F]/ numerator value
mg (8576)[2]	10	ml(8587)	G	1	1000	100
mg (8576)[3]	200	act(45744809)	G	10	1000	50

[2](100 ml of 10 mg/ml=>1 G)

[3](50 ml of 200mg/ml =>10 G)

Table 3. Calculations of amount in MG for all VNRs based on STRNUM and PACKSIZE. Finding the conversion factor

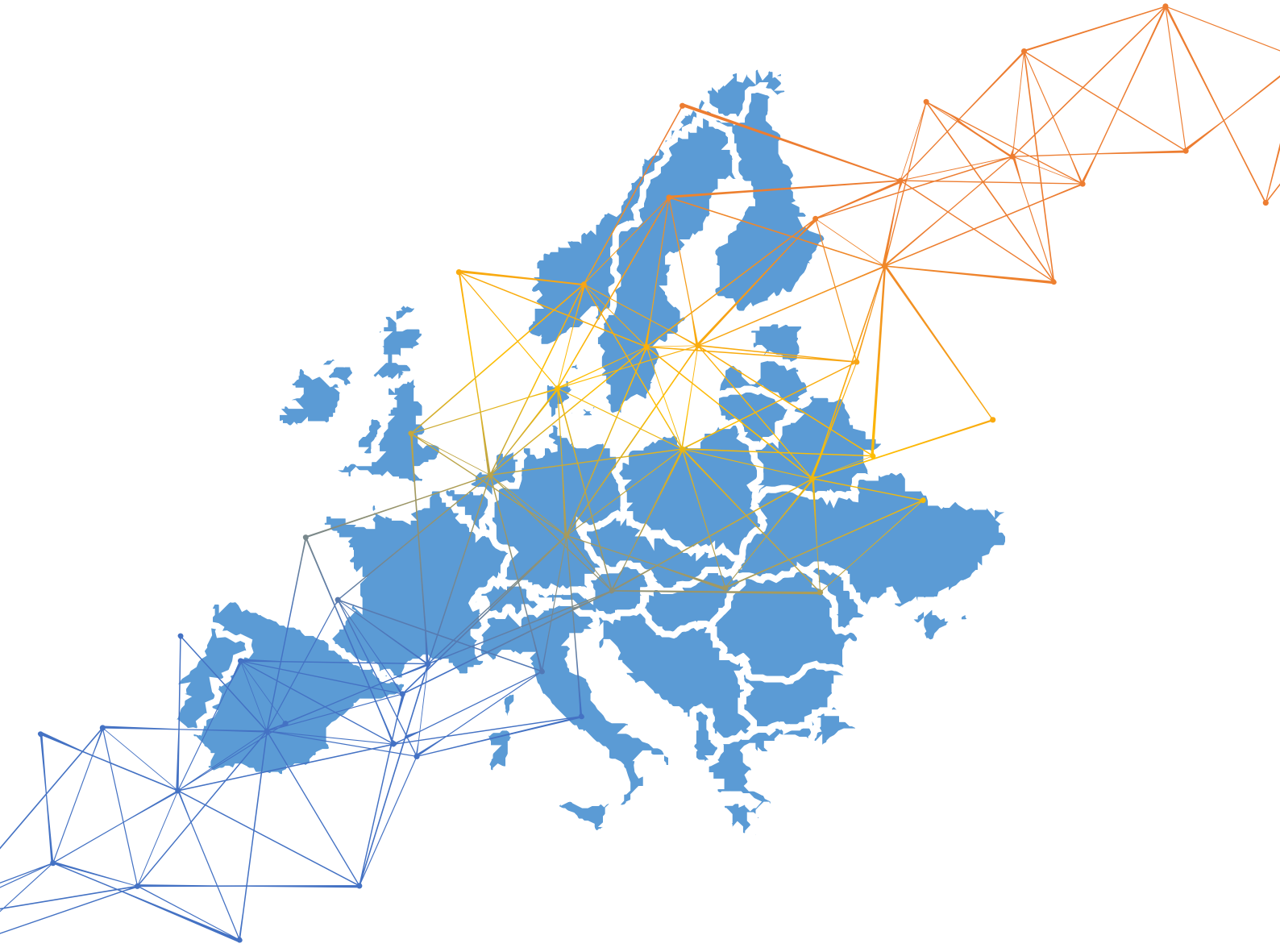
Example	STRUNIT	STRNUM	SIZEUNIT	PACKSIZE	Factor	MG
100 mg pr stk *50 stk	MG	100	ST, FL, HT, HS, AS, AM	50	1	5000
	RG	100000	ST, FL, HT, HS, AS, AM	50	0,001	5000
	G	0,1	ST, FL, HT, HS, AS, AM	50	1000	5000
100 mg/ml *50 ml	MGM	100	ML	50	1	5000
	MGM	100	LT	0,05	1000	5000
	RGM	100000	ML	50	0,001	5000
	GML	0,1	ML	50	1000	5000
	GML	0,1	LT	0,05	1000000	5000
	GL	100	ML	50	1	5000
100 mg/dose *50 doses	MGD	100	DO, DOS	50	1	5000
	RGD	100000	DO,DOS	50	0,001	5000
	GD	0,1	DO,DOS	50	1000	5000
100 mg/g*50 G	MGG	100	G	50	1	5000
	MGG	100	KG	0,05	1000	5000
	RGG	100000	G	50	0,001	5000
100 mg/ml *50 ml*	W/V	0,1	ML	50	1000	5000
(10% solution)**	PC	10	ML	50	10	5000
	PC	10	LT	0,05	10000	5000
100 mg/G	PC	10	G	50	10	5000
	PC	10	KG	0,05	10000	5000

*10 mg/ml =1% =0.01 W/V

**100 mg/ml =10% =0.1 W/V

ETL-ENGINE

Version 1.2



ROOK-IT
IT Consulting Services

TABLE OF CONTENTS

Rook-IT is a specialized consultancy focused on data Transformation and High-Performance Computing (HPC). The primary objective is to assist users in enhancing efficiency, flexibility, and security by designing, implementing, and managing secure systems.

Rook-IT has a strong reputation for developing secure systems tailored to handle sensitive data processing. We as EHDEN SME provide users with technical experts to support for the standardisation of their data and mapping to the OMOP Common Data Model.

More info on: WWW.Rook-IT.COM

1. INTRODUCTION

The ETL-Engine is used to transform a random database of health information into the OMOP (Observational Medical Outcomes Partnership) CDM (Common Data Model) (1). It attempts to make these data transformations simple by leveraging the output of the existing OHDSI (Observational Health Data Sciences and Informatics) tools (2) of Rabbit in a Hat and Usagi, allowing data owners to focus on the data and not so much on how to transform it.

By utilizing Rabbit in a Hat to define how to perform the transformation of a source database into a destination OMOP CDM database, the data modelling becomes visual within a tool which is already familiar to those within the community of OMOP CDM data modelers. Therefore, instead of adding a new tool and a new interface, the functionality of Rabbit in a Hat, which performs modeling and ETL-documentation functions, and the ETL-Engine, which does the transformation, are both merged into a single interface.

Using an existing tool obviously comes with the limitations imposed by the fact that Rabbit in a Hat isn't designed to orchestrate an ETL process, but merely to document how an ETL process should be created. Rabbit in a Hat can be used out of the box to describe the data path in simple table to table and field-to-field mappings. However, any mappings that require logic to transform the data, for example scenarios that combines several fields into one, splits a field into several data components, or simply maps data using Usagi files or the OMOP vocabulary, must use a standardized way to communicate this to the ETL-Engine. For this purpose, a mapping language has been defined, aiming to provide a simple syntax which is capable of covering a wide range of use cases.

This document will focus on how to install, configure, and use the ETL-Engine. The first part will guide you through the steps to get the ETL-Engine installed, configured and ready to map data. This is followed by descriptions of how to use Rabbit in a Hat to map tables and fields, then finally an explanation of the mapping language used to interact with the ETL-Engine from Rabbit in a Hat.

2. STARTING WITH ETL-ENGINE

The ETL-Engine has three different end-point drivers that each allow you to read from a source database and output to an OMOP CDM:

- CSV
- MariaDB / MySQL
- PostgreSQL

The source and CDM endpoints can be mixed as you see fit.

Before going through the steps described in this guide, you will need a working docker installation to run the ETL-Engine.

The ETL-Engine is distributed in a tar.gz package containing a docker image, configuration templates, and the necessary bash scripts to install, upgrade, and run the ETL-Engine. Distributing the ETL-Engine as a docker image allows the engine and all its dependencies to be delivered in a single package and avoid the hassle of resolving those dependencies during installation. When you have docker installed on your platform, you can unzip the package in the directory of your choosing and install the ETL-Engine image using the following commands:

```
$ tar xzf etl-engine.tar.gz
$ cd etl-engine/bin
$ ./install-etl-engine.sh
```

The `install-etl-engine.sh` command may need to be run as root. If your user is not a member of a group designed to run docker commands. If your account does not have the required permissions, then the command will fail and display a warning.

Running the above commands will install the ETL-Engine image in docker. When the ETL-Engine package is unzipped, in addition to expanding the bin directory and the install script, it will also generate a folder structure which is prepared to hold the different data sources that can be used for the transformation. The data sources that can be used to perform the transformation are as follows:

- The source database to be transformed, this could be CSV format files (which must be stored in the folder structure) or a supported RDBMS (which is merely configured in the configuration file).
- OMOP CDM vocabulary in the destination OMOP CDM database used for mappings. This source is expected to be in a supported RDBMS.
- Usagi files for specific mappings. These are CSV format files which must be stored in the folder structure.
- Rabbit in a hat file describing how the transformation should be performed. This is a gzipped json file which must be stored in the folder structure.

The folder structure looks as follows, and each folder or file has the following description:

```
bin/install-etl-engine.sh # Install script
bin/run-etl-engine.sh     # Runs the ETL-Engine with the current
                          # configuration
data/etl-engine.docker    # Docker image containing the ETL-Engine
etc/etl-engine.d/db       # Location to store CSV files. Only used
                          # if the database is stored in CSV files
```

```
etc/etl-engine.d/usagi     # Location to store Usagi files.
etc/etl-engine             # Location to store Rabbit in a Hat file
etc/etl-engine/etl.conf    # The ETL-Engine configuration file
logs                       # Location where log files will be saved
```

The next step, before we can successfully run the ETL-Engine, is to prepare the ETL-Engine configuration file. The configuration file is explained in section 3, "ETL-Engine Configuration File" which follows this section. Another requirement is to provide a valid Rabbit in a Hat file, this is explained in the fourth section "Mapping a Data Source". When all these prerequisites are in place, then the run script for the ETL-Engine container can be executed as follows:

```
$ cd etl-engine/bin
$ ./run-etl-engine.sh
```

As per the `install-etl-engine.sh` command, the `run-etl-engine.sh` may also need to be run as root, depending on how your docker is set up. When all the files required for the mapping are present, the ETL-Engine will perform the ETL to its best and output any errors and warnings to the log directory. In the log directory a log file will be present for the ETL-Engine itself, named: `etl-engine.log`, plus one log file for each non-vocabulary CDM table, named after the table in question. These log files will contain information specific to the parsing of the commands which define the transformation of data for that specific table as well as any rows that failed when the transformation was executed.

3. THE ETL-ENGINE CONFIGURATION FILE

The configuration file contains information about where the ETL-Engine extracts data from, how it will be transformed, where it will be loaded to, where log files will be stored and finally, if any automation steps should be performed. Correspondingly, the configuration file consists of five blocks:

- Source database – pointing to the source database or input CSV files which contain the dataset that should be transformed into the OMOP CDM format. The different supported types of source databases are listed in section 2.
- Cdm database – pointing to the destination database which must contain the OMOP CDM. The OMOP CDM can be configured to be installed in the automation section. The destination database is also the database used for vocabulary mappings, vocabulary mappings can likewise, be loaded as part of the automation.
- Rabbit configuration – Configuration files from OHDSI Rabbit tools, such as Usagi and Rabbit in a Hat.
- Automation – configuration block to enable and disable automation steps.
- Logging – Log level and where log is stored.

The `etl.conf` file that ships with the ETL-Engine contains a demonstration of how a configuration can look. As stated above, the configuration file consists of five parts. Throughout the remainder of this section, we will review each block in turn.

3.1 SOURCE DATABASE

The source database block defines which driver to use to collect the source database, with the valid driver option being `csv`, `postgresql` and `mariadb`. The 'connection-string' property defines how the ETL Engine will connect to the source database. In this example we are using the `csv` driver, which simply requires the path to the csv file. However, as both `postgresql` and `mariadb` are database drivers, they will require a database-specific connection string which are documented in section 3.2 Cdm Database. In this section the csv driver is connection string is documented, as this is the default setting in the configuration file. However, both csv and database drivers are valid for source and cdm database blocks within the configuration file.

```
source database {
  driver: "csv"
  connection string: "/etc/etl-engine.d/db/"
}
```

If using a csv database, then the relevant csv files simply need to be dropped into the `etc/etl-engine.d/db` folder in the prepared folder structure. Each path in the configuration file can be either relative or absolute. The relative path will be relative to the `etl.conf` file. As long as you are running the ETL-Engine from within its container it is recommended to use the pre-entered paths throughout the configuration template and the prepared folder structure.

3.2 CDM DATABASE

The CDM database block defines which driver to use to connect to the destination database. This database is also expected to hold the OMOP CDM vocabulary, which can be used to map data from the source database.

The database connection string has several properties, which must be provided in the form of key value pairs. All databases support the following keys:

- `user` – the username to connect to the database with.
- `password` – the password for the user that is connected to the database with.
- `host` – the resolvable hostname or ip address of the database.
- `port` – the port used to connect to the database.
- `dbname` – the name of the database connected to.

Some databases support additional keys. These are:

- postgresql
    - options – postgresql specific parameters can be passed using this key in the connection string. E.g. --search_path which can be used to set a specific schema.

The default configuration file contains the following `cdm database` block:

```
cdm database {
  driver: "postgresql"
  connection string: "user=postgres host=<hostname> port=5432
dbname=postgres options='--search_path=omop_cdm'
password=<password>"
  use transaction: false
  vocabulary cache size: 1000
  limit tables {
    table: person
    table: condition_occurrence
  }
}
```

As with the source database, a driver is chosen for the chosen CDM database type and the connection string for that database is defined. The connection string format for RDBMSes is almost identical across platforms with very few differences. E.g. as PostgreSQL supports schemas this can be configured using the options parameter as shown above. MariaDB on the other hand, doesn't support schemas or any other specialized features, so the options parameter isn't available. An example of a PostgreSQL connection string has been provided in the `etl.conf` template, with dummy values for the username, port, dbname, and options. If you are using a PostgreSQL for your database, you will need to update the connection string with the correct username, hostname, port, dbname, options, and password. More information on the PostgreSQL and MariaDB database connection strings can be collected from those vendors.

There are a few additional properties which can be set on the CDM database to configure the behaviour during the ETL. The 'use transaction' property can be set to true to perform bulk inserts into the OMOP CDM in commits of 10,000 inserts at a time. This increases performance on the database inserts by a factor of 4, however, this speed increase comes at a price; if one record fails in a block of 10,000 inserts, all 10,000 records will fail to be inserted. This property is not mandatory and if not defined then it will default to *false*. The property 'vocabulary cache size' can be used to define the number of records looked up from the OMOP CDM vocabulary that should be cached. The property is not mandatory and if not defined then a default value of 1000 will be used.

Finally, the 'limit tables' block can be defined to specify a limited set of tables to be included when the ETL engine. As only tables named within the block will be handled by the ETL-Engine, if a table which other tables

depend on, e.g. the person table, is not defined in the list then the other dependent tables which are in the list will not be able to load. The 'limit tables' block can, however, be useful when debugging the load of a specific table.

3.3 RABBIT CONFIGURATION

The rabbit configuration block points to the locations of files generated by the OHDSI rabbit tools: Usagi and Rabbit in a Hat. The 'usagi directory' property points to a directory in which Usagi files are stored. If you are not using Usagi for your mapping, the directory can be empty and can be skipped from the configuration file.

```
rabbit configuration {
  usagi directory: "/etc/etl-engine.d/usagi/"
  hat file: "ScanReport.xlsx.json.gz"
}
```

The 'hat file' property must point to a Rabbit in a Hat output file. This file must contain the mapping from the source database into the CDM database. More on how to create this Rabbit in a Hat file can be found in the "Mapping Language" section of this document. The hat file is intended to be stored next to the `etl.conf` file in the folder structure, i.e. in the /`etc/etl-engine` folder.

3.4 AUTOMATION

The fourth block of the configuration file is the automation block. This block is used to enable and disable the different automation parts of the ETL-Engine.

```
automation {
  pre {
    create omop cdm: once
    populate vocabulary: once
    clean omop data: true
    vocabulary directory: /etc/etl-engine.d/vocab/
    vocabulary backup: /etc/etl-engine.d/vocab.backup/
  }

  post {
    add preceding visit ids: true
    create condition eras: true
    create drug eras: true
    create dose eras: true
  }
}
```

Within the automation block itself, there are two blocks, a pre and a post block. The pre block contains automation steps that are executed before the mapping is done and properties that relate to these steps. Two pre-automation steps exist:

- Setting up the OMOP CDM database, this is done based on the version in the Rabbit in a Hat file. The 'create omop cdm' key has three options: "always" will build or rebuild the database on every run, "once", will only build the database if it doesn't already exist, and "never" will not run the automation for building the database. The default value for "create omop cdm" is "never".
- Populating the vocabulary tables with a zip file from Athena. The zip file must be located in the directory path defined by `vocabulary directory`. The 'populate vocabulary' key has three options: "always" will build or rebuild the vocabulary on every run, "once" will build or rebuild the vocabulary every time a vocabulary file is present in the directory defined by the `vocabulary directory`. By default the vocabulary directory is /etc/etl-engine.d/vocab/. When the vocabulary file is processed it will be moved to the directory defined by the `vocabulary backup`. By default, the vocabulary backup is /etc/etl-engine.d/vocab.backup/. Finally, the "never" option will not run the automation for populating the vocabularies, this value is by default set to "never".

To remove any pre-existing health data in the OMOP CDM, the "clean omop data" key can be set to "true"; this will delete all content of the OMOP CDM outside of the standardized vocabularies before performing the database transformation and load. This property is not mandatory and if missing it will be considered "false".

The post block contains switches to enable or disable automatic population of fields and tables in the OMOP CDM. Currently the following four post-automation steps can be enabled:

- Add preceding visit IDs to both the visit_occurrence and visit_detail tables. A value of "true" will enable this setting, while "false" will disable it. The default value is 'false'.
- Populate the condition_era table based on the data in the condition_occurrence table. A value of "true" will enable this setting, while "false" will disable it. The default value is 'false'.
- Populate the drug_era table based on the data in the drug_exposure table. A value of "true" will enable this setting, while "false" will disable it. The default value is 'false'.
- Populate the dose_era table based on the data in the drug_exposure table. A value of "true" will enable this setting, while "false" will disable it. The default value is 'false'.

None of these automation lines are required, if they are not present in the configuration file, the default values for each will be used, as defined above.

3.5 LOGGING

The final block of the configuration file is the logging block, which defines where logs are written to and what log level is being used. Five log levels exist: `TRACE`, `DEBUG`, `INFO`, `WARN`, and `ERROR`. The location for the log directory is the default and when using the docker container to run the ETL-Engine it is best practice to not change this setting.

```
logging {
  log level: "INFO"
  log directory: "/var/log/etl-engine/"
}
```

4. MAPPING A DATA SOURCE

The ETL-Engine uses data generated by Rabbit in a Hat to define how it will operate. This section will look at how the semantics of data mapping are defined within the ETL-Engine, and how the different artifacts used in Rabbit in a Hat are mapped. Rabbit in a Hat has six artifacts that are being consumed by the ETL-Engine when performing a mapping. They are:

- Source tables
- CDM tables
- Source to CDM table maps
- Source fields
- CDM fields
- Source to CDM field map

The ETL-Engine uses neither stem tables nor completeness on mappings. Instead, it builds its own internal tables for the mapping, and simply checks whether all mappings on a table-to-table mapping are valid. If so, it will attempt to map the source table into the CDM table.

In this section, we will start by reviewing the table mapping and mapping definitions, then we will review the fields and source to CDM mappings, and finally we will finish off with explaining the ETL-Engine mapping language. To be able to demonstrate how real-life mappings can be done, the synthetic mass (A synthetic data set of health data which comes from Massachusetts – In the rest of this document it is referred to as the Synthmas data set) data set (3) is being employed as a source data set in all the examples.

4.1 TABLE MAPPING

The mapping of tables is about defining which source tables deliver input into which CDM tables. At times this may seem a trivial task, but at other times it can be quite complex to define how the data should be mapped. At the heart of it, the very simplest mapping will look like Figure 1.

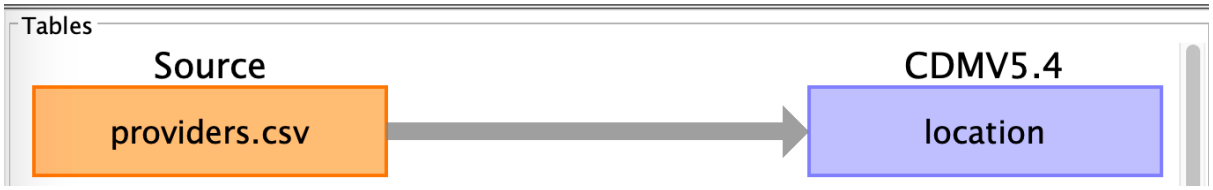


Figure 1 - Simple source table to CDM table mapping

This particular transaction will make the ETL-Engine copy records from the providers.csv table into the location table, on the condition that all mandatory fields in the location table are being mapped, more about that in section 4.2.

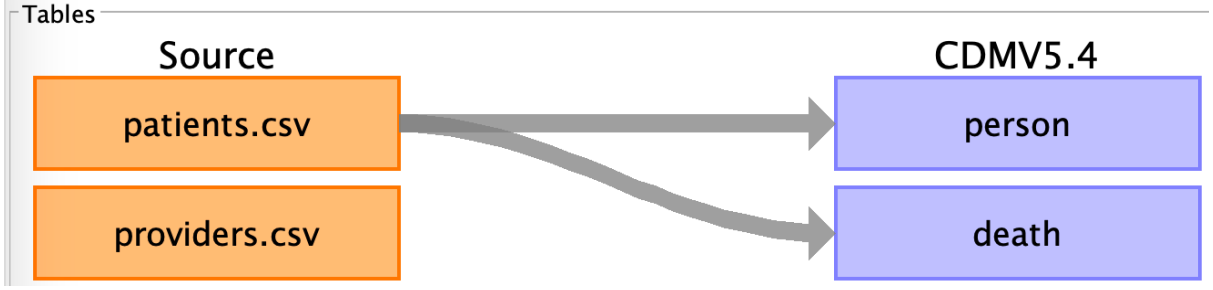


Figure 2 - Simple source table to multiple CDM tables mapping

Another simple mapping is the one in which a single source table is being mapped to multiple CDM tables.

The mapping scenario in Figure 2 is considered simple because the configuration is all visual. There are no additional parameters to be set anywhere to make the patients records go to both the person and the death tables. Another important point to make here is that even though not all patients have died, the mapping must still be made to assure that those who have died are in fact mapped. To assure that records for patients who are still alive aren't mapped to the death table it can be forced that only records with a non-null value in the deathdate are mapped. But that happens on the field level, and not here, on the table level.

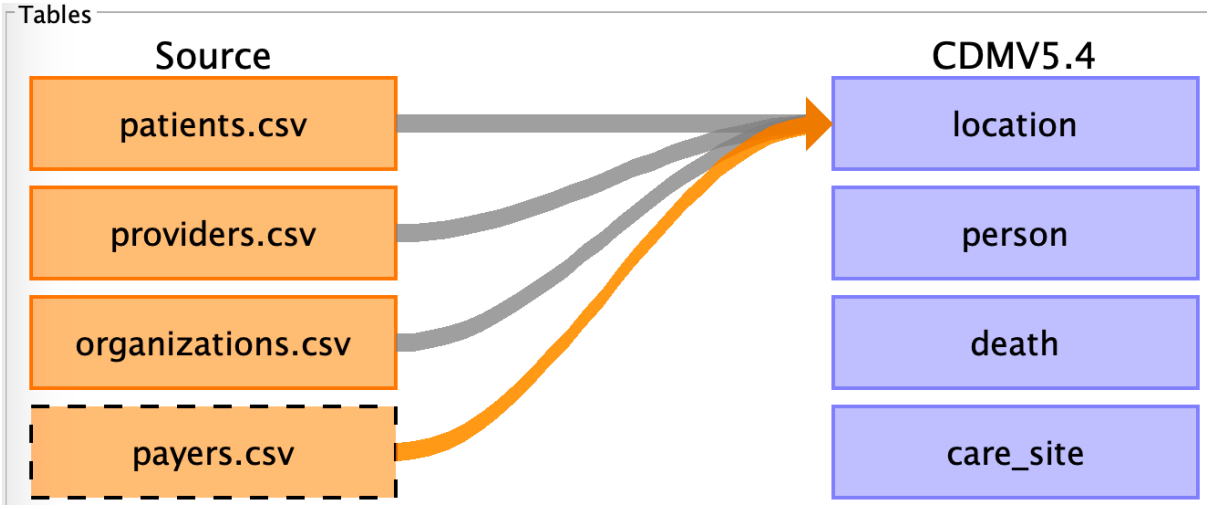


Figure 3 – Multiple source tables to a single CDM table

The opposite scenario, in which multiple source tables are mapped to a single CDM table can also be handled, but this poses more challenges, and it is not as simple as the two scenarios we have just reviewed.

In the scenario presented in Figure 3 multiple source tables contain location information. One could say that the source database is not highly normalized. The OMOP CDM is on the other hand normalized, and so, all of the location information needs to enter the location table. At this stage, we will make our first encounter with the ETL-Engine's mapping language. Without using the mapping language on the location table to tell how the source tables should be treated, the tables will simply be concatenated into the location table. So that every record that can possibly be generated in the location from the four source tables will be generated. However, when you review the data in the data set, you may realize that the location data in the providers.csv table often appears in different records, and at times when a physician works for a specific organization the location data is also repeated in that table.

If you want to avoid having those duplicated records in your database, you can use the *Normalize* rule.

In Figure 4 you will see that the Normalize rule is added to the "Comments" field of the CDM Location table. This is done because there is no available Logic field in this part of the Rabbit in a Hat GUI. So, to allow users to add their comments to the "Comments" field as it is intended, without interfering with the Mapping language parser, the tag "logic:" must be the placed in the beginning of the line when a mapping rule is entered. The Normalize rule builds an index over the fields listed in its input, then whenever a new record is inserted to the Location table, it is matched against the index and if it exists, the record will not be entered into the Location table.

The Normalize rule is not only useful for tables that have multiple sources. As mentioned, even if the providers.csv table was the only table mapping to the Location table, due to the fact that many of the providers have the same address, normalization might be a good idea in this case.

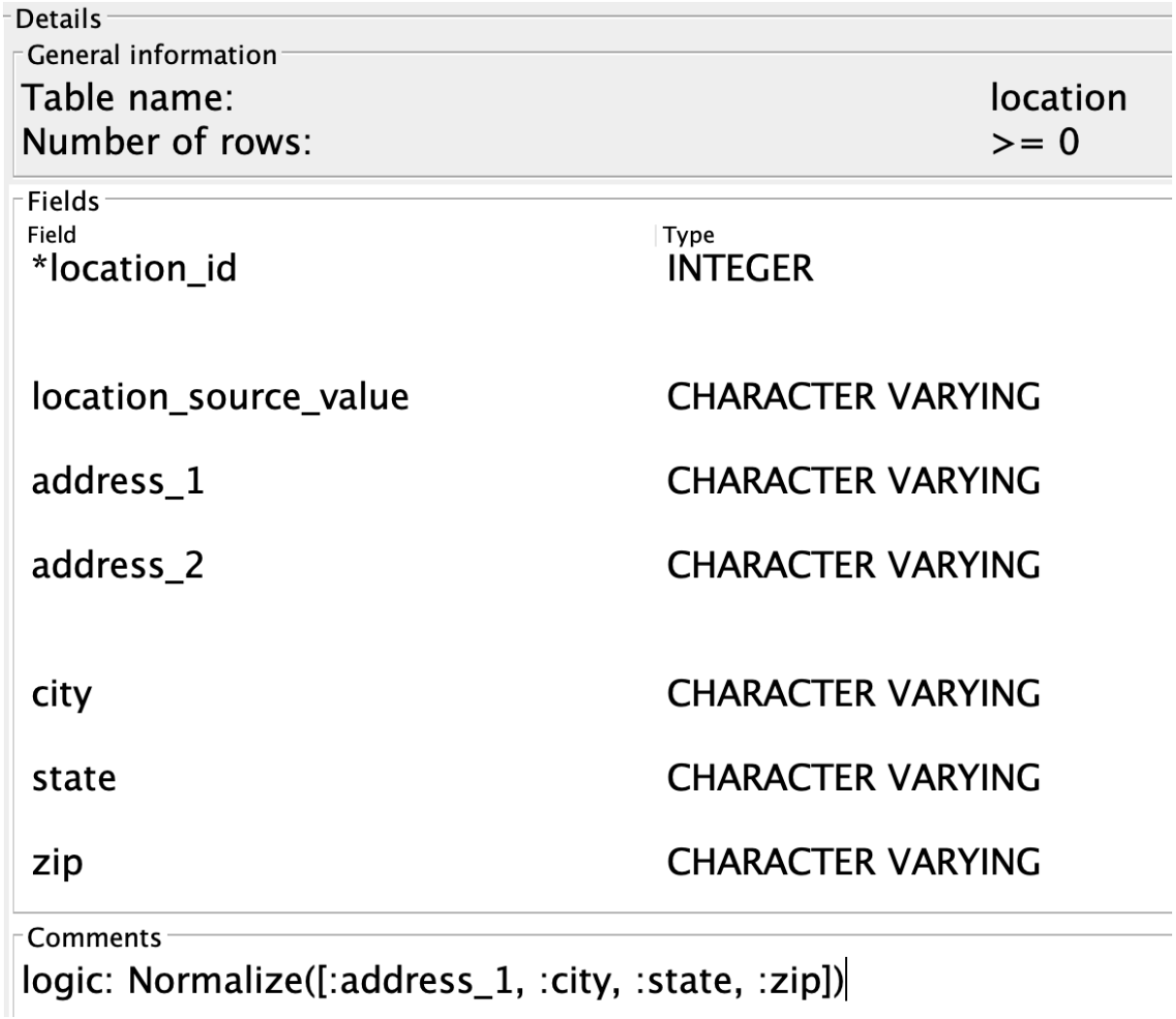


Figure 4 – The details pane for the location table with a Normalize rule added

In a scenario where multiple source tables are being merged into a single CDM table, you may need to combine the individual records from those tables into a single record. Here, the Join and LeftJoin rules can be helpful. The two rules work as expected from an SQL join, except that the syntax is very different. The Join and LeftJoin rules can be nested by storing the joined result to a temporary table and then reference it in a subsequent Join or LeftJoin rule.

When the ETL-Engine is being executed it builds a dependency model from which tables a CDM table gets populated. These may be both source tables and CDM tables. The source table dependencies are defined by the arrows drawn in Rabbit in a Hat. In ETL-Engine lingo these are called basic dependencies. While dependencies for joined tables become complex dependencies as the ETL-Engine must not only load these but generate a completely new table based on the join rules defined on the CDM table. The identification and building of dependencies are done primarily to reduce the memory overhead of the ETL-Engine, and to assure that the data needed to populate a CDM table is available at the time where it is needed.

While the source table dependencies are in fact user defined through the Rabbit in a Hat file, this may not be the case for the CDM table dependencies. Two types of CDM table dependencies exist.

The first type of CDM table dependencies is for the use case explained in-depth in section 4.2.4 Mapping the Person Tablein which the field rule *GetCDMTableId* is used to populate a field in a CDM table from a field in

another CDM table. In this case, *GetCDMTableId* will automatically create a dependency on the CDM table where it is collecting its input.

The second type of CDM table dependencies as about the database used for the OMOP CDM table. Depending on the database foreign keys may be configured between fields in different tables. The ETL-Engine does not know about these inter-table dependencies, so to assure that a table is populated before another table the table rule *DependOn* can be used. This rule simply states that another table must be loaded before the current table. This is demonstrated in **Error! Reference source not found.** that shows the person table depends on the location table. Beware however, that using the *DependOn* rule can be used to create cyclic references, which in the end will result in no data will be transformed and loaded.

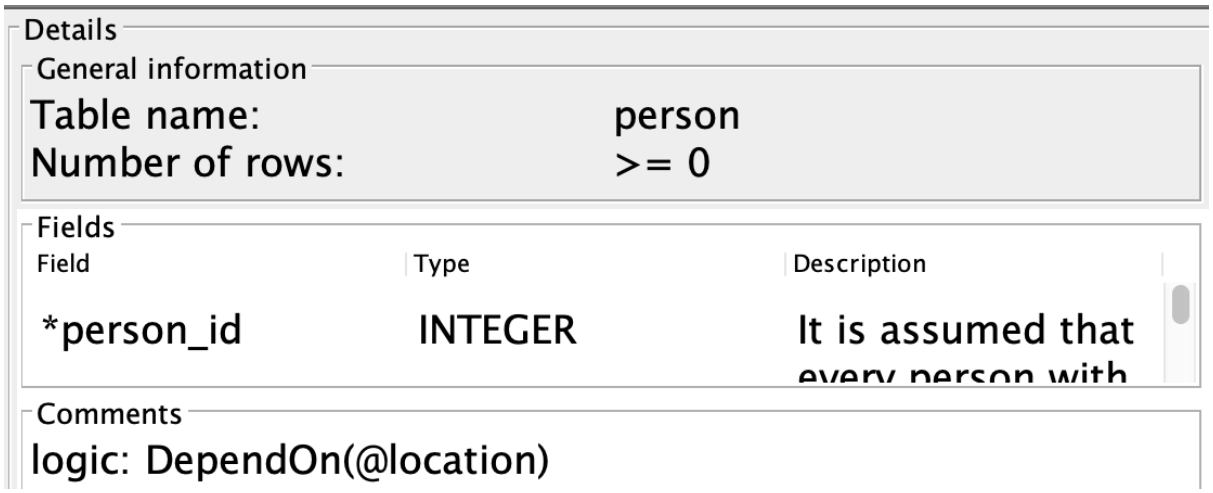


Figure 5 - The Person table depends on the location table

Static records can be added to a table using the *InsertRecord* rule, a common use case might be the CDM_source table in which only a single record is expected describing the source of the data. This is shown in **Error! Reference source not found.Error! Reference source not found.** in which a static record is inserted into the cdm_source CDM table for the Synthmas dataset mapping defined throughout this document.

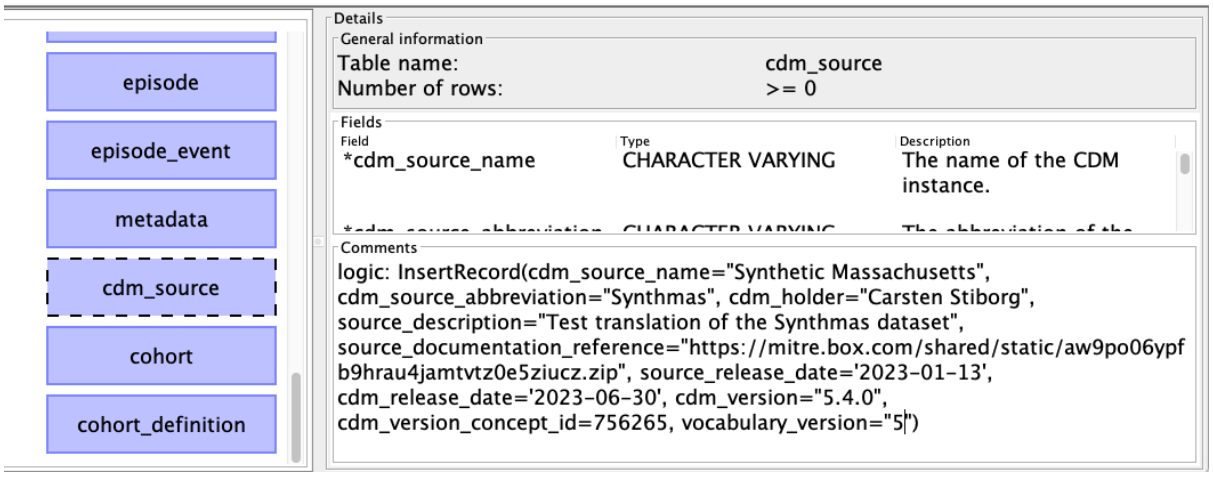


Figure 6 - Inserting a static record in cdm_source

The final type of alteration that can be performed on a table, is to create a virtual field on the table. Virtual fields can be helpful in mapping scenarios where temporary data is needed for normalization, indexing, or field mappings. Virtual fields will only exist in the ETL-Engine and will never be copied to the CDM table. To create a

virtual field on a table the following example adds a visit_occurrence_source_value field to the visit_occurence table, which in CDM version 5.4 is the only table that lacks a source value field for the id.

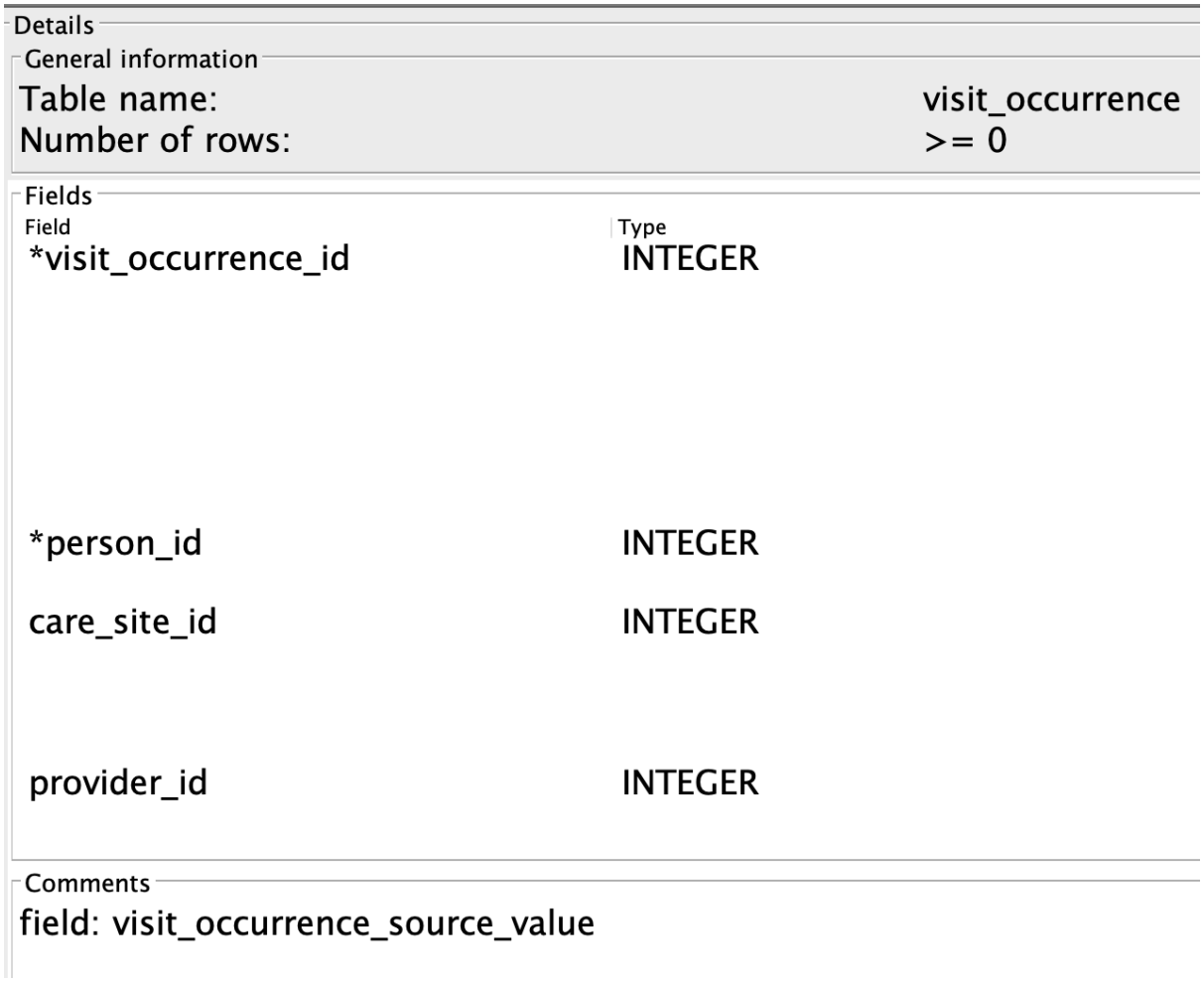


Figure 7 - The details pane for the location table with a visit_occurrence_source_value field added

The example in Figure 7**Error! Reference source not found.** demonstrates that instead of using a "logic:" header in the Comments field for the new field, a "field:" header is used instead. It is the same principle that is being followed, the only difference being that the leading word is "field:". It is also important to notice that "field:" is not being followed by a rule, it is merely being followed by a text string that will become the name of a virtual field which can be accessed from all rules performed on either source to CDM field maps or directly on CDM fields.

The different rules available for table mappings are listed in Appendix A – Table Rules.

4.2 FIELD MAPPING

Like table mappings, the mapping of field values doesn't need any special attention if the source field being mapped from has the same type as the CDM field being mapped to. Quite often, this will not be the case, either because the field's type needs to change, or because the value of the field needs to be altered. In the case of type changes, you can imagine a zip code stored as an integer in a source field while in the CDM it must be presented as a string. Likewise, a common example of a value change is found in the person table, which requires the field birth_year to be an integer, but if you have your birthdate as a date type in your source field then some data transformation needs to happen. The next section will focus on the syntax and semantics of

the mapping language introduced by the ETL-Engine to perform these data transformations. However, in this section we will show examples of how it can be used in a few common scenarios before going all the way down the rabbit hole of explaining how the syntax and semantics of the mapping language works.

4.2.1 TYPES

The first thing to consider when performing field mappings is types and in particular which types are compatible. The synthetic mass database that we are using throughout this document is stored in CSV files, and so, for Rabbit in a Hat, it is the CSV types that will be used on those fields. Our CDM database is stored in a PostgreSQL DB, and therefore, the types are those defined by that database. Had we been using MariaDB, or another database, then the types of that database would be the types we would see on the field. Internally in the ETL-Engine, only four types exist for fields:

- Date (Includes Datetime, Date, Timestamp, Timestamp with zone, etc)
- Float (Includes Real, Float, Double, and all versions of those with specific precision, etc)
- Integer (Includes Integer, Long Integer, Bigint, and all versions of those with specific precision, etc)
- String (Includes Varchar, Character Varying, and all versions of those with specific precision, etc)

However, with the different types of databases available to both Rabbit in a Hat and the ETL-Engine, these types may have different names, and for strings in particular, several types may exist within a single database. While many such types exist in databases to conserve space or provide an increase in speed over other types, the ETL-Engine translates all these types from the source representation into its own representation, which uses the broadest definition of the type, before translating the internal representation into the CDM database representation. When we talk about a specific type in this document, we will use the type names from the ETL-Engine realm, but what is meant is the domain of data that this type covers, no matter whether it is in the source or CDM database.

4.2.2 MAPPING THE LOCATION TABLE

To make the ETL-Engine perform a translation of source data into a CDM database, the minimal requirement for populating a CDM table is that all required fields in that table have a defined mapping. Fields that are not required may be omitted from the mapping and the table mapping will still be performed. In Figure 8 the minimal field mapping configuration for the location table is depicted. Here, the *id* field is expected to contain an integer, as is the *location_id* field. Therefore, no data transformation needs to be performed on the mapping or in the field itself. Obviously, this type of mapping is rather useless when there is no data present. But for the sake of this demonstration, it will suffice.

Figure 8 - Simple functional mapping for the ETL-Engine

As previously stated, if the type of the source field *id* matches the CDM *location_id* type, then running the ETL-Engine on a Rabbit in a Hat file with this content will get the *location* table populated with the *location_id*. Unfortunately, as we will see from Figure 9, the source *id* type does not have an integer type, but instead a string type holding a UUID. And therefore, this mapping would not work. If it is not possible to reuse the IDs from the source database, which will often be the case, either because of type differences as in this case, or for other reasons e.g., when data enters a CDM table from multiple source tables with identical IDs. Here, the best practice solution is to use the *<table name>_source_value* field in the CDM table to store the ID in a unique way. This means that if you have multiple tables with identical IDs you will have to pre- or postfix these with information about the source of the ID before storing this value, however, in this particular case where we have a UUID, we can simply trust that this ID is globally unique and we can directly map that into the *location_source_value* field as it is a string type field. The next question is then: What do we do with the *location_id* field? – As we always need a unique id for a CDM table, a command in the mapping language has been created for exactly this purpose. This command is named *AutoGenId*.

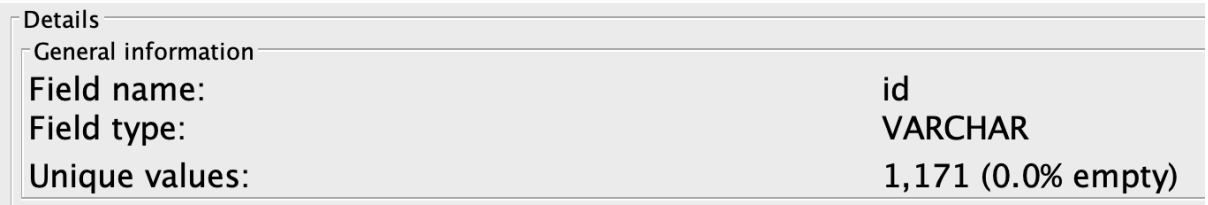


Figure 9 - The Source id field's type

In Figure 10 it is seen that an *AutoGenId* command has been added to the comments section of the *location_id* field. As for tables there is no logic section for field. So, to add logic we must once again add the "logic:" tag before the command.

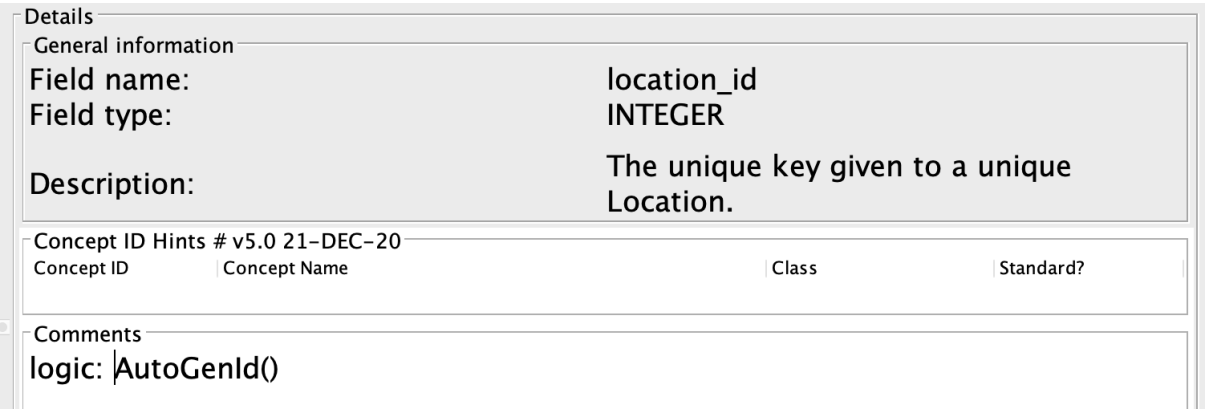


Figure 10 - Using the AutoGenId command

The Rabbit in a Hat visual mapping table is shown in Figure 11. At this stage the *location* table could actually be transformed using the ETL-Engine. It will generate a location table and fill out the two fields: *location_id* and *location_source_value*. The *location_id* will contain a sequence starting from 0, while *location_source_value* will contain the original id from the source.

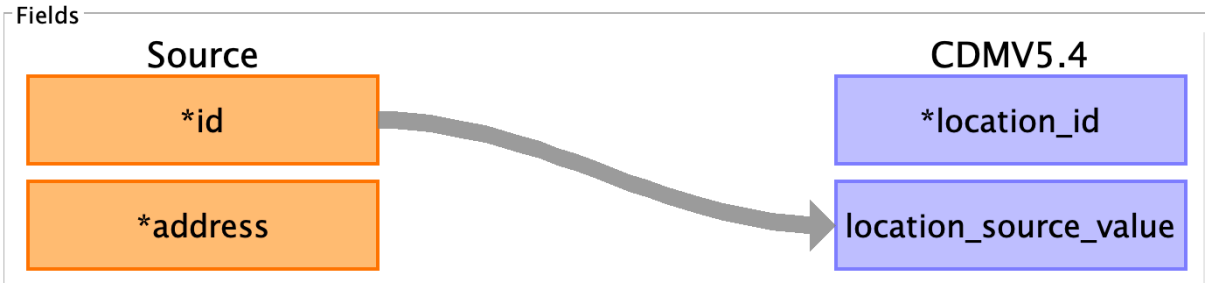


Figure 11 - Using the location_source_value for the source id

At this stage, the remaining mappings to the location table can be mapped using field-to-field mappings, as shown in Figure 12. This will work out of the box for the fields that share the same type in the source and CDM tables. One mapping does not share the type between the source and CDM tables, which is the zip-to-zip mapping. Here, the source zip field is of integer type, while the CDM zip field is of type string. Another mapping which will need attention is the state-to-state mapping. Here, when reviewing Rabbit in a Hat, everything looks correct, but when mapping the field, it will be discovered that the state field in the location CDM table is a character field of size 2. Thus, as the data in the source table contains the full state name which in our case is “Massachusetts” for all records, we need to map this state data to its two-character representation. But let’s start by handling the zip field mapping.

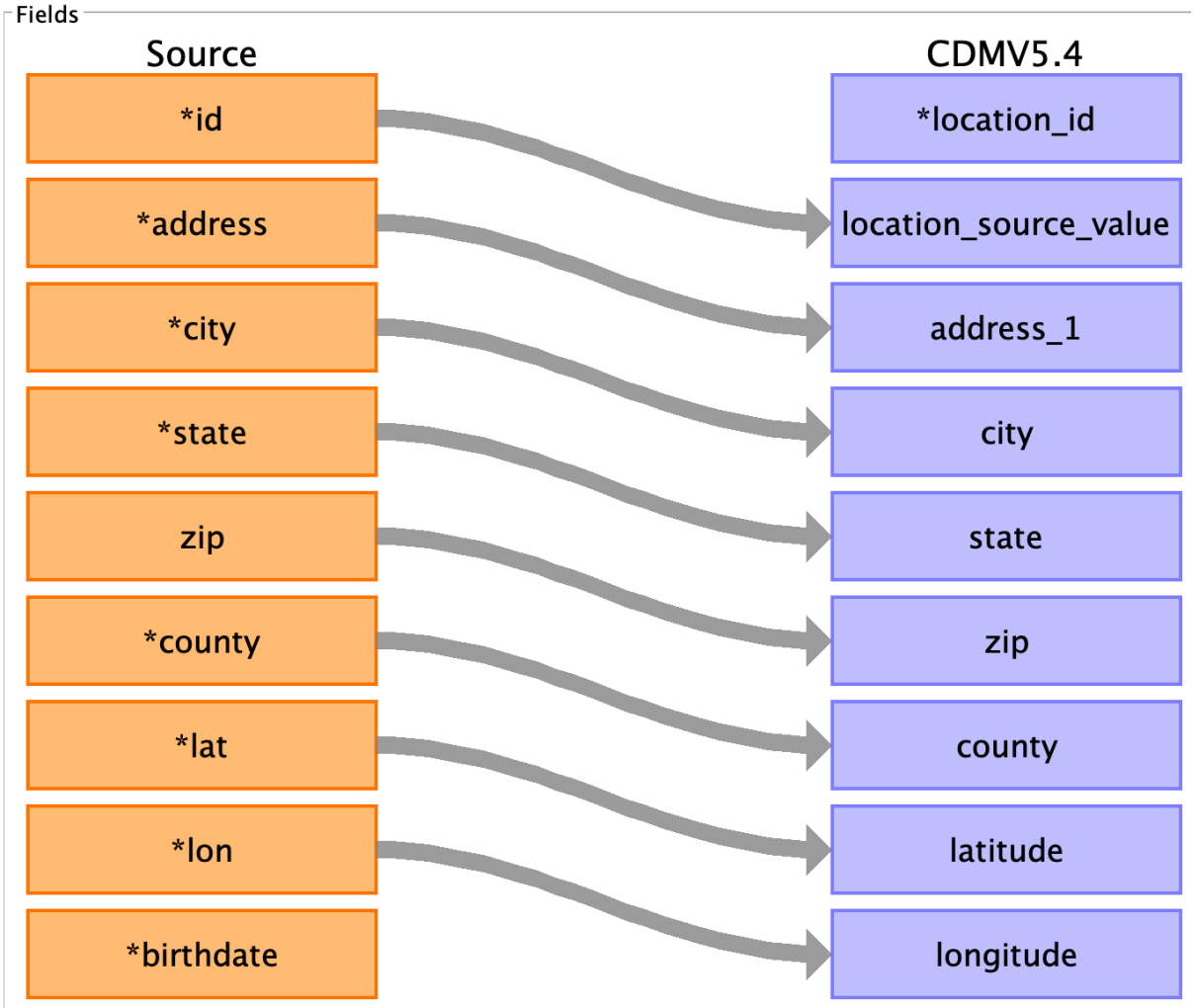


Figure 12 - All fields are mapped in the location table

The zip field in the CDM table also has a maximum length of 9 characters. However, we know that US zip code (particularly when represented as integers – without hyphens) are no longer than 5 characters but may also be less than 5. To assure that every zip code is five characters long we may have to pre-pad it with zeros.

The mapping logic for the zip-to-zip mapping then comes to look as is seen in Figure 13. First, the integer is cast to a string, then, if it less than 5 chars, it is padded with leading zeroes to increase the length to 5.

Note that on the mapping there is a logic block, and with the logic block it is not necessary to use any form for prefixing for the mapping rules. If we want to make sure that the zipcode string never exceeds five characters, then we could add a Truncate rule to the end of the logic block. We can also decide that we want that logic block on the field itself. In terms of the Synthmas data set, in which several of the source tables contain location information but often in different formats.

Figure 13 - The zip to zip mapping rules

For example, the *organizations* table contains the *zip* as a string because many of the organizations uses US zip codes with routing information, i.e. the 5 digit zip code followed by a hyphen and four digits. As the *location* table's zip field only can carry 9 characters, it cannot hold these type of zip codes, therefore by adding a truncate function on the field itself rather than on the map, we can extend it to more of a global scope, or at least a scope that covers all input to that particular field no matter where it comes from. This also means that everything which enters the *location* table *zip* field must be a string of at least 5 characters. This also requires that all maps must assure to deliver this output to the field.

Figure 14 - Zip field rule

With this in mind, we add a Truncate rule to the *zip* field making it look as in Figure 14.

Returning to the state map we have two options to map that data. Either we choose not to map it and hardcode the state, or we make a mapping replacing "Massachusetts" with "MA". As mentioned earlier, multiple source tables deliver input to the location table, while most of them have the state using a two-character code, some of them do not have values which match "Massachusetts". Thus, the proper approach would be to perform a mapping on the map between the source *state* field and the CDM *state* field, as is seen in Figure 15.

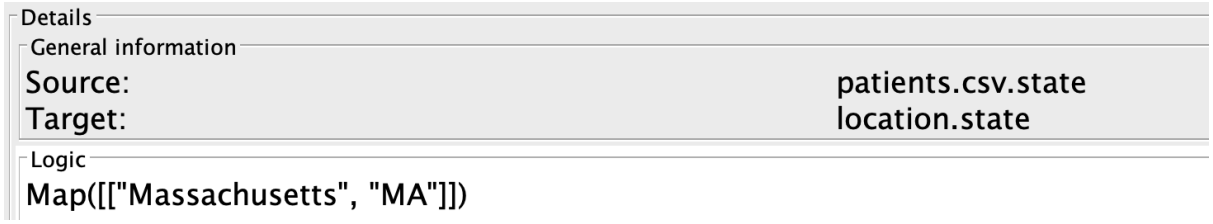


Figure 15 - State to state map rule

Figure 16 - Missing mapped fields

At this stage we have most fields mapped in the *location* table, only three fields remain unmapped, as shown in Figure 16. As, none of the fields are required it is perfectly fine to leave them empty. However, as we know that all addresses are going to be within the US, we will hardcode the *country_concept_id*.

When hardcoding a field to have a generated value, like a static value or a sequence, the rule must be added to the target field. There simply won't be any field-to-field map to that field and the rule itself will fail if it is given any input. Figure 17 shows how we set the *country_concept_id* to the *concept_id* for the US. This ID has been found by following a link to Athena which is added to Rabbit in a Hat's description field inside the details block.

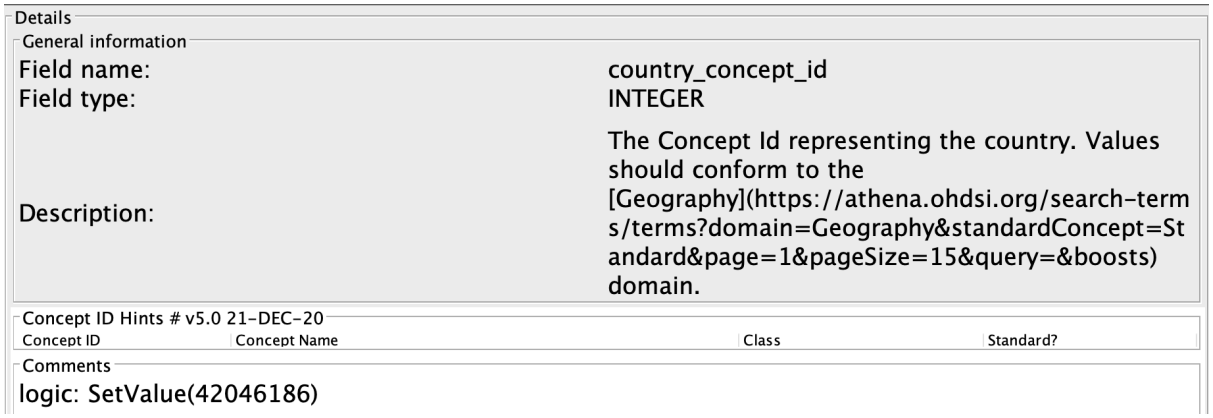


Figure 17 - The rule for the country_concept_id field

Finally, we have built a functioning transformation of the Synthmas data set's patients.csv table to the OMOP CDM location table. The Synthmas data set has a lot of tables that must be mapped and while we will only look at a few in this document to give an overview of the functionality, you should already now be able to map simple tables. However, before continuing with the Synthmas data set it is now time that we look a bit into the different types of rules.

4.2.3 MAPPING RULE TYPES

Looking back at the mapping of the fields from the *patients.csv* table to the *location* table performed in section 4.2.2 Mapping the Location Table, we have already seen a few rules. These are: *AsString*, *AutoGenId*, *Map*,

PrePad, *SetValue*, and *Truncate*. While mapping rules have been used as a broad term for any rule used in the ETL-Engine's mapping language, there are in fact two different types of rules which, whilst alike in terms of syntax, are quite different in terms of semantics.

If we look at the four rules *AsString*, *Map*, *PrePad*, and *Truncate*, they are truly mapping rules. They take one input, commonly from a source field or as part of a chain of rules, transform it, and produce an output which will usually be given to a CDM field or another rule in a chain of rules. The majority of rules are mapping rules.

The other type of rule is the aggregating rule. Aggregating rules are any rule that does not fit the first description. They take their name from the fact that they commonly take any amount of input and generate a single output. The exceptions are two aggregating rules which are a bit peculiar as they are the only aggregating rules which do not accept any input at all.

While mapping rules are used to focus on transforming a single data value, aggregating rules take multiple values into consideration to produce or choose a single value.

A table of common rules are available below – their full documentation and the full list of rules are available in Appendix B – Mapping Rules and Appendix C – Aggregating Rules.

Mapping Rules				Aggregating Rules		
Map	**YearFromDate**	**GetCDMTableId**	**Truncate**	**AutoGenId**	**MergeDate**	**Min**
UsagiMap	**MonthFromDate**	**SaveToVirtualField**	**PostPad**	**SetValue**	**Add**	**Max**
VocabMap	**DayFromDate**	**AssertNot**	**PrePad**	**Selector**	**Subtract**	**Use**

4.2.4 MAPPING THE PERSON TABLE

With this short interlude on the different types of rules that can be used in field-to-field maps and on fields themselves, we will proceed to look at how to map the person table, which is the most important table in the OMOP CDM.

We will not pay as close attention to all details, but we will present a way to map data to the person table. Once again, we are using the Synthmas data set and a mapping from its patients.csv table to the CDM person table. That field mapping is seen in

Figure **18**.

More fields exist in the person table than are depicted in

Figure **18**. The fields not shown are *provider_id*, *care_site_id*, *gender_source_concept_id*, *race_source_concept_id*, and *ethnicity_source_concept_id*.

None of the source_concept_ids are mapped because the values in the Synthmas data set are strictly strings. M and F for *gender*. White, black, asian, and native for *race*, and nonhispanic and hispanic for *ethnicity*.

For these values simple *Map* rules are created. However, unlike that Map rule which is shown for the location table's state field in Figure 15 on page 20 a default value is added to the Map rule and an extra rule is added to the rule chain, namely, the *AssertNot* rule. That way we can assure that values which do not have a mapping in the Map rule get a default value, and that records having that default value will not be inserted into the CDM person table, but instead they will be logged. Reviewing the log will then allow us to find those values that do not yet have a mapping and that will allow us to improve the ETL following an iterative process.

The three different *Map* rules are shown in Figure 19, Figure 20, and Figure 21. Another thing to note is that for each of the three source fields there is a field-to-field mapping to a source_value, these mappings do not have any rules, they are merely there to keep the original data in the CDM person table.

Having looked at the three source fields which must be mapped to concept IDs, only two fields of the *patients.csv* source table need to be investigated. The first is the *id* field and the second is the *birthdate* field. We will start with the birthdate field. The source *birthdate* field is mapped to four fields in the CDM *person* table, the *year_of_birth* field, which is mandatory for the transformation to function, the *month_of_birth*, the *day_of_birth*, and the *birth_datetime*. The birth_datetime is a simple field-to-field mapping without any rules as both fields, in ETL-Engine terms are date types. The rest of the aforementioned CDM fields are of integer type and require the individual year, month and day components of the date. Special mapping rules exist that can do this, they are: *YearFromDate*, *MonthFromDate*, and *DayFromDate*, respectively.

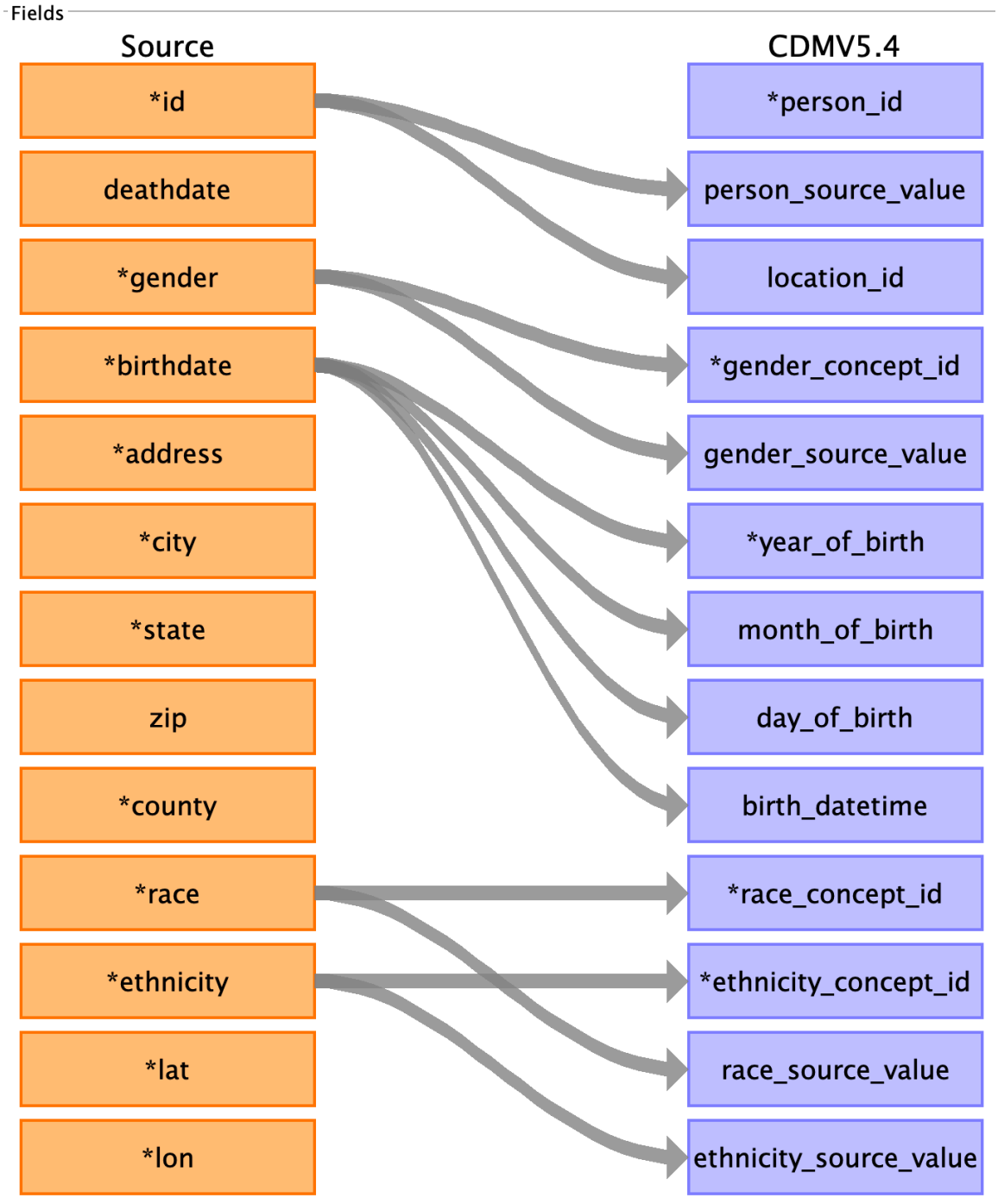


Figure 18 - The field mappings in the person table

Figure **22** shows the year from date mapping for the required *year_of_birth* field in the CDM person table. The two remaining field-to-field mappings look the same, albeit using the mapping rules for the specific part of the date.

The final source field to investigate is the *id* field. As for the *location* table's *location_id*, the same pattern is used for the *person_id*, ie. the original UUID from the source *id* field is stored without modification to the *person_source_value* field, while the *person_id* field has an *AutoGenId* rule added to create a new sequence of IDs specifically for the CDM person table. There is nothing new to this. However, the *location_id* field is also being set from the source *id* field. This works because the location records transformed from the patients.csv table have the same ID stored in the *location_source_value* field. This can be seen in Figure 12.

Figure 19 - Person table gender mapping

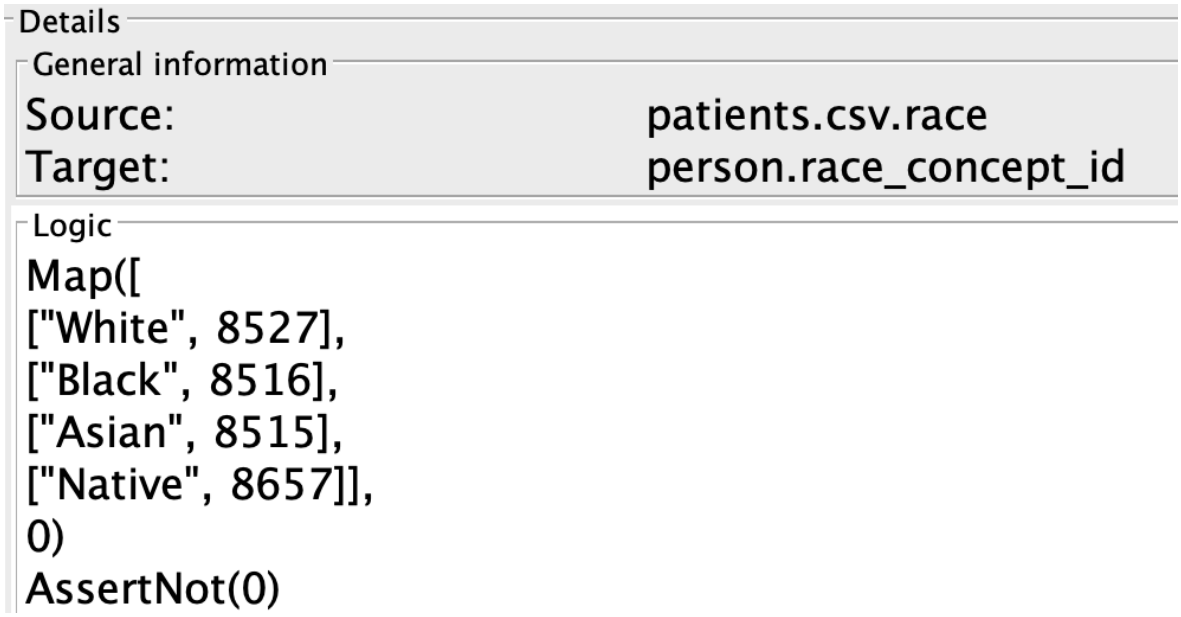


Figure 20 - Person table race mapping

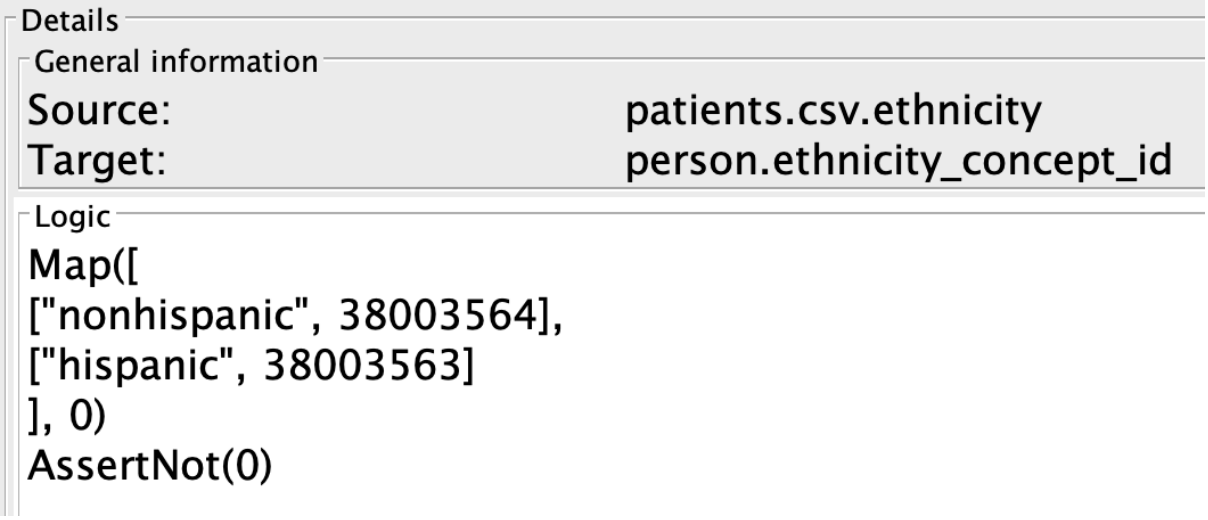


Figure 21 - Person table ethnicity mapping

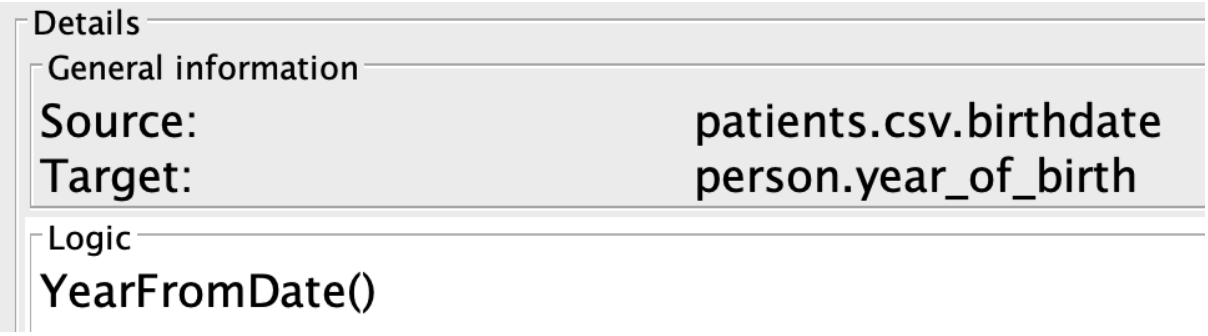


Figure 22 - Year from date mapping

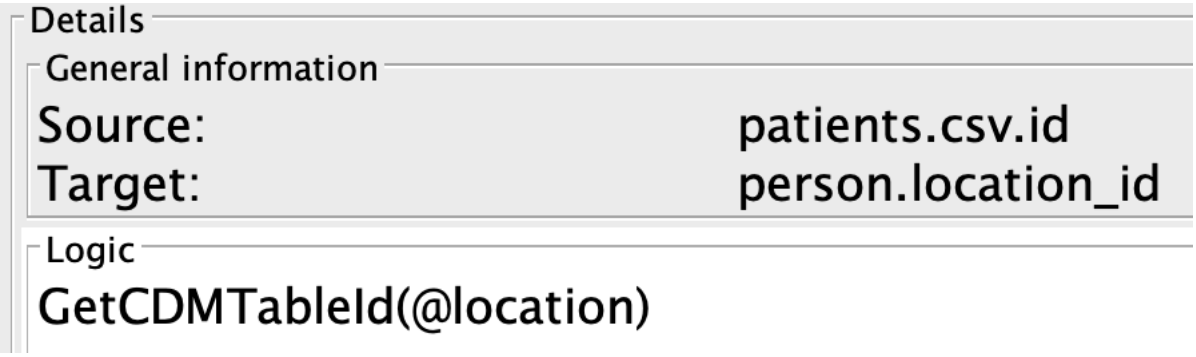


Figure 23 - Transforming the patients.csv id field into the location_id

Therefore, to collect the proper *location_id* from the *location* table, a lookup using the *id* from the patients.csv table on the *location_source_value* field to get the *location_id* must be used. Luckily this is exactly what the *GetCDMTableId* rule does. It has one mandatory input, which is the name of the table to perform the lookup on. It does accept additional inputs for the fields to use, but the defaults are the *<table>_source_value* field for the key that the rule's input must match and the *<table>_id* field for what the rule's input must be transformed into. The mapping rule is shown in Figure 23.

This concludes the walk-through of the mapping of the Synthmas data set's patients.csv table into the OMOP CDM person table. Obviously, every data set is different, some already have data stored using a standardized vocabulary, some highly normalize their databases requiring many joins to be performed to get the required data for a the CDM tables. The scope of the source databases is obviously open-ended, the ETL-Engine

attempts to cater to all these different scenarios by simplifying and generalizing its approach to performing the transformation. Before we end this section on field mappings, we will have a look at some of the more useful topics that the ETL-Engine handles.

4.2.5 USEFUL TOPICS

In this section we will take a look at some useful scenarios handled by the ETL-Engine, which were not covered in the table mapping walk-throughs presented earlier. We will still be using the Synthmas data set mapping into the OMOP CDM for this.

The three scenarios covered in this section revolve around mappings, the first is using the OMOP CDM Vocabulary for its mapping, the second is using Usagi for the mapping, and the final third mapping will take a look at conditionally chained mappings.

When using the OMOP CDM, a fantastic feature is the standard OMOP vocabulary which contains a lot of different vocabulary sets. Nobody wants to do Usagi maps or static maps using the mapping language in Rabbit in a Hat. The OMOP Vocabulary is here – let's use it!

In the source table *devices.csv* there's a code field for the type of device a patient is wearing. Quite commonly a defibrillator. The code for this is stored in the SNOMED vocabulary, and many of the device domain vocabulary values are also mapped to SNOMED values, but not all. But that is really not important as we won't store that SNOMED code, but instead use the OMOP CDM Vocabulary concept id in the CDM field. We will also store the concept ID of the original SNOMED code in the *device_source_concept_id* field. This mapping can be seen in Figure 24. In Figure 25 it is shown how the mapping is performed, the SNOMED code is stored as an integer so first it is cast to a string, next it is treated as a SNOMED code, for which we will get the concept_id. Likewise for the *device_source_concept_id*, we simply ask for the concept_id for that SNOMED code. This is shown in Figure 26 - Getting the concept_id for a SNOMED code.

Unfortunately, we aren't so lucky as to always get the SNOMED code handed on a silver platter. In other cases, we simply get a string and then we must use Usagi. Detailed documentation on the use of Usagi is available in the book of OHDSI (2).

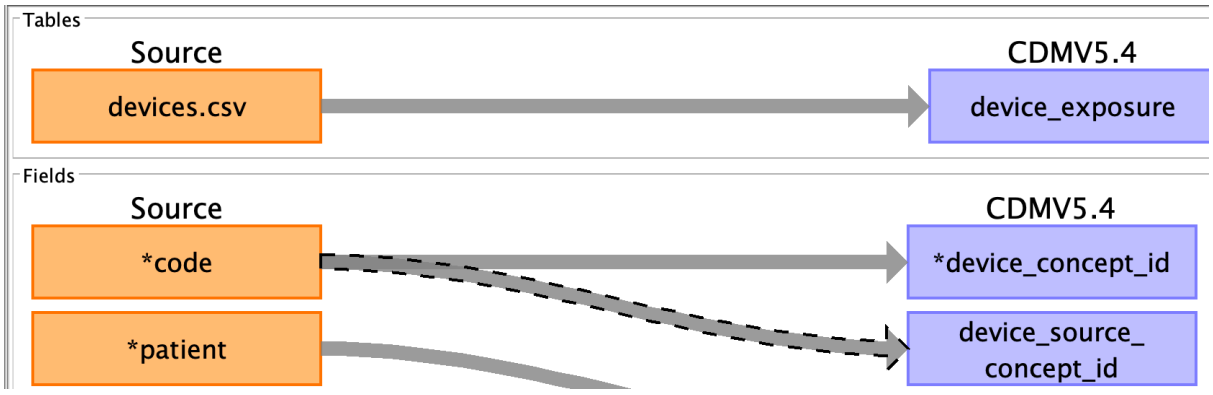


Figure 24 - Device table mapping

Figure 25 - Getting the standard concept_id for a SNOMED code

Figure 26 - Getting the concept_id for a SNOMED code

Figure 27 - The units mapping

When the Usagi map has been created it can be put in the folder defined for Usagi maps in the ETL-Engine's configuration file (see section 3). In the source table *observations.csv,* the units field contains unit descriptions; however, we would like to get them mapped to concept_ids for units. Figure 27 shows the mapping of the *units* source field to the *unit_concept_id* CDM field. While Figure 28 shows some of the text strings that the Usagi file must map.

Figure 28 - Values to map to concept_ids

When the Usagi file has been created and put into the defined directory, the rules as shown in Figure 29 can be used to call it. The name of the Usagi file is "units-exported.csv".

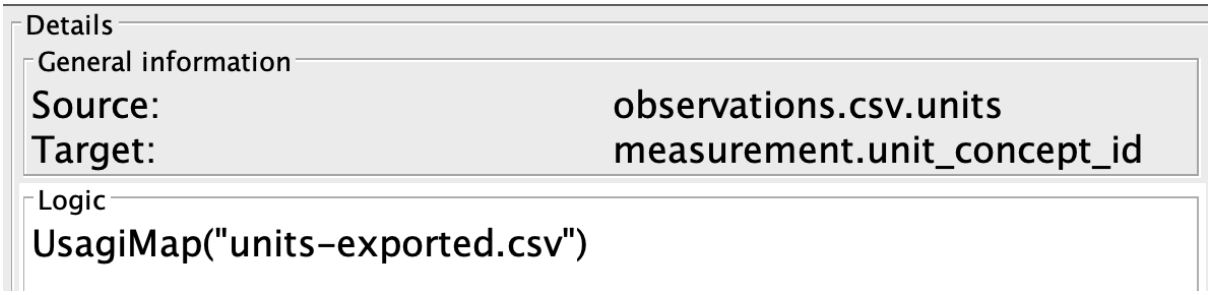


Figure 29 - Using the UsagiMap

Using the vocabulary is certainly the best way to go as the mapping comes for free. However, sometimes codes may be missing in the vocabulary, and it is necessary to get the proper codes mapped. Several approaches for this exist, one is to have a Map prior to the VocabMap assuring that missing codes are replaced with existing codes. This approach is visualized in Figure 30.

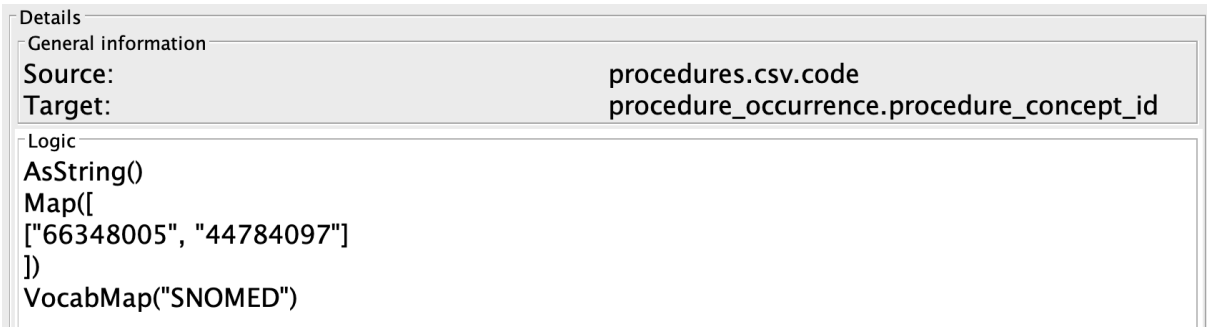


Figure 30 - Using a pre-map to remedy missing vocabulary codes

However, as VocabMap expects codes using a specific vocabulary you may find that if codes are missing in the vocabulary there is no easy way to pre-map these codes to an existing code within the vocabulary. So, even if you know which concept id you need to map to, using the pre-mapped approach will not work as you need an existing code in the chosen vocabulary. An elaborate approach to circumvent this could be to execute both a Map and a VocabMap and store each of the resulting values in their own variable. Both Map and VocabMap must use default values for missing values. Finally, a Selector is used to return the Map result if it has a non-default value or VocabMap in any other case. This approach is depicted in Figure 31.

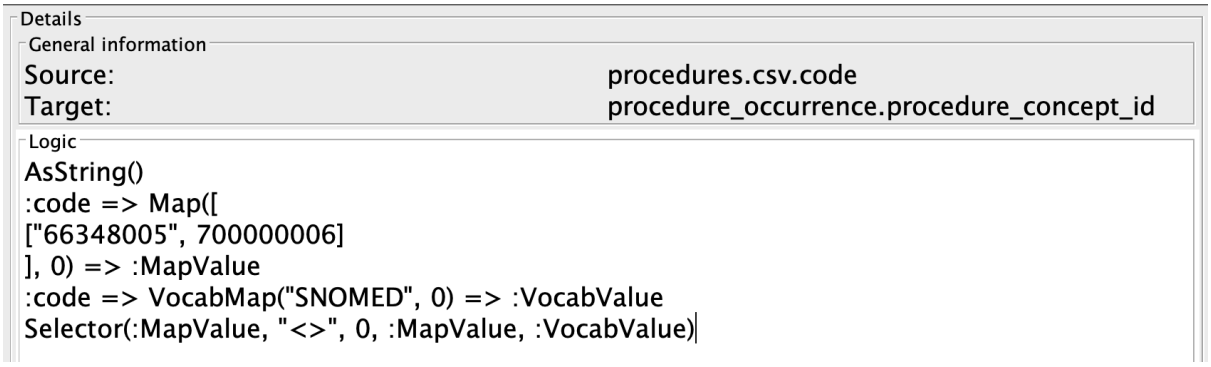


Figure 31 - Using both a Map and a VocabMap to find a proper code

Another approach which is both more readable and has faster execution time (advantageous on larger data sets) and essentially results in the same output, is using the conditional chaining feature of the mapping language. Here, we are getting back to something that looks pretty close to the pre-map scenario, but

between the two rules we are using the sign for conditional chaining: ||. This means that VocabMap is only executed if the Map fails. This approach is depicted in Figure 32.

The conditional chaining approach can also be used with UsagiMaps, in any order, and with multiple steps.

Figure 32 – Using conditional chaining

With this, the walk-through of how to perform field mapping with the ETL-Engine comes to an end. In this section we have been looking at how the ETL-Engine uses Rabbit in a Hat, but also adds its own mapping language to facilitate the mapping of values. We have looked at how tables from the Synthmas data set can have their fields mapped using the ETL-Engine, discussed types within the ETL-Engine, looked at the two different types of mapping rules, and finally we have reviewed a few of the most useful rules. In the next section we will review the syntax and discuss the semantics of the mapping language.

4.3 THE MAPPING LANGUAGE SYNTAX

The ETL-Engine employs its own mapping language for specific purposes that cannot be expressed in the Rabbit in a Hat mapping model. In this section, the mapping language's syntax and semantics will be explained along with its integration to Rabbit in a Hat.

We will start by reviewing the syntax of the mapping language. Up until this point, the majority of rules we have looked at have been single line rules. The mapping language does however support some more advanced use cases and thus it is a bit more complex than what has been shown up until now. The table below shows the syntax tree of the mapping language. In the left-hand column we have the name of the current leaf in the syntax tree, In the right column is the expansion, or what that leaf name represents, written as a mix of leaf names and regular expressions. The syntax tree should be read from the top down, starting with the most general rule on top (namely the RuleChain which is how every mapping language statement can be considered.), down to the most specific leaf, such as *Integer* which describes how an integer is represented in the language.

Syntax tree leaf name	Expansion
RuleChain	Rule_Definition+
Rule_Definition	[Input_Field_Filter =>]? Rule [|| Conditional_Rule]* [=> Output_Fields_List]?
Input_Field_Filter	Field_List
Conditional_Rule	Rule
Rule	Rule_Name\(Rule_Parameters\)
Output_Fields_List	Field_List
Field_list	Field [, Field_list]*
Rule_Name	[A-Z][A-Za-z]+
Rule_Parameters	Parameter [, Rule_Parameters]*
Parameter	Keywork_Parameter | Table_Field | Table | Field | String | Date | Integer | Float | Regex | NULL
Keywork_Parameter	[a-z][a-z0-9._]= (Table_Field | Table | Field | String | Date | Integer | Float | Regex | NULL)

Table_Field	@[A-Za-z][A-Za-z0-9._]+ :[A-Za-z][A-Za-z0-9._]+
Table	@[A-Za-z][A-Za-z0-9._]+
Field	:[A-Za-z][A-Za-z0-9._]+
String	".*"
Date	'.*'
Integer	[0-9]+
Float	[0-9]+\.[0-9]+
Regex	/.*/

In the following table an explanation of the semantics for a number of these will be provided in general terms and finally a number of examples with explanations will be given.

Syntax tree leaf name	Semantic explanation
RuleChain	At the beginning of the rule chain execution, the environment for execution is set up. In a field-to-field map, the environment will always consist of one input field and one output field. The input field will have the *Table_Field* name of the source field in the map. The output field will not be named, and doesn't need to have a specific name, however, one field exactly must exist when the RuleChain has been executed. If multiple values exist, then an error will occur, and nothing will be mapped for that table. If the RuleChain is being run from a field, then the environment will contain all the fields that are mapped into it, which means that no fields are also a valid case. However, at the end of the execution, the RuleChain must only contain a single field and if this requirement is not met an error will occur. All rules in a RuleChain are executed top down, this means that any changes made to a field, or the environment will affect any subsequent rules.
Rule_Definition	Each rule definition comprises a set of inputs, explicitly or implicitly to a rule, a rule, or a conditional rule chain, to execute, and finally an explicit or implicit list of outputs.
Input_Field_Filter	An input field filter can be added in front of any rule, which assures that only the fields specifically named in the filter are passed to that rule. All fields referred in an input field filter must exist at the time of execution. Meaning that fields must either be defined from source fields or via an output fields list which was executed as part of a preceding rule. When an input field filter is present, but no output_fields_list is defined, then the behavior depends on whether the following rule is a mapping or an aggregation rule. - Mapping rules will update the fields passed into them through the input field filter. - Aggregation rules will delete the fields passed into them and create a new output field.
Conditional_Rule	Some mapping rules can be joined as a conditional chain of rules. Meaning that if the first mapping rule didn't find an appropriate mapping, the second rule will be executed, and so on and so forth. Not all mapping rules support being conditionally chained.
Rule	Rules have a lot of variation in their semantics depending on whether they are mapping rules or aggregating rules. • Mapping rules will always execute in a one-to-one basis in terms of the available input fields. This means that where a rule is applied to a field (not a field-to-field map), the mapping rule will be performed on all of the available input fields, unless an input field filter is defined. If a list of output fields is defined it must have exactly the same length as the length of the inputs available to the rule. • Aggregating rules can only ever have a single output. However, the number of inputs is unlimited. Thus, an input field filter will work just as

	for the mapping rule, while the output fields list only can name a single field.
Output_Fields_List	An output fields list holds all of the fields where output from a rule will be stored. The semantics depend on the type of rule. - Mapping rules will simply provide the output to fields in the order of the input fields. Thus, a mapping rule can change the content of input fields. Also, new fields can be declared in the output fields list. If a new field is declared the output will simply be added to this field. - Aggregating rules can only have a single output field. The aggregating rule will remove all input fields, except if the output fields list references a new field. In that case a new field will then be created for the output of the aggregating rule and the input will be kept. If the output fields list references an existing input which is also part of the input given to the aggregating rule, that field will not be deleted but the rest of the input fields will.
Field_list	The field list does not have any special semantics related to it.
Rule_Name	The available rule names, their semantics, and valid parameters and types are described in Appendix B – Mapping Rules and Appendix C – Aggregating Rules.
Rule_Parameters	
Parameter	
Table_Field	Table fields reference the absolute path to a source field by also mentioning the table it belongs to. This can be necessary when using joined tables. Table field, table, and field types are automatically convertible within the ETL-Engine, thus if a field is mentioned as an input a table field will also be valid.
Table	Tables are used to reference either a source or a CDM table, this may depend on context.
Field	Fields are used to reference either a source or a CDM field, this may depend on context.
String	Strings are any characters surrounded by "'s
Date	Anything surrounded by ''s will be parsed as a date. Thus, various date formats are valid. However, a guideline is to use 'DD/MM-YYYY HH:MI:SS'. The time can be ignored in which case it will become 00:00:00. If the date part is left out it will become 01/01-0000 – but as dates covers many different types of date types in the endpoint databases this might not pose a problem.
Integer	Integers are any sequence of digits
Float	Floats are any sequence of digits which contains a . – A single dot must exist in a float.
Regex	Certain parameters can have a regex instead of a string, the regex language is not defined in the mapping language, it simply forwards anything with /'s around it to a regular expression engine.

Thus, having defined the syntax and semantics of the mapping language a few examples with explanations will be presented in the following table.

Mapping language example	Explanation
Map([["M", 8507], ["F", 8532]])	Here a mapping rule is used on a field-to-field map in which the input is expected to be a string being either "M" or "F". The input value from the source field will be replaced if its value can be mapped.
Map([["M", 8507]]) || Map([["F", 8532]])	These conditionally chained rules lead to exactly the same result as above. The first rule will only map the character "M" to the code 8507, if "F" is seen it will fail to perform a mapping and the second Map will be executed, leading to a map of "F" to the value 8532 being performed.
:gender => Map([["M", 8507], ["F", 8532]	This example shows a mapping rule is used on a specific field in a field-to-field map. (No other field should be present – thus being explicit about the source field name)

]) => :result **:result => AssertNot(8532)** **Use(:gender)**	The mapping rule result is stored into a new virtual field named *result*. The virtual field *result* can be passed to the field rule chain, but at the end of that rule chain only one field can persist. In this case however, *result* is given to the mapping rule *AssertNot*, which means that the *gender* field is not having *AssertNot* executed for it. In the end the aggregating rule *Use* is setting the *gender* field as the output field for the rule chain, thus discarding the *result* field. In essence what is being done here could be achieved by simply having an AssertNot("F"). The ETL-Engine does not do anything to simplify statement like this. Input field filters and output fields lists are expensive in terms of time and should only be used when there's no other option.
logic: Selector(:int1, "=", :int2, "Equal", "Unequal")	Here an aggregation rule is used directly on a CDM field. Two source fields are expected to be mapped to this field, *int1* and *int2*. In case these two fields have the same value the string "Equal" will be stored in a field called *Selector*. (The name of the aggregating rule is the default name of a returning field when no output field is named) All input fields, no matter whether they are used by the *Selector* rule or not will be deleted. In case the two input fields int1 and int2 differ, the string "Unequal" will be returned.
logic: Selector(:date1, "<", :date2, :date1, :date2) => :Result **logic: Use(:Result)**	In this aggregation rule all inputs on a CDM field are being passed to the *Selector* rule. The two fields *date1* and *date2* are being compared for ordinality, after which the lowest one of them are being stored in a virtual field called *Result*. At this stage at least three fields will exist: *date1*, *date2*, and *Result*. The next rule in the rule chain is the *Use* aggregation rule, it will consume all available fields and only keep the *Result* field, which will in fact be stored in a virtual field called *Use*, whilst the three fields *date1*, *date2*, and *Result* are all deleted.
logic: Min(:int1, :int4)	In this example a simple aggregation rule on a field is taking all input and using the two fields *int1* and *int4*, from among which the lowest is selected and returned in the virtual field *Min* while all other fields are deleted.
logic: :int1, int4 => Min(:int1, :int4) => :min **logic: :int2, int3 => Max(:int2, :int3) => :max** **logic: Selector(:min, "<", :max, :min, :max)**	In an extension of the example above the aggregation rule on the field is now only taking the two specific fields *int1* and *int4* to find the minimum of those two and store it in the virtual field *min*. This will result in the following fields: *int1*, *int2*, *int3*, *int4*, and *min*. The next rule in the rule chain will find the maximal value between *int2* and *int3*. As the fields are specified in both the *Min* rule and the *Max* rule, we are actually being a bit redundant here. We could choose to write either: :int1, :int4 => Min() =>:min – or Min(:int1, :int4) => :min. That would yield exactly the same result, and the same for the Max rule. Obviously, this is functionality that lies with the *Min* and *Max* rules, not all aggregating rules work like this, but if you have large tables it might be worth taking the improvement in time of only using one. When the *Max* rule has been executed the following fields exist: *int1*, *int2*, *int3*, *int4*, *min*, and *max*. The final aggregation rule *Selector* is used to return the minimal value of the fields *min* and *max*. This could in fact have been handled with a Min rule, but sometimes it's better to show what is happening by using rules that are commonly used for a specific purpose, like choosing between two virtual fields. As the *Selector* rule finishes the result will be stored in a virtual field Selector, which at this stage will be the only remaining field in the rule chain.

Throughout this section

4. Mapping a Data source with the ETL-Engine. We used a top-down approach, looking at the tables first, fields next, before delving into the details of the mapping language. Using Rabbit in a Hat is a means to making the ETL-Engine available to more people, while the mapping language truly is at the heart of the transformations performed by the ETL-Engine.

In the next section we will take a look at the output provided when running the ETL-Engine; how to use it for debugging and how to locate any errors.

5 TROUBLESHOOTING

In this section we will look at the output provided by the ETL-Engine. We will start by looking at the ETL-Engine itself and what output it can produce, then we will continue to the ETL-Engine log file, before concluding with the logs created for the table transformations.

On its own the ETL-Engine isn't very chatty. At most, it accepts a single argument for a local configuration file, without that argument it will use the configuration file found in `/etc/elt-engine`. If anything fails while trying to open or parse the configuration file, the ETL-Engine will write back to the terminal with information about the failure, which might look like this:

```
$ ./etl-engine etl.conf

ETL-Engine version: 1.1.0 Build d612c89

©2023 Rook-IT ApS

:

    Couldn't open file: etl.conf (-2)


Couldn't initialize etl-engine
```

Or like this:

```
$ ./etl-engine etl.conf

ETL-Engine version: 1.1.0 Build d612c89

©2023 Rook-IT ApS

Missing source connection string
```

From this it is seen that it always states the version and build number followed by any error message.

However, in terms of output, as soon as the configuration file has been loaded and the log file directory is correctly configured, this is where all other communication is going to occur. A number of files will be created Inside the directory chosen for the output in the configuration file. The first file to be created is the `etl-engine.log` file. When configuring the logging in the configuration file, a log level can be defined. This log level applies only to the `etl-engine.log` file. The etl-engine.log file will contain information about what is going on in the ETL-Engine. The different log levels that can be set all have different purpose:

- `TRACE`: Will tell a lot about what is going on when reading the Rabbit in a Hat file and building the internal representation of the mapping tables, when transforming the data, which decisions are made for optimizing table transformations and joining of tables.
- `DEBUG`: Will not be as chatty as `TRACE`, but it will still tell you a lot about what is going on, minus the internal details. This setting should be able to help you pinpoint where an error occurs.
- `INFO`: Will only print info about which stage the ETL-Engine is in, such as loading Rabbit in a Hat file, cleaning database, analyzing tables and mapping, or mapping of tables.
- `WARN`: Will only print output when the ETL-Engine experiences unexpected behavior. E.g., if the rules for a mapping couldn't be parsed, this will be reported, or if a transformation couldn't be fulfilled

because of a cascading error. These are maybe unexpected, but the ETL-Engine can continue its transformation of the source database.

- `ERROR`: Will only print output if the ETL-Engine experiences a fatal error and is forced to stop. If you are running at this level, you will only see a single line in the log file stating the cause of death to the ETL-Engine.

If the ETL-Engine suddenly stops without any output, then the `etl-engine.log` file is the place to go. Changing the log level to `DEBUG` and `TRACE` will usually provide enough information about what has caused the failure. If the errors being raised in the `etl-engine.log` occur outside of the configuration of the ETL-Engine, i.e. the beginning of the execution while the ETL-Engine and the transformation are still being configured, then errors are often of such a type that they require changes to the ETL-Engine itself.

Then the OMOP CDM table logs are much more interesting for self-servicing. These logs will appear as soon as the ETL-Engine starts to configure the transformation. The logs always show information level logs (non-configurable) about what is happening for a specific table. Which log files will appear will depend on the CDM version. But all tables of the CDM will have a log file. Therefore, you will see a log file for note.log with the following errors if you haven't done anything to fill that table.

```
note.log:[2023-05-23 15:26:38.950] [note] [error] Field 'note_id'
is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'person_id' is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'note_date' is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'note_type_concept_id' is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'note_class_concept_id' is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'note_text' is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'encoding_concept_id' is non-nullable and not mapped or defined

note.log:[2023-05-23 15:26:38.950] [note] [error] Field
'language_concept_id' is non-nullable and not mapped or defined
```

This is not something to be concerned about, as it merely states that this table cannot get any data because the required fields are not being mapped. More concerning would be something like this:

```
[2023-05-23 15:26:38.962] [device_exposure] [info] Adding field
logic for table device_exposure:  Dates(30) => :interval

 Add(:start, :interval)

 -> device_exposure_end_datetime

[2023-05-23 15:26:38.962] [device_exposure] [error] Semantic
error: The declared rule `Dates' is undefined.
```

Here the rules *Dates* and *Add* are being used in a rule chain, however the first rule is undefined, meaning that it doesn't exist, or hasn't been implemented. In this case the rule should probably have been *Days*. Upon updating the rule in the Rabbit in a Hat file, this error should no longer appear.

Any error appearing in a CDM table log file will make the last line of that file be:

> *[2023-05-23 15:26:38.962] [device_exposure] [critical] Errors break ETL of table*

This basically means that the table cannot be mapped, and that all other CDM tables depending on this table will not be mapped either. This error will cascade.

The good news is that any error happening in the CDM table log files are the result of a bad Rabbit in a Hat configuration and should be resolved by updating the mapping for the specific OMOP CDP table which corresponds to the log file where the error appeared.

BIBLIOGRAPHY

1. *Feasibility and utility of applications of the common data model to multiple, disparate observational health databases.* **Voss EA, Makadia R, Matcho A, Ma Q, Knoll C, Schuemie M, DeFalco FJ, Londhe A, Zhu V, Ryan PB.** May 2015, Journal of the American Medical Informatics Association, Volume 22, Issue 3, pp. 553-64.

2. **OHDSI.** *The Book of OHDSI: Observational Health Data Sciences and Informatics.* s.l. : OHDSI, 2019.

3. *Synthea: An approach, method, and software mechanism for generating synthetic patients and the synthetic electronic health care record.* **Jason Walonoski, Mark Kramer, Joseph Nichols, Andre Quina, Chris Moesel, Dylan Hall, Carlton Duffett, Kudakwashe Dube, Thomas Gallagher, Scott McLachlan.** March 2018, Journal of the American Medical Informatics Association, Volume 25, Issue 3, pp. 230–238.

FIGURES

APPENDIX A – TABLE RULES

Rule name	Rule information
DependOn	**Explanation:** A DependOn rule will assure that a table different from the current is already loaded when the current table is **Parameters:** 1 – The CDM table that the current table depends on to be loaded. The table must be referenced using @-nota **Requires:** - The CDM table must exist. **Examples:** DependOn(@location)
InsertRecord	**Explanation:** A static record can be inserted into a CDM table using the InsertRecord rule. **Parameters:** Each parameter must be a keyword argument in which the key is the field name that a parameter should be in will be the type of the specific field. All non-nullable fields in the table are mandatory. **Requires:** - Non-nullable fields on the table are mandatory. **Examples:** InsertRecord(person_id=1,gender_concept_id=8507,year_of_birth=2000,race_concept_id=4218674,ethnicity_ InsertRecord(cdm_source_name="Synthetic Massachusetts", cmd_source_abbreviation="Synthmas", cdm_ho source_description="Test translation of the Synthmas dataset", source_documentation_reference="https://mitre.box.com/shared/static/aw9po06ypfb9hrau4jamtvtz0e5ziucz 01-13', cdm_release_date='2023-06-30', cdm_version="5.4", cdm_version_concept_id=756265, vocabulary_ve
Join	**Explanation:** The Join rule will create merged records based on the input tables such that all possible permutations are pres introduced using a comparison key for both tables. **Parameters:** 1 – The left table to be joined. The table must be referenced using @-notation. (Mandatory) 2 – The right table to be joined. The table must be referenced using @-notation. (Mandatory) 3 – The key for the left table. This is an array of fields. All fields must be referenced using :-notation. (Optional) 4 – The key for the right table. This is an array of fields. All fields must be referenced using :-notation. (Optiona 5 – A string used to provide a name for the joined table. This can then be referenced from any subsequent Join **Requires:**

	- All referenced source tables must exist. - All referenced source fields must exist. - The same number of keys must exist for both tables. - The name used for the joined table must not exist already. **Examples:** Join(@conditions.csv, @encounter.csv,[:encounter],[:id], "ce_temp") Join(@encounter.csv, @conditions.csv) Join(@observations.csv, @patients.csv, [:patient], [:id])
LeftJoin	**Explanation:** The LeftJoin rule requires that keys are used such that a limited set is created. However, the left table will alwa **Parameters:** 1 – The left table to be joined. The table must be referenced using @-notation. (Mandatory) 2 – The right table to be joined. The table must be referenced using @-notation. (Mandatory) 3 – The key for the left table. This is an array of fields. All fields must be referenced using :-notation. (Mandato entries) 4 – The key for the right table. This is an array of fields. All fields must be referenced using :-notation. (Mandat entries) 5 – A string used to provide a name for the joined table. This can then be referenced from any subsequent Join **Requires:** - All referenced source tables must exist. - All referenced source fields must exist. - The same number of keys must exist for both tables. - A minimum of one key must be used. - The name used for the joined table must not exist already. **Examples:** LeftJoin(@patients.csv, @conditions.csv,[:id,:deathdate],[:patient, :start], "pc_temp") LeftJoin(@patients.csv, @conditions.csv,[:id],[:patient])
Normalization	**Explanation:** The Normalization rule builds a unique index on the table such that when a record is being entered into the tal record is unique according to the normalization criteria. If that is not the case, the record is skipped. **Parameters:** 1 – An array of fields within the table in :-notation that in combination must be unique for every record in the t have one or more entries) **Requires:** - All referenced CDM fields must exist.

	Examples: Normalize([:address_1,:city,:state,:zip])

APPENDIX B – MAPPING RULES

Rule name	Rule information
AsDate	**Explanation:** Transforms any type of input into a date. Integers will be interpreted as seconds, floats will be rounded and interpreted as seconds, strings will be parsed into a date using a best effort algorithm. The algorithm isn't learning, which means that it doesn't always get the dates right. To be clear on the format, a format template can be specified. **Parameters:** 1 – A format template can be given as a parameter if the input is string. A reference for the conversion specifiers available can be found at: https://en.cppreference.com/w/cpp/io/manip/get_time (Optional only for string input) **Input type:** - Date - Float - Integer - String **Output type:** - Date **Requires:** - Doesn't support conditional chaining. **Examples:** AsDate() AsDate("%d%m%y-%H%M%s")
AsFloat	**Explanation:** Transforms any type of input into a floating point. Integers will simply be cast to floats, while dates will be stored as seconds. Strings will be parsed as a float, if this is not feasible the record will fail transformation. To avoid such errors a default value can be given as a parameter. **Parameters:** 1 – A default float value which will be used if the string input cannot be parsed. This value can be NULL. (Optional) **Input type:** - Date - Float - Integer - String **Output type:**

	- Float **Requires:** - Doesn't support conditional chaining. **Examples:** AsFloat() AsFloat(0.0) AsFloat(NULL)
AsInt	**Explanation:** Transforms any type of input into an integer. Floats will be rounded down, while dates will be stored as seconds. Strings will be parsed as an integer, if this is not feasible the record will fail transformation. To avoid such errors a default value can be given as a parameter. **Parameters:** 1 – A default integer value which will be used if the string input cannot be parsed. The value can be NULL. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** AsInt() AsInt(42)
AssertNot	**Explanation:** Takes a single parameter which the input must not match. If the parameter matches this input the rule will fail. **Parameters:** 1 – A value of any type matching the input type, or NULL. (Mandatory) **Input type:** - Date - Float

	- Integer - String **Output type:** - Same as input type **Requires:** - Doesn't support conditional chaining. **Examples:** AssertNot("Error state") AssertNot(NULL) AssertNot(0.0)
AsString	**Explanation:** Transforms any type of input into a string. **Parameters:** No parameters accepted. **Input type:** - Date - Float - Integer - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:** AsString()
DayFromDate	**Explanation:** Expects a date as an input and will return an integer representing the day of the month in the date. This is useful for the day_of_birth field in the person table. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:**

	- Integer **Requires:** - Doesn't support conditional chaining. **Examples:** DayFromDate()
GetCDMTableId	**Explanation:** Maps the input using two fields in a single CDM table, one field is assigned to be a *key*, the other a *value*. If the input matches the *key* in the CDM table specified, it will be replaced with its corresponding *value*. If the *key* cannot be found in the CDM table, the record mapping will fail. If failures are unwanted a default value can be assigned. A common use of this rule is to map IDs in a source table to IDs in the OMOP CDM. In such cases the best practice is to save the source ID to the field named <table>_source_value and use that as the *key* of the GetCDMTableID rule and have the CMD table id in <table>_id, this should be used as the *value* in the GetCDMTableID rule. This particular use case is the default for the GetCDMTableID rule, and thus, it is not required to specify these two fields. **Parameters:** 1 – The CDM Table to collect the *key* and *value* fields from in @-notation. If this is the only parameter, or the second parameter is the default value, the fields will default to <table_name>_source_value for the *key* and <table_name>_id for the *value*. (Mandatory) 2 – If only two parameters are present this parameter will work as a default value in case the *key* cannot be found in the CDM table. (Optional) **– OR –** 2 – If three or four parameters are present this parameter must name the *key* field in the CDM table, in :-notation. (Optional) 3 – The *value* field in the CDM table, in :-notation. (Optional) 4 – The default value in case the *key* cannot be found in the CDM table. (Optional) **Input type:** - The input type must match that of the *key* field. **Output type:** - The output type will match that of the *value* field. **Requires:** - The table used must exist. - Either both the *key* and *value* field are specified or none of them are. - Both fields must be available in the table. - If a default value is set, this value's type must match that of the *value*.

	- Doesn't support conditional chaining. **Examples:** GetCDMTableID(@location) GetCDMTableId(@location, :location_source_value, :location_id) GetCDMTableId(@location, 0) GetCDMTableId(@location, :location_source_value, :location_id, 0)
HourFromDate	**Explanation:** Expects a date as an input and will return an integer representing the hour of the day in the date. It does not accept any parameters. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** HourFromDate()
InsertStringAt	**Explanation:** Inserts a static string into a given location in a string input. **Parameters:** 1 – The location to insert the static string in the string input, if the insert position is after the end of the input string, nothing will be inserted. 2 – The static string to insert. **Input type:** - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:**

	InsertStringAt(5, "Inserted text")
Map	**Explanation:** Maps the input based on a static mapping table created from the Map rule's parameters. **Parameters:** 1 – An array of arrays each with two entries. The first entry is the key that the input must match, and the second entry is the value that the key must be replaced with. The map rule accepts any input type and multiple key types can be present in the map, however, all output types must be identical. To match string input, the key can be a regular expression in //-notation. (Mandatory) 2 – A default value having the same type as the output types of the array of arrays map. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - The output type of the values in the array of arrays. **Requires:** - At least one entry must exist in the mapping array. - The inner arrays of the array in the first parameter must all have two entries. - The inner arrays of the array in the first parameter must all have the same output type. - A default value must have the same type as the output types of the inner arrays of the array in the first parameter. - Supports conditional chaining. **Examples:** Map([["F", 8532], ["M", 8507]]) Map([[1, 8532], [2, 8507]], 0) Map([[/F(emale)?/, 8532], [/M(ale)?/, 8507]])
MinuteFromDate	**Explanation:** Expects a date as an input and will return an integer representing the minute of the day in the date. **Parameters:** No parameters accepted. **Input type:**

	- Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** MinuteFromDate()
MonthFromDate	**Explanation:** Expects a date as an input and will return an integer representing the month of the year in the date. This is useful for the month_of_birth field in the person table. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** MonthFromDate()
PostPad	**Explanation:** Expands a string to a given size by padding it with a specified character at its end. If the input string matches the given size or is larger, it will not be changed. **Parameters:** 1 – The size that the input string should have after padding. (Mandatory) 2 – The character that should be used to pad the string. If not present, space will be used as the padding character. (Optional) **Input type:** - String **Output type:** - String

	Requires: - The padding character can only be a single character and not a string. - Doesn't support conditional chaining. **Examples:** PostPad(5, "0") PostPad(10)
PrePad	**Explanation:** Expands a string to a given size by padding it with a specified character at its beginning. If the input string matches the given size or is larger, it will not be changed. **Parameters:** 1 – The size that the input string should have after padding. (Mandatory) 2 – The character that should be used to pad the string. If not present, space will be used as the padding character. (Optional) **Input type:** - String **Output type:** - String **Requires:** - The padding character can only be a single character and not a string. - Doesn't support conditional chaining. **Examples:** PrePad(5, "0") PrePad(10)
Regex	**Explanation:** Selects the text from the input which matches a given regular expression. If the input doesn't match the regular expression an empty string will be returned. **Parameters:** 1 – The regular expression that must be matched (Mandatory) **Input type:** - String **Output type:** - String

	Requires: - Doesn't support conditional chaining. **Examples:** Regex(/\(01\)[0-9]+/) Regex(//)
SaveToVirtualField	**Explanation:** Special mapping rule to store a single value from a source table to a virtual field on a CDM table. This is needed as virtual fields are not visually represented in Rabbit in a Hat. Thus, this rule is placed on a transaction to an otherwise unrelated field, the transaction will essentially (virtually) be to the virtual field named in the rule. **Parameters:** 1 – The name of the virtual field in :-notation **Input type:** - Date - Float - Integer - String **Output type:** - Does not produce output to a field. Only the output to the virtual field. **Requires:** - Can only save to virtual fields. For actual fields make a transition in Rabbit in a Hat. - The virtual field referenced must be defined on the CDM table. - Doesn't support conditional chaining. **Examples:** SaveToVirtualField(:care_site_map)
SecondFromDate	**Explanation:** Expects a date as an input and will return an integer representing the second of the day in the date. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:**

	- Integer **Requires:** - Doesn't support conditional chaining. **Examples:** SecondFromDate()
SubString	**Explanation:** Gets a string from within the input string. If the definition of the substring is outside the inputted string, then the available characters will be returned. **Parameters:** 1 – The position of the first character of the substring. (Mandatory) 2 – The length of the substring. If not present, the string starting at the character defined in parameter 1 until the end of the input string will be returned. (Optional) **Input type:** - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:** SubString(2, 5) SubString(10)
Truncate	**Explanation:** Makes sure that an input string does not exceed a certain number of characters. If the string is shorter than the length defined, it will not be truncated. **Parameters:** 1 – The maximal length that the input string can have. (Mandatory) **Input type:** - String **Output type:** - String **Requires:**

	- Doesn't support conditional chaining. **Examples:** Truncate(5) Truncate(40)
UsagiMap	**Explanation:** Uses a Usagi map to map its input to a concept id. **Parameters:** 1 – The name of the Usagi file in the directory pointed to by the "usagi directory" clause in the ETL-Engine configuration file. (Mandatory) 2 – A default value to use when no match is found for the input in the Usagi file. By default this value is 0. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - Integer **Requires:** - Supports conditional chaining. **Examples:** UsagiMap("usagi-file.csv") UsagiMap("map.csv", 0)

VocabMap	**Explanation:**
	
	Uses the OMOP CDM vocabulary to map a vocabulary code to a standard concept id.
	
	The query used towards the database is:
	`select concept_id_1`
	`  from concept`
	`  inner join concept_relationship`
	`  on concept_id = concept_id_1`
	`  where relationship_id = 'Maps to'`
	`  and vocabulary_id = :vocabulary_id`
	`  and concept_code = :concept_code`
	
	Parameters:
	
	1 – The name of the vocabulary used in the mapping. (Mandatory)

	2 – A default value to use when there is no match for the input in the vocabulary. By default this value is `NULL`. (Optional) **Input type:** - String **Output type:** - Integer **Requires:** - Supports conditional chaining. **Examples:** VocabMap("SNOMED") VocabMap("ATC", 0)

VocabSourceId	**Explanation:**
	
	Uses the OMOP CDM vocabulary to map a vocabulary code to a source concept id.
	
	The query used towards the database is:
	`select concept_id_2`
	`  from concept`
	`  inner join concept_relationship`
	`  on concept_id = concept_id_1`
	`  where relationship_id = 'Maps to'`
	`  and vocabulary_id = :vocabulary_id`
	`  and concept_code = :concept_code`
	
	Parameters:
	
	1 – The name of the vocabulary used in the mapping. (Mandatory)
	
	2 – A default value to use when there is no match for the input in the vocabulary. By default this value is `NULL`. (Optional)
	
	Input type:
	
	- String
	
	Output type:
	
	- Integer
	
	Requires:
	
	- Doesn't support conditional chaining.
	
	Examples:
	
	VocabSourceId("SNOMED")
	
	VocabSourceId("ATC", 0)

YearFromDate	**Explanation:**

	YearFromDate expects a date as an input and will return an integer representing the year in the date. This is useful for the year_of_birth field in the person table. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** YearFromDate()

APPENDIX C – AGGREGATING RULES

Rule name	Parameters
Add	**Explanation:** Adds all values given to the command. If no values are provided, all inputs will be added. **Parameters:** *N* – The n^{th} addend of the addition. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Integer - Float - Date **Output type:** - Integer - Float - Date **Requires:** - Only one date can be given in the input. - A date cannot be added to a float. - If a float is in the input the return type will always be a float. - If a date is in the input, the return type will always be a date. **Examples:** Add() Add(1, 2) Add(:integer1, 0.6) Add(:date1, :integer2, 3600)
AsDays	**Explanation:** AsDays is a helper function to convert the integer value of seconds into the corresponding number of days – i.e. divide it by 60 * 60 * 24. The AsDays command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of days must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:**

	- Integer **Requires:** - Either a single value must be presented as an input to AsDays or a single integer parameter must be provided. **Examples:** AsDays() AsDays(3628800)
AsHours	**Explanation:** AsHours is a helper function to convert the integer value of seconds into the corresponding number of hours – i.e. divide it by 60 * 60. The AsHours command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of hours must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to AsHours or a single integer parameter must be provided. **Examples:** AsHours() AsHours(10800)
AsMinutes	**Explanation:** AsMinutes is a helper function to convert the integer value of seconds into the corresponding number of minutes – i.e. divide it by 60. The AsMinutes command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of minutes must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:**

	- Integer **Requires:** - Either a single value must be presented as an input to AsMinutes or a single integer parameter must be provided. **Examples:** AsMinutes() AsMinutes(240)
AsMonths	**Explanation:** AsMonths is a helper function to convert the integer value of seconds into the corresponding number of months – i.e. divide it by 60 * 60 * 24 * 30. The AsMonths command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of months must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to AsMonths or a single integer parameter must be provided. **Examples:** AsMonths() AsMonths(38880000)
AsSeconds	**Explanation:** AsSeconds is a helper function to convert an integer value of seconds into the corresponding number of seconds. The integer value is in seconds, so the command only passes the value through, it merely exists to complete the set of commands: AsMinutes, AsHours, AsDays, AsMonths, and AsYears. The AsSeconds command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer

	Output type: - Integer **Requires:** - Either a single value must be presented as an input to AsSeconds or a single integer parameter must be provided. **Examples:** AsSeconds() AsSeconds(53)
AsYears	**Explanation:** AsYears is a helper function to convert the integer value of seconds into the corresponding number of years – i.e. divide it by 60 * 60 * 24* 365. The AsYears command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of years must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to AsYears or a single integer parameter must be provided. **Examples:** AsYears() AsYears(94608000)
AutoGenId	**Explanation:** Creates an ID sequence. The sequence can either be a sequence for the field, meaning a sequence unique for the field it is on, or a named sequence, in which case it is unique for the name. **Parameters:** 1 – The sequence name if any. (Optional) **Input type:** - Does not accept input **Output type:**

	- Integer **Requires:** - Doesn't have any hard requirements. **Examples:** AutoGenId() AutoGenId("My Sequence")
Concat	**Explanation:** Concatenates any number of strings. The input strings must be references in the correct order in the parameters. Static strings can also be used. **Parameters:** *N* – The N^{th} component that should be part of the resulting string, this can be either a static string or a reference to a string field in :-notation. (Mandatory) **Input type:** - String **Output type:** - String **Requires:** - Concat works both with and without inputs. Without inputs it can only contain static strings. **Examples:** Concat(:field1, " – ", :field2) Concat("I work like SetValue") Concat("Field7 contains: ", :field7) Concat(:field1, :field2, :field3)
Days	**Explanation:** Days is a helper function to create the integer value for a specified number of days. The integer value is in seconds. The Days command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of days to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:**

	- Integer **Requires:** - Either a single value must be presented as an input to Days or a single integer parameter must be provided. **Examples:** Days() Days(42)
Divide	**Explanation:** Divides a specified value by all additional values given to the command. If no additional values are provided, the specified value will be divided by all available values. **Parameters:** 1 – The value specified as the dividend of the division. This can be either an input in :-notation or a static value. *N* – The n^{th} divisor that the dividend must be divided by. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Float - Integer **Output type:** - Float **Requires:** - Doesn't have any hard requirements. **Examples:** Divide(:dividend) Divide(:dividend, :divisor1, :divisor2) Divide(:dividend, 10) Divide(:dividend, 3.14, :divisor2)
Hours	**Explanation:** Hours is a helper function to create the integer value for a specified number of hours. The integer value is in seconds. The Hours command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of hours to calculate the number of seconds for. If not present a single integer input must be present. (Optional)

	Input type: - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Hours or a single integer parameter must be provided. **Examples:** Hours() Hours(19)
Max	**Explanation:** Selects the highest value of all values given to it. If no parameters are given, it selects between all inputs, if parameters are given it only choses between the parameters. **Parameters:** 1 – The first value to check. This can be either an input type in :-notation or a static value. (Optional) *N* – The n^{th} value to check. This can be either an input type in :-notation or a static value. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - Same type as the input type. **Requires:** - All parameter types and input types must be identical. **Examples:** Max() Max(:value1, :value2) Max(:value, 42) Max(:text1, :text2, "Zulu")
MergeDate	**Explanation:**

	Combines several integers or dates into a single date. Each integer must be marked with its role in the date. These roles are: - date: input type Date - time: input type Date - year: input type Integer - month: input type Integer - day: input type Integer - hour: input type Integer - minute: input type Integer - second: input type Integer **Parameters:** 1 – An array of two entries – the first entry must be the field to use in :-notation, the second must be the role of the input in the date returned. (Mandatory) 2 – An array of two entries – the first entry must be the field to use in :-notation, the second must be the role of the input in the date returned. (Mandatory) *3..6* – An array of two entries – the first entry must be the field to use in :-notation, the second must be the role of the input in the date returned. (Optional) **Input type:** - Date - Integer **Output type:** - Date **Requires:** - The role of the entries cannot be outside that of date, hour, year, month, day, hour, minute and second. **Examples:** MergeDate([:start_date, "date"], [:start_hour, "time"]) MergeDate([:year, "year"], [:month, "month"],[:day, "day"], [:hour, "hour"],[:minute, "minute"], [:second, "second"])
Min	**Explanation:** Selects the lowest value of all values given to it. If no parameters are given, it selects between all inputs, if parameters are given it only choses between the parameters. **Parameters:** 1 – The first value to check. This can be either an input type in :-notation or a static value. (Optional) *N* – The n^{th} value to check. This can be either an input type in :-notation or a static value. (Optional) **Input type:** - Date

	- Float - Integer - String **Output type:** - Same type as the input type. **Requires:** - All parameter types and input types must be identical. **Examples:** Min() Min(:value1, :value2) Min(:value, 42) Min(:text1, :text2, "Alpha")
Minutes	**Explanation:** Minutes is a helper function to create the integer value for a specified number of minutes. The integer value is in seconds. The Minutes command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of minutes to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Minutes or a single integer parameter must be provided. **Examples:** Minutes() Minutes(240)
Months	**Explanation:** Months is a helper function to create the integer value for a specified number of months. The integer value is in seconds. The Months command either accepts a single input or it accepts a static parameter. **Parameters:**

	1 – The number of months to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Months or a single integer parameter must be provided. - A single month is expected to be 30 days. Thus, 12 months is 360 days. **Examples:** Months() Months(12)
Multiply	**Explanation:** Multiplies all values given to the command. If no values are provided, all inputs will be multiplied. **Parameters:** 1 – The multiplier of the multiplication. This can be either an input in :-notation or a static value. (Optional) *N* – The n^{th} multiplicand that the multiplier must be multiplied by. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Float - Integer **Output type:** - Float (If any float was given as input) - Integer (If no floats were given as input) **Requires:** - Doesn't have any hard requirements. **Examples:** Multiply() Multiply(:multiplier, :multiplicand1, :multiplicand2) Multiply(10, 10, 10) Multiply(4.2, :multiplicand)

Seconds	**Explanation:** Seconds is a helper function to create the integer value for a specified number of seconds. The integer value is in seconds, so the command only passes the value through, it merely exists to complete the set of commands: Minutes, Hours, Days, Months, and Years. The Seconds command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Seconds or a single integer parameter must be provided. **Examples:** Seconds() Seconds(53)
Selector	**Explanation:** Based on a comparison between two values a choice is either made between one value and `NULL` or it is made between two values. **Parameters:** 1 – The first value of the comparison. This can be either a static value or a field reference in :-notation. (Mandatory) 2 – The comparison operator, this must be a static string value of one of the following: - < - The first value is less than the second. - = - The first value is equal the second. - > - The first value is greater than the second. - <= - The first value is less than or equal to the second. - <> - The first value is not equal to the second. - >= - The first value is greater than or equal to the second. (Mandatory) 3 – The second value of the comparison. This can be either a static value or a field reference in :-notation. (Mandatory) 4 – The value chosen if the comparison evaluates to true. This can be either a static value or a field reference in :-notation. (Mandatory)

	5 - The value chosen if the comparison evaluates to true. This can be either a static value or a field reference in :-notation. If unset the value will be `NULL`. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - The same as the fourth parameter's type. **Requires:** - The first and the third parameter must have the same type. - If a fifth parameter is present if must have the same type as the fourth parameters. - The second parameter must be one of: <, =, >, <=, <>, and >=. **Examples:** Selector(:int1, "=", :int2, "Equal", "Unequal") Selector(:str1, "<", "E", :str2) Selector(:date1, "<>", :date2, "Unequal dates", "Equal dates") Selector(:float1, ">", :float2, :str1)
SetValue	**Explanation:** Creates an output from a parameter value. **Parameters:** 1 – The value to set to the output. (Mandatory) **Input type:** - Does not accept input **Output type:** - The type of the value in the first parameter. **Requires:** - Doesn't have any hard requirements. **Examples:** SetValue(42) SetValue("My Value")
Subtract	**Explanation:**

	Subtracts a specified value by all additional values given to the command. If no additional values are provided, the specified value will be subtracted by all available values. **Parameters:** 1 – The value specified as the minuend of the subtraction. This can be either an input in :-notation or a static value. *N* – The n^{th} subtrahend that must be subtracted from the minuend. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Integer - Float - Date **Output type:** - Integer - Float - Date **Requires:** - At most two dates can be given in the input. - A float cannot be subtracted from a date. - A date can neither be subtracted from a float nor an integer. - If a float is in the input the return type will always be a float. - If a single date is in the input, the return type will be a date. - If two dates are in the input, the return type will be an integer. **Examples:** Subtract(42) Subtract(:float, 42) Subtract(:date1, :date2) Subtract(:integer, 1000, :float)
Use	**Explanation:** When having multiple source fields mapped to a CDM field, or having created multiple local field variables in a calculation this rule selects the one that will go in to the CDM field. This command is special as it will always remove the input fields and only leave one field as an output of the rule. Even if it has an output field defined. **Parameters:** 1 – The field or local field variable in :-notation to commit to the CDM field. **Input type:** - Date - Float - Integer

	- String **Output type:** - The same as the input type. **Requires:** - Doesn't have any hard requirements. **Examples:** Use(:Result) Use(:location_id) Use(:Result) => :Result The line above will leave the :Result field in the rule chain, and it can be referenced as :Result.
Years	**Explanation:** Years is a helper function to create the integer value for a specified number of years. The integer value is in seconds. The Years command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of years to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Years or a single integer parameter must be provided. - A single year is expected to be 365 days. **Examples:** Years() Years(2)

APPENDIX D – LICENSES FOR USED LIBRARIES

ETL-Engine depends on a number of libraries to function. This appendix contains the library licenses and an explanation of the use of each library.

BOOST

Boost is used for the json parser and the object value model throughout the ETL-Engine.

> *Boost Software License - Version 1.0 - August 17th, 2003*
>
> *Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:*
>
> *The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.*
>
> *THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.*

CONFIGTOOL

ConfigTool is used to create and manage the parser for the etl.conf file.

> *"THE BEER-WARE LICENSE" (Revision 42):*
>
> *<carsten@stiborg.dk> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return. Carsten Stiborg*

CSV2

csv2 is used as the input parser for Usagi files and database input CSV files. It is also used to create CSV output.

FMT

fmt is used to format log output.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

--- Optional exception to the license ---

As an exception, if, as a result of your compiling your source code, portions of this Software are embedded into a machine-executable object form of such source code, you may redistribute such embedded portions in such object form without including the above copyright and permission notices.

MARIADB

MariaDB is used to interact with the MariaDB and it is also compatible with the MySQL database.

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom
to share and change it. By contrast, the GNU General Public Licenses
are intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You

can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations
below.

When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you. You must make sure that they, too, receive or can get the source
code. If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there
is no warranty for the free library. Also, if the library is modified
by someone else and passed on, the recipients should know that what
they have is not the original version, so that the original author's

reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it

*becomes a de-facto standard. To achieve this, non-free programs must
be allowed to use the library. A more frequent case is that a free
library does the same job as widely used non-free libraries. In this
case, there is little to gain by limiting the free library to free
software only, so we use the Lesser General Public License.*

*In other cases, permission to use a particular library in non-free
programs enables a greater number of people to use a large body of
free software. For example, permission to use the GNU C Library in
non-free programs enables many more people to use the whole GNU
operating system, as well as its variant, the GNU/Linux operating
system.*

*Although the Lesser General Public License is Less protective of the
users' freedom, it does ensure that the user of a program that is
linked with the Library has the freedom and the wherewithal to run
that program using a modified version of the Library.*

*The precise terms and conditions for copying, distribution and
modification follow. Pay close attention to the difference between a
"work based on the library" and a "work that uses the library". The
former contains code derived from the library, whereas the latter must
be combined with the library in order to run. TERMS AND CONDITIONS
FOR COPYING, DISTRIBUTION AND MODIFICATION*

*0. This License Agreement applies to any software library or other
program which contains a notice placed by the copyright holder or
other authorized party saying it may be distributed under the terms of
this Lesser General Public License (also called "this License"). Each
licensee is addressed as "you".*

*A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.*

*The "Library", below, refers to any such software library or work
which has been distributed under these terms. A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language. (Hereinafter, translation is
included without limitation in the term "modification".)*

*"Source code" for a work means the preferred form of the work for
making modifications to it. For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control
compilation and installation of the library.*

*Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope. The act of
running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it). Whether that is true depends on what the Library does and
what the program that uses the Library does.*

*1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.*

*You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a
fee.*

2. You may modify your copy or copies of the Library or any portion of

it, thus forming a work based on the Library, and copy and distribute
such modifications or work under the terms of Section 1 above,
provided that you also meet all of these conditions:

*   * a) The modified work must itself be a software library.*

*   * b) You must cause the files modified to carry prominent notices*
*   stating that you changed the files and the date of any change.*

*   * c) You must cause the whole of the work to be licensed at no*
*   charge to all third parties under the terms of this License.*

*   * d) If a facility in the modified Library refers to a function or*
*   a table of data to be supplied by an application program that*
*   uses the facility, other than as an argument passed when the*
*   facility is invoked, then you must make a good faith effort to*
*   ensure that, in the event an application does not supply such*
*   function or table, the facility still operates, and performs*
*   whatever part of its purpose remains meaningful.*

*   (For example, a function in a library to compute square roots*
*   has a purpose that is entirely well-defined independent of the*
*   application. Therefore, Subsection 2d requires that any*
*   application-supplied function or table used by this function*
*   must be optional: if the application does not supply it, the*
*   square root function must still compute square roots.)*

These requirements apply to the modified work as a whole. If
identifiable sections of that work are not derived from the Library,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works. But when you
distribute the same sections as part of a whole which is a work based
on the Library, the distribution of the whole must be on the terms of

this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Library.

In addition, mere aggregation of another work not based on the Library
with the Library (or with a work based on the Library) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public
License instead of this License to a given copy of the Library. To do
this, you must alter all the notices that refer to this License, so
that they refer to the ordinary GNU General Public License, version 2,
instead of to this License. (If a newer version than version 2 of the
ordinary GNU General Public License has appeared, then you can specify
that version instead if you wish.) Do not make any other change in
these notices.

Once this change is made in a given copy, it is irreversible for that
copy, so the ordinary GNU General Public License applies to all
subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the
Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative
of it, under Section 2) in object code or executable form under the
terms of Sections 1 and 2 above provided that you accompany it with
the complete corresponding machine-readable source code, which must be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6,

whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link
a "work that uses the Library" with the Library to produce a work
containing portions of the Library, and distribute that work under
terms of your choice, provided that the terms permit modification of
the work for the customer's own use and reverse engineering for
debugging such modifications.

You must give prominent notice with each copy of the work that the
Library is used in it and that the Library and its use are covered by
this License. You must supply a copy of this License. If the work
during execution displays copyright notices, you must include the
copyright notice for the Library among them, as well as a reference
directing the user to the copy of this License. Also, you must do one
of these things:

* *a) Accompany the work with the complete corresponding*
  machine-readable source code for the Library including whatever
  changes were used in the work (which must be distributed under
  Sections 1 and 2 above); and, if the work is an executable
  linked with the Library, with the complete machine-readable
  "work that uses the Library", as object code and/or source code,
  so that the user can modify the Library and then relink to
  produce a modified executable containing the modified
  Library. (It is understood that the user who changes the
  contents of definitions files in the Library will not
  necessarily be able to recompile the application to use the
  modified definitions.)
* *b) Use a suitable shared library mechanism for linking with the*
  Library. A suitable mechanism is one that (1) uses at run time a
  copy of the library already present on the user's computer
  system, rather than copying library functions into the

*executable, and (2) will operate properly with a modified
version of the library, if the user installs one, as long as the
modified version is interface-compatible with the version that
the work was made with.
* c) Accompany the work with a written offer, valid for at least
three years, to give the same user the materials specified in
Subsection 6a, above, for a charge no more than the cost of
performing this distribution.
* d) If distribution of the work is made by offering access to
copy from a designated place, offer equivalent access to copy
the above specified materials from the same place.
* e) Verify that the user has already received a copy of these
materials or that you have already sent this user a copy.*

*For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it. However, as a special exception,
the materials to be distributed need not include anything that is
normally distributed (in either source or binary form) with the major
components (compiler, kernel, and so on) of the operating system on
which the executable runs, unless that component itself accompanies
the executable.*

*It may happen that this requirement contradicts the license
restrictions of other proprietary libraries that do not normally
accompany the operating system. Such a contradiction means you cannot
use both them and the Library together in an executable that you
distribute.*

*7. You may place library facilities that are a work based on the
Library side-by-side in a single library together with other library
facilities not covered by this License, and distribute such a combined
library, provided that the separate distribution of the work based on
the Library and of the other library facilities is otherwise
permitted, and provided that you do these two things:*

*   * a) Accompany the combined library with a copy of the same work
      based on the Library, uncombined with any other library
      facilities. This must be distributed under the terms of the
      Sections above.

    * b) Give prominent notice with the combined library of the fact
      that part of it is a work based on the Library, and explaining
      where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the
Library except as expressly provided under this License. Any attempt
otherwise to copy, modify, sublicense, link with, or distribute the
Library is void, and will automatically terminate your rights under
this License. However, parties who have received copies, or rights,
from you under this License will not have their licenses terminated so
long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not
signed it. However, nothing else grants you permission to modify or
distribute the Library or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by
modifying or distributing the Library (or any work based on the
Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the
Library), the recipient automatically receives a license from the
original licensor to copy, distribute, link with or modify the Library
subject to these terms and conditions. You may not impose any further
restrictions on the recipients' exercise of the rights granted
herein. You are not responsible for enforcing compliance by third
parties with this License.

11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License. If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all. For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply, and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system which is
implemented by public license practices. Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the

original copyright holder who places the Library under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded. In such case, this License incorporates
the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new
versions of the Lesser General Public License from time to time. Such
new versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library
specifies a version number of this License which applies to it and
"any later version", you have the option of following the terms and
conditions either of that version or of any later version published by
the Free Software Foundation. If the Library does not specify a
license version number, you may choose any version ever published by
the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free
programs whose distribution conditions are incompatible with these,
write to the author to ask for permission. For software which is
copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this. Our
decision will be guided by the two goals of preserving the free status
of all derivatives of our free software and of promoting the sharing
and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE
LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS
AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF
ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES. END OF TERMS AND CONDITIONS.

MINIZIP

Minizip is used to read the files structure in the vocabulary files from Athena.

Condition of use and distribution are the same than zlib :

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

3. The origin of this software must not be misrepresented; you must not

claim that you wrote the original software. If you use this software
in a product, an acknowledgment in the product documentation would be
appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be

```
     misrepresented as being the original software.
  3. This notice may not be removed or altered from any
source distribution.
```

MIO

mio is used in relation to the CSV parser, allowing data to be read into memory for random access of csv fields in a fast manner.

```
MIT License

Copyright © 2018 https://github.com/mandreyel/

Permission is hereby granted, free of charge, to any
person obtaining a copy of this software and associated
documentation files (the "Software"), to deal in the
Software without restriction, including without
limitation the rights to use, copy, modify, merge,
publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software
is furnished to do so, subject to the following
conditions:

The above copyright notice and this permission notice
shall be included in all copies or substantial portions
of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF
ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED
TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR
IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

PCRE

The Perl compatible regular expressions library is used in the Regex and Map rules as well as the configuration file field validation.

```
Redistribution and use in source and binary forms, with
or without
modification, are permitted provided that the following
conditions are met:

    * Redistributions of source code must retain the
above copyright notices,
      this list of conditions and the following
disclaimer.

    * Redistributions in binary form must reproduce the
above copyright
```

notices, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

** Neither the name of the University of Cambridge nor the names of any*
contributors may be used to endorse or promote products derived from this
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

EXEMPTION FOR BINARY LIBRARY-LIKE PACKAGES

The second condition in the BSD licence (covering binary redistributions) does
not apply all the way down a chain of software. If binary package A includes
PCRE2, it must respect the condition, but if package B is software that
includes package A, the condition is not imposed on package B unless it uses
PCRE2 independently.

POSTGRESQL

PostgreSQL is used to interact with the PostgreSQL database.

PostgreSQL is released under the PostgreSQL License, a liberal Open Source license, similar to the BSD or MIT licenses.

PostgreSQL Database Management System (formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2023, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

SOCI-4.0.3

SOCI is used as a higher level database abstraction layer to communicate with several databases without having to implement individual interfaces for each database supported.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices ©n the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE ©S PROVIDED "AS ©S", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

SPDLOG

spdlog is used throughout the ETL-Engine for logging.

The MIT License (MIT)

Copyright © 2016 Gabi Melman.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

-- NOTE: Third party dependency used by this software -

This software depends on the fmt lib (MIT License), and users must comply to its license: https://github.com/fmtlib/fmt/blob/master/LICENSE.rst

ZLIB

ZLIB is used to unzip the vocabulary file from Athena and the Rabbit in a Hat file.

```
zlib.h -- interface of the 'zlib' general purpose compression library
  version 1.3, August 18th, 2022

  Copyright (C) 1995-2023 Jean-loup Gailly and Mark Adler

  This software is provided 'as-is', without any express or implied
  warranty.  In no event will the authors be held liable for any damages
  arising from the use of this software.

  Permission is granted to anyone to use this software for any purpose,
  including commercial applications, and to alter it and redistribute it
  freely, subject to the following restrictions:

  1. The origin of this software must not be misrepresented; you must not
     claim that you wrote the original software. If you use this software
     in a product, an acknowledgment in the product documentation would be
     appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be
     misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.

  Jean-loup Gailly        Mark Adler
  jloup@gzip.org          madler@alumni.caltech.edu
```

APPENDIX A – TABLE RULES

Rule name	Rule information
DependOn	**Explanation:** A DependOn rule will assure that a table different from the current is already loaded when the current table is being transformed. **Parameters:** 1 – The CDM table that the current table depends on to be loaded. The table must be referenced using @-notation. (Mandatory) **Requires:** - The CDM table must exist. **Examples:** DependOn(@location)
InsertRecord	**Explanation:** A static record can be inserted into a CDM table using the InsertRecord rule. **Parameters:** Each parameter must be a keyword argument in which the key is the field name that a parameter should be inserted into. The type of the argument will be the type of the specific field. All non-nullable fields in the table are mandatory. **Requires:** - Non-nullable fields on the table are mandatory. **Examples:** InsertRecord(person_id=1,gender_concept_id=8507,year_of_birth=2000,race_concept_id=4218674,ethnicity_concept_id=38003564,location_id=1) InsertRecord(cdm_source_name="Synthetic Massachusetts", cmd_source_abbreviation="Synthmas", cdm_holder="Rook IT ApS", source_description="Test translation of the Synthmas dataset", source_documentation_reference="https://mitre.box.com/shared/static/aw9po06ypfb9hrau4jamtvtz0e5ziucz.zip", source_release_date='2023-01-13', cdm_release_date='2023-06-30', cdm_version="5.4", cdm_version_concept_id=756265, vocabulary_version="v5.0 31-MAY-23")
Join	**Explanation:** The Join rule will create merged records based on the input tables such that all possible permutations are present. Limitations to this scope can be introduced using a comparison key for both tables. **Parameters:** 1 – The left table to be joined. The table must be referenced using @-notation. (Mandatory) 2 – The right table to be joined. The table must be referenced using @-notation. (Mandatory)

	3 – The key for the left table. This is an array of fields. All fields must be referenced using :- notation. (Optional) 4 – The key for the right table. This is an array of fields. All fields must be referenced using :- notation. (Optional) 5 – A string used to provide a name for the joined table. This can then be referenced from any subsequent Join or LeftJoin rules. (Optional) **Requires:** - All referenced source tables must exist. - All referenced source fields must exist. - The same number of keys must exist for both tables. - The name used for the joined table must not exist already. **Examples:** Join(@conditions.csv, @encounter.csv,[:encounter],[:id], "ce_temp") Join(@encounter.csv, @conditions.csv) Join(@observations.csv, @patients.csv, [:patient], [:id])
LeftJoin	**Explanation:** The LeftJoin rule requires that keys are used such that a limited set is created. However, the left table will always be present in its entirety. **Parameters:** 1 – The left table to be joined. The table must be referenced using @-notation. (Mandatory) 2 – The right table to be joined. The table must be referenced using @-notation. (Mandatory) 3 – The key for the left table. This is an array of fields. All fields must be referenced using :- notation. (Mandatory and array must have one or more entries) 4 – The key for the right table. This is an array of fields. All fields must be referenced using :- notation. (Mandatory and array must have one or more entries) 5 – A string used to provide a name for the joined table. This can then be referenced from any subsequent Join or LeftJoin rules. (Optional) **Requires:** - All referenced source tables must exist. - All referenced source fields must exist. - The same number of keys must exist for both tables. - A minimum of one key must be used. - The name used for the joined table must not exist already. **Examples:** LeftJoin(@patients.csv, @conditions.csv,[:id,:deathdate],[:patient, :start], "pc_temp") LeftJoin(@patients.csv, @conditions.csv,[:id],[:patient])

Normalization	**Explanation:** The Normalization rule builds a unique index on the table such that when a record is being entered into the table a check is performed that the record is unique according to the normalization criteria. If that is not the case, the record is skipped. **Parameters:** 1 – An array of fields within the table in :-notation that in combination must be unique for every record in the table. (Mandatory and array must have one or more entries) **Requires:** - All referenced CDM fields must exist. **Examples:** Normalize([:address_1,:city,:state,:zip])
StoreIndexMap	**Explanation:** When storing an index map, an index and its associated values are store to disk, such that the index can be used in a later run of the ETL-Engine. Storing indexes can be useful when splitting the ETL over multiple Rabbit in a Hat files and when using the GetCDMTableId mapping rule. **Parameters:** 1 – An array of fields within the table in :-notation that defines the index. 2 – An array of fields within the table in :-notation that defines the fields that the index maps to. **Requires:** - All referenced CDM fields must exist. **Examples:** StoreIndexMap([:person_source_value], [:person_id])
StoreSequence	**Explanation:** When storing a sequence, a global sequence defined by the AutoGenId aggregation rule is stored to disk, such that it can be reused in a later ETL run. This is useful when using AutoGenId and populating a table over multiple ETL-Engine runs. **Parameters:** 1 – The name of the global ID. **Requires:** - An existing global ID with the defined name. **Examples:** StoreSequence("drug_exposure_seq")

APPENDIX B – MAPPING RULES

Rule name	Rule information
AsDate	**Explanation:** Transforms any type of input into a date. Integers will be interpreted as seconds, floats will be rounded and interpreted as seconds, strings will be parsed into a date using a best effort algorithm. The algorithm isn't learning, which means that it doesn't always get the dates right. To be clear on the format, a format template can be specified. **Parameters:** 1 – A format template can be given as a parameter if the input is string. A reference for the conversion specifiers available can be found at: https://en.cppreference.com/w/cpp/io/manip/get_time (Optional only for string input) **Input type:** - Date - Float - Integer - String **Output type:** - Date **Requires:** - Doesn't support conditional chaining. **Examples:** AsDate() AsDate("%d%m%y-%H%M%s")
AsFloat	**Explanation:** Transforms any type of input into a floating point. Integers will simply be cast to floats, while dates will be stored as seconds. Strings will be parsed as a float, if this is not feasible the record will fail transformation. To avoid such errors a default value can be given as a parameter. **Parameters:** 1 – A default float value which will be used if the string input cannot be parsed. This value can be NULL. (Optional) **Input type:** - Date - Float - Integer - String **Output type:**

	- Float **Requires:** - Doesn't support conditional chaining. **Examples:** AsFloat() AsFloat(0.0) AsFloat(NULL)
AsInt	**Explanation:** Transforms any type of input into an integer. Floats will be rounded down, while dates will be stored as seconds. Strings will be parsed as an integer, if this is not feasible the record will fail transformation. To avoid such errors a default value can be given as a parameter. **Parameters:** 1 – A default integer value which will be used if the string input cannot be parsed. The value can be NULL. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** AsInt() AsInt(42)
Assert	**Explanation:** Takes either a single parameter which the input must match or an array of parameters which must be matched. If any parameter doesn't match this input the rule will fail. **Parameters:** 1 – A value of any type matching the input type, NULL, or an array of values with identical types as the input – the input must match either of these values. (Mandatory)

	2 – A comment that is added to the log whenever the assert statement doesn't match. Must be a string. (Optional) **Input type:** - Boolean - Date - Float - Integer - String **Output type:** - Same as input type **Requires:** - Doesn't support conditional chaining. **Examples:** Assert("Error state", "An error state was entered") Assert (NULL) Assert ([0.0, 0.1, 0.2], "Out of limits")
AssertNot	**Explanation:** Takes either a single parameter which the input must not match or an array of parameters which must not be matched. If any parameter doesn't match this input the rule will fail. **Parameters:** 1 – A value of any type matching the input type, NULL, or an array of values with identical types as the input – the input must not match either of these values. The default value is NULL. (Optional) 2 – A comment that is added to the log whenever the assert statement doesn't match. Must be a string. (Optional) **Input type:** - Boolean - Date - Float - Integer - String **Output type:** - Same as input type **Requires:** - Doesn't support conditional chaining.

	Examples: AssertNot("Error state") AssertNot(NULL, "Null value seen") AssertNot([0, 1, 2])
AssertVocabDomain	**Explanation:** Takes a single parameter which the input must not match. If the parameter matches this input the rule will fail. **Parameters:** 1 – A string holding the name of a domain. The input concept id must be in this domain. (Mandatory) 2 – A comment that is added to the log whenever the assert statement doesn't match. Must be a string. (Optional) **Input type:** - Integer **Output type:** - Same as input type **Requires:** - Doesn't support conditional chaining. **Examples:** AssertVocabDomain("Condition")
AsString	**Explanation:** Transforms any type of input into a string. **Parameters:** No parameters accepted. **Input type:** - Boolean - Date - Float - Integer - String **Output type:** - String **Requires:** - Doesn't support conditional chaining.

	Examples: AsString()
DayFromDate	**Explanation:** Expects a date as an input and will return an integer representing the day of the month in the date. This is useful for the day_of_birth field in the person table. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** DayFromDate()
GetCDMTableId	**Explanation:** Maps the input using two fields in a single CDM table, one field is assigned to be a *key*, the other a *value*. If the input matches the *key* in the CDM table specified, it will be replaced with its corresponding *value*. If the *key* cannot be found in the CDM table, the record mapping will fail. If failures are unwanted a default value can be assigned. A common use of this rule is to map IDs in a source table to IDs in the OMOP CDM. In such cases the best practice is to save the source ID to the field named <table>_source_value and use that as the *key* of the GetCDMTableID rule and have the CMD table id in <table>_id, this should be used as the *value* in the GetCDMTableID rule. This particular use case is the default for the GetCDMTableID rule, and thus, it is not required to specify these two fields. **Parameters:** 1 – The CDM Table to collect the *key* and *value* fields from in @-notation. If this is the only parameter, or the second parameter is the default value, the fields will default to <table_name>_source_value for the *key* and <table_name>_id for the *value*. (Mandatory) 2 – If only two parameters are present this parameter will work as a default value in case the *key* cannot be found in the CDM table. (Optional) **– OR –** 2 – If three or four parameters are present this parameter must name the *key* field in the CDM table, in :-notation. (Optional)

	3 – The *value* field in the CDM table, in :-notation. (Optional) 4 – The default value in case the *key* cannot be found in the CDM table. (Optional) **Input type:** - The input type must match that of the *key* field. **Output type:** - The output type will match that of the *value* field. **Requires:** - The table used must exist. - Either both the *key* and *value* field are specified or none of them are. - Both fields must be available in the table. - If a default value is set, this value's type must match that of the *value*. - Doesn't support conditional chaining. **Examples:** GetCDMTableId(@location) GetCDMTableId(@location, :location_source_value, :location_id) GetCDMTableId(@location, 0) GetCDMTableId(@location, :location_source_value, :location_id, 0)
HourFromDate	**Explanation:** Expects a date as an input and will return an integer representing the hour of the day in the date. It does not accept any parameters. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** HourFromDate()
InsertStringAt	**Explanation:** Inserts a static string into a given location in a string input.

	Parameters: 1 – The location to insert the static string in the string input, if the insert position is after the end of the input string, nothing will be inserted. 2 – The static string to insert. **Input type:** - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:** InsertStringAt(5, "Inserted text")
Map	**Explanation:** Maps the input based on a static mapping table created from the Map rule's parameters. **Parameters:** 1 – An array of arrays each with two entries. The first entry is the key that the input must match, and the second entry is the value that the key must be replaced with. The map rule accepts any input type and multiple key types can be present in the map, however, all output types must be identical. To match string input, the key can be a regular expression in //-notation. (Mandatory) 2 – A default value having the same type as the output types of the array of arrays map. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - The output type of the values in the array of arrays. **Requires:** - At least one entry must exist in the mapping array. - The inner arrays of the array in the first parameter must all have two entries. - The inner arrays of the array in the first parameter must all have the same output type. - A default value must have the same type as the output types of the inner arrays of the array in the first parameter. - Supports conditional chaining.

	Examples: Map([["F", 8532], ["M", 8507]]) Map([[1, 8532], [2, 8507]], 0) Map([[/F(emale)?/, 8532], [/M(ale)?/, 8507]])
MinuteFromDate	**Explanation:** Expects a date as an input and will return an integer representing the minute of the day in the date. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** MinuteFromDate()
MonthFromDate	**Explanation:** Expects a date as an input and will return an integer representing the month of the year in the date. This is useful for the month_of_birth field in the person table. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** MonthFromDate()
Not	**Explanation:**

	Reverses the value of a boolean. **Parameters:** No parameters accepted. **Input type:** - Boolean **Output type:** - Boolean **Requires:** - Doesn't support conditional chaining. **Examples:** Not()
PostPad	**Explanation:** Expands a string to a given size by padding it with a specified character at its end. If the input string matches the given size or is larger, it will not be changed. **Parameters:** 1 – The size that the input string should have after padding. (Mandatory) 2 – The character that should be used to pad the string. If not present, space will be used as the padding character. (Optional) **Input type:** - String **Output type:** - String **Requires:** - The padding character can only be a single character and not a string. - Doesn't support conditional chaining. **Examples:** PostPad(5, "0") PostPad(10)
PrePad	**Explanation:** Expands a string to a given size by padding it with a specified character at its beginning. If the input string matches the given size or is larger, it will not be changed. **Parameters:**

	1 – The size that the input string should have after padding. (Mandatory) 2 – The character that should be used to pad the string. If not present, space will be used as the padding character. (Optional) **Input type:** - String **Output type:** - String **Requires:** - The padding character can only be a single character and not a string. - Doesn't support conditional chaining. **Examples:** PrePad(5, "0") PrePad(10)
Regex	**Explanation:** Selects the text from the input which matches a given regular expression. If the input doesn't match the regular expression an empty string will be returned. **Parameters:** 1 – The regular expression that must be matched (Mandatory) **Input type:** - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:** Regex(/\(01\)[0-9]+/) Regex(//)
SaveToVirtualField	**Explanation:** Special mapping rule to store a single value from a source table to a virtual field on a CDM table. This is needed as virtual fields are not visually represented in Rabbit in a Hat. Thus, this rule is placed on a transaction to an otherwise unrelated field, the transaction will essentially (virtually) be to the virtual field named in the rule.

	Parameters: 1 – The name of the virtual field in :-notation **Input type:** - Date - Float - Integer - String **Output type:** - Does not produce output to a field. Only the output to the virtual field. **Requires:** - Can only save to virtual fields. For actual fields make a transition in Rabbit in a Hat. - The virtual field referenced must be defined on the CDM table. - Doesn't support conditional chaining. **Examples:** SaveToVirtualField(:care_site_map)
SecondFromDate	**Explanation:** Expects a date as an input and will return an integer representing the second of the day in the date. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:** SecondFromDate()
SubString	**Explanation:** Gets a string from within the input string. If the definition of the substring is outside the inputted string, then the available characters will be returned. **Parameters:** 1 – The position of the first character of the substring. (Mandatory)

	2 – The length of the substring. If not present, the string starting at the character defined in parameter 1 until the end of the input string will be returned. (Optional) **Input type:** - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:** SubString(2, 5) SubString(10)
Truncate	**Explanation:** Makes sure that an input string does not exceed a certain number of characters. If the string is shorter than the length defined, it will not be truncated. **Parameters:** 1 – The maximal length that the input string can have. (Mandatory) **Input type:** - String **Output type:** - String **Requires:** - Doesn't support conditional chaining. **Examples:** Truncate(5) Truncate(40)
UsagiMap	**Explanation:** Uses a Usagi map to map its input to a concept id. **Parameters:** 1 – The name of the Usagi file in the directory pointed to by the "usagi directory" clause in the ETL-Engine configuration file. (Mandatory) 2 – A default value to use when no match is found for the input in the Usagi file. By default this value is 0. (Optional)

	Input type: - Date - Float - Integer - String **Output type:** - Integer **Requires:** - Supports conditional chaining. **Examples:** UsagiMap("usagi-file.csv") UsagiMap("map.csv", 0)

VocabMap	**Explanation:**
	
	Uses the OMOP CDM vocabulary to map a vocabulary code to a standard concept id.
	
	The query used towards the database is:
	`select concept_id_1`
	`  from concept`
	`  inner join concept_relationship`
	`  on concept_id = concept_id_1`
	`  where relationship_id = 'Maps to'`
	`  and vocabulary_id = :vocabulary_id`
	`  and concept_code = :concept_code`
	
	Parameters:
	
	1 – The name of the vocabulary used in the mapping. (Mandatory)
	
	2 – A default value to use when there is no match for the input in the vocabulary. By default this value is `NULL`. (Optional)
	
	Input type:
	
	- String
	
	Output type:
	
	- Integer
	
	Requires:
	
	- Supports conditional chaining.
	
	Examples:
	
	VocabMap("SNOMED")
	
	VocabMap("ATC", 0)

VocabSourceId	**Explanation:**

	Uses the OMOP CDM vocabulary to map a vocabulary code to a source concept id.
	
	The query used towards the database is:
	`select concept_id_2`
	`  from concept`
	`  inner join concept_relationship`
	`  on concept_id = concept_id_1`
	`  where relationship_id = 'Maps to'`
	`  and vocabulary_id = :vocabulary_id`
	`  and concept_code = :concept_code`
	
	Parameters:
	
	1 – The name of the vocabulary used in the mapping. (Mandatory)
	
	2 – A default value to use when there is no match for the input in the vocabulary. By default this value is `NULL`. (Optional)
	
	Input type:
	
	- String
	
	Output type:
	
	- Integer
	
	Requires:
	
	- Doesn't support conditional chaining.
	
	Examples:
	
	VocabSourceId("SNOMED")
	
	VocabSourceId("ATC", 0)

YearFromDate	**Explanation:** YearFromDate expects a date as an input and will return an integer representing the year in the date. This is useful for the year_of_birth field in the person table. **Parameters:** No parameters accepted. **Input type:** - Date **Output type:** - Integer **Requires:** - Doesn't support conditional chaining. **Examples:**

	YearFromDate()

APPENDIX C – AGGREGATING RULES

Rule name	Parameters
Add	**Explanation:** Adds all values given to the command. If no values are provided, all inputs will be added. **Parameters:** *N* – The n^{th} addend of the addition. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Integer - Float - Date **Output type:** - Integer - Float - Date **Requires:** - Only one date can be given in the input. - A date cannot be added to a float. - If a float is in the input the return type will always be a float. - If a date is in the input, the return type will always be a date. **Examples:** Add() Add(1, 2) Add(:integer1, 0.6) Add(:date1, :integer2, 3600)
And	**Explanation:** And is a Boolean operator that ands together multiple Boolean values. 2 or more values can be specifically given to the command. If no values are provided, all inputs will be anded. **Parameters:** *N* – The n^{th} Boolean value to be anded. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Boolean **Output type:**

	- Boolean **Requires:** - At least two inputs are available. **Examples:** And() Add(True, False) Add(:BooleanValue1, :BooleanValue2)
AsDays	**Explanation:** AsDays is a helper function to convert the integer value of seconds into the corresponding number of days – i.e. divide it by 60 * 60 * 24. The AsDays command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of days must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to AsDays or a single integer parameter must be provided. **Examples:** AsDays() AsDays(3628800)
AsHours	**Explanation:** AsHours is a helper function to convert the integer value of seconds into the corresponding number of hours – i.e. divide it by 60 * 60. The AsHours command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of hours must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:**

	- Integer **Requires:** - Either a single value must be presented as an input to AsHours or a single integer parameter must be provided. **Examples:** AsHours() AsHours(10800)
AsMinutes	**Explanation:** AsMinutes is a helper function to convert the integer value of seconds into the corresponding number of minutes – i.e. divide it by 60. The AsMinutes command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of minutes must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to AsMinutes or a single integer parameter must be provided. **Examples:** AsMinutes() AsMinutes(240)
AsMonths	**Explanation:** AsMonths is a helper function to convert the integer value of seconds into the corresponding number of months – i.e. divide it by 60 * 60 * 24 * 30. The AsMonths command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of months must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:**

	- Integer **Requires:** - Either a single value must be presented as an input to AsMonths or a single integer parameter must be provided. **Examples:** AsMonths() AsMonths(38880000)
AsSeconds	**Explanation:** AsSeconds is a helper function to convert an integer value of seconds into the corresponding number of seconds. The integer value is in seconds, so the command only passes the value through, it merely exists to complete the set of commands: AsMinutes, AsHours, AsDays, AsMonths, and AsYears. The AsSeconds command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to AsSeconds or a single integer parameter must be provided. **Examples:** AsSeconds() AsSeconds(53)
AsYears	**Explanation:** AsYears is a helper function to convert the integer value of seconds into the corresponding number of years – i.e. divide it by 60 * 60 * 24* 365. The AsYears command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds for which the equivalent number of years must be found. If not present a single integer input must be present. (Optional) **Input type:** - Integer

	Output type: - Integer **Requires:** - Either a single value must be presented as an input to AsYears or a single integer parameter must be provided. **Examples:** AsYears() AsYears(94608000)
AutoGenId	**Explanation:** Creates an ID sequence. The sequence can either be a sequence for the field, meaning a sequence unique for the field it is on, or a named sequence, in which case it is unique for the name. Named sequences are global and can be used on all tables. And the can be stored between runs of the ETL-Engine using the table rule StoreSequence. **Parameters:** 1 – The sequence name if any. (Optional) **Input type:** - Does not accept input **Output type:** - Integer **Requires:** - Doesn't have any hard requirements. **Examples:** AutoGenId() AutoGenId("My Sequence")
Concat	**Explanation:** Concatenates any number of strings. The input strings must be references in the correct order in the parameters. Static strings can also be used. **Parameters:** *N* – The N^{th} component that should be part of the resulting string, this can be either a static string or a reference to a string field in :-notation. (Mandatory) **Input type:** - String **Output type:**

	- String **Requires:** - Concat works both with and without inputs. Without inputs it can only contain static strings. **Examples:** Concat(:field1, " – ", :field2) Concat("I work like SetValue") Concat("Field7 contains: ", :field7) Concat(:field1, :field2, :field3)
Days	**Explanation:** Days is a helper function to create the integer value for a specified number of days. The integer value is in seconds. The Days command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of days to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Days or a single integer parameter must be provided. **Examples:** Days() Days(42)
Divide	**Explanation:** Divides a specified value by all additional values given to the command. If no additional values are provided, the specified value will be divided by all available values. **Parameters:** 1 – The value specified as the dividend of the division. This can be either an input in :-notation or a static value. *N* – The n^{th} divisor that the dividend must be divided by. This can be either an input in :-notation or a static value. (Optional)

	Input type: - Float - Integer **Output type:** - Float **Requires:** - Doesn't have any hard requirements. **Examples:** Divide(:dividend) Divide(:dividend, :divisor1, :divisor2) Divide(:dividend, 10) Divide(:dividend, 3.14, :divisor2)
Equal	**Explanation:** Without any parameters this checks if all inputs are equal. If so, True is returned otherwise false. When run with parameters, the values, static or referenced must likewise be equal to produce a True value. **Parameters:** *N* – The N^{th} value to check for equality. This can be either a static value or a field reference in :-notation. (Optional) **Input type:** - Boolean - Date - Float - Integer - String **Output type:** - Boolean **Requires:** - All values must have the same type. **Examples:** Equal() Equal(:float1, :float2, :float3) Equal(42, :the_answer)
GreaterThan	**Explanation:**

	At least two parameters must be given, static or referenced in :-notation, to the rule. Any value preceding another must be larger than the following parameter for the rule to produce a True value. **Parameters:** *N* – The *N*th value to check for being larger than the following parameter. This can be either a static value or a field reference in :-notation. (Mandatory) **Input type:** - Date - Float - Integer - String **Output type:** - Boolean **Requires:** - All values must have the same type. - Two or more parameters must be given to the rule. **Examples:** GreaterThan(:float1, :float2, 0.1) GreaterThan(12, :int)
GreaterThanOrEqual	**Explanation:** At least two parameters must be given, static or referenced in :-notation, to the rule. Any value preceding another must be larger than or equal to the following parameter for the rule to produce a True value. **Parameters:** *N* – The *N*th value to check for being larger than or equal to the following parameter. This can be either a static value or a field reference in :-notation. (Mandatory) **Input type:** - Date - Float - Integer - String **Output type:** - Boolean **Requires:** - All values must have the same type. - Two or more parameters must be given to the rule.

	Examples: GreaterThanOrEqual(:float1, :float2, 0.1) GreaterThanOrEqual(12, :int)
Hours	**Explanation:** Hours is a helper function to create the integer value for a specified number of hours. The integer value is in seconds. The Hours command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of hours to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Hours or a single integer parameter must be provided. **Examples:** Hours() Hours(19)
LessThan	**Explanation:** At least two parameters must be given, static or referenced in :-notation, to the rule. Any value preceding another must be smaller than the following parameter for the rule to produce a True value. **Parameters:** *N* – The N^{th} value to check for being smaller than the following parameter. This can be either a static value or a field reference in :-notation. (Mandatory) **Input type:** - Date - Float - Integer - String **Output type:** - Boolean **Requires:**

	- All values must have the same type. - Two or more parameters must be given to the rule. **Examples:** LessThan(:float1, :float2, 0.1) LessThan(12, :int)
LessThanOrEqual	**Explanation:** At least two parameters must be given, static or referenced in :-notation, to the rule. Any value preceding another must be smaller than or equal to the following parameter for the rule to produce a True value. **Parameters:** *N* – The N^{th} value to check for being smaller than or equal to the following parameter. This can be either a static value or a field reference in :-notation. (Mandatory) **Input type:** - Date - Float - Integer - String **Output type:** - Boolean **Requires:** - All values must have the same type. - Two or more parameters must be given to the rule. **Examples:** LessThanOrEqual(:float1, :float2, 0.1) LessThanOrEqual(12, :int)
Max	**Explanation:** Selects the highest value of all values given to it. If no parameters are given, it selects between all inputs, if parameters are given it only choses between the parameters. **Parameters:** 1 – The first value to check. This can be either an input type in :-notation or a static value. (Optional) *N* – The n^{th} value to check. This can be either an input type in :-notation or a static value. (Optional) **Input type:** - Date

	- Float - Integer - String **Output type:** - Same type as the input type. **Requires:** - All parameter types and input types must be identical. **Examples:** Max() Max(:value1, :value2) Max(:value, 42) Max(:text1, :text2, "Zulu")
MergeDate	**Explanation:** Combines several integers or dates into a single date. Each integer must be marked with its role in the date. These roles are: - date: input type Date - time: input type Date - year: input type Integer - month: input type Integer - day: input type Integer - hour: input type Integer - minute: input type Integer - second: input type Integer **Parameters:** 1 – An array of two entries – the first entry must be the field to use in :-notation, the second must be the role of the input in the date returned. (Mandatory) 2 – An array of two entries – the first entry must be the field to use in :-notation, the second must be the role of the input in the date returned. (Mandatory) *3..6* – An array of two entries – the first entry must be the field to use in :-notation, the second must be the role of the input in the date returned. (Optional) **Input type:** - Date - Integer **Output type:** - Date **Requires:** - The role of the entries cannot be outside that of date, hour, year, month, day, hour, minute and second.

	Examples: MergeDate([:start_date, "date"], [:start_hour, "time"]) MergeDate([:year, "year"], [:month, "month"],[:day, "day"], [:hour, "hour"],[:minute, "minute"], [:second, "second"])
Min	**Explanation:** Selects the lowest value of all values given to it. If no parameters are given, it selects between all inputs, if parameters are given it only choses between the parameters. **Parameters:** 1 – The first value to check. This can be either an input type in :-notation or a static value. (Optional) *N* – The n^{th} value to check. This can be either an input type in :-notation or a static value. (Optional) **Input type:** - Date - Float - Integer - String **Output type:** - Same type as the input type. **Requires:** - All parameter types and input types must be identical. **Examples:** Min() Min(:value1, :value2) Min(:value, 42) Min(:text1, :text2, "Alpha")
Minutes	**Explanation:** Minutes is a helper function to create the integer value for a specified number of minutes. The integer value is in seconds. The Minutes command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of minutes to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer

	Output type: - Integer **Requires:** - Either a single value must be presented as an input to Minutes or a single integer parameter must be provided. **Examples:** Minutes() Minutes(240)
Months	**Explanation:** Months is a helper function to create the integer value for a specified number of months. The integer value is in seconds. The Months command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of months to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Months or a single integer parameter must be provided. - A single month is expected to be 30 days. Thus, 12 months is 360 days. **Examples:** Months() Months(12)
Multiply	**Explanation:** Multiplies all values given to the command. If no values are provided, all inputs will be multiplied. **Parameters:** 1 – The multiplier of the multiplication. This can be either an input in :-notation or a static value. (Optional) *N* – The n^{th} multiplicand that the multiplier must be multiplied by. This can be either an input in :-notation or a static value. (Optional) **Input type:**

	- Float - Integer **Output type:** - Float (If any float was given as input) - Integer (If no floats were given as input) **Requires:** - Doesn't have any hard requirements. **Examples:** Multiply() Multiply(:multiplier, :multiplicand1, :multiplicand2) Multiply(10, 10, 10) Multiply(4.2, :multiplicand)
NotEqual	**Explanation:** Without any parameters this checks if all inputs are different. If so, True is returned otherwise false. When run with parameters, the values, static or referenced must likewise be inequal to produce a True value. **Parameters:** *N* – The N^{th} value to check for inequality. This can be either a static value or a field reference in :-notation. (Optional) **Input type:** - Boolean - Date - Float - Integer - String **Output type:** - Boolean **Requires:** - All values must have the same type. **Examples:** NotEqual() NotEqual(:float1, :float2, :float3) NotEqual(17, :int)
Or	**Explanation:**

	Or is a Boolean operator that ors together multiple Boolean values. 2 or more values can be specifically given to the command. If no values are provided, all inputs will be ored. **Parameters:** *N* – The *n*th Boolean value to be ored. This can be either an input in :-notation or a static value. (Optional) **Input type:** - Boolean **Output type:** - Boolean **Requires:** - At least two inputs are available. **Examples:** Or() Or(True, False) Or(:BooleanValue1, :BooleanValue2)
Seconds	**Explanation:** Seconds is a helper function to create the integer value for a specified number of seconds. The integer value is in seconds, so the command only passes the value through, it merely exists to complete the set of commands: Minutes, Hours, Days, Months, and Years. The Seconds command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of seconds to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Seconds or a single integer parameter must be provided. **Examples:** Seconds() Seconds(53)

Selector	**Explanation:** Based on either a Boolean value or a comparison between two values a choice is made, either between one value and `NULL` or between two values. **Parameters in Boolean mode:** 1 – A boolean value referenced in :-notation. (Mandatory) 2 – The value chosen if the Boolean value given as the first parameter is True. This can be either a static value or a field reference in :-notation. (Mandatory) 3 - The value chosen if the Boolean value given as the first parameter is false. This can be either a static value or a field reference in :-notation. If unset the value will be `NULL`. (Optional) **Parameters in Comparison mode:** 1 – The first value of the comparison. This can be either a static value or a field reference in :-notation. (Mandatory) 2 – The comparison operator, this must be a static string value of one of the following: - < - The first value is less than the second. - = - The first value is equal the second. - > - The first value is greater than the second. - <= - The first value is less than or equal to the second. - <> - The first value is not equal to the second. - >= - The first value is greater than or equal to the second. (Mandatory) 3 – The second value of the comparison. This can be either a static value or a field reference in :-notation. (Mandatory) 4 – The value chosen if the comparison evaluates to true. This can be either a static value or a field reference in :-notation. (Mandatory) 5 - The value chosen if the comparison evaluates to false. This can be either a static value or a field reference in :-notation. If unset the value will be `NULL`. (Optional) **Input type:** - Boolean - Date - Float - Integer - String **Output type:** - The same as the second parameter's type for Boolean mode or the fourth parameter's type for comparison mode. **Requires:**

	- In comparison mode the first and the third parameter must have the same type. - If a third parameter in Boolean mode or a fifth parameter in comparison mode is present it must have the same type as the fourth parameters. - In comparison mode the second parameter must be one of: <, =, >, <=, <>, and >=. **Examples:** Selector(:int1, "=", :int2, "Equal", "Unequal") Selector(:str1, "<", "E", :str2) Selector(:date1, "<>", :date2, "Unequal dates", "Equal dates") Selector(:float1, ">", :float2, :str1) Selector(:Boolean, "True", "False")
SetValue	**Explanation:** Creates an output from a parameter value. **Parameters:** 1 – The value to set to the output. (Mandatory) **Input type:** - Does not accept input **Output type:** - The type of the value in the first parameter. **Requires:** - Doesn't have any hard requirements. **Examples:** SetValue(42) SetValue("My Value")
Subtract	**Explanation:** Subtracts a specified value by all additional values given to the command. If no additional values are provided, the specified value will be subtracted by all available values. **Parameters:** 1 – The value specified as the minuend of the subtraction. This can be either an input in :-notation or a static value. *N* – The n^{th} subtrahend that must be subtracted from the minuend. This can be either an input in :-notation or a static value. (Optional) **Input type:**

	- Integer - Float - Date **Output type:** - Integer - Float - Date **Requires:** - At most two dates can be given in the input. - A float cannot be subtracted from a date. - A date can neither be subtracted from a float nor an integer. - If a float is in the input the return type will always be a float. - If a single date is in the input, the return type will be a date. - If two dates are in the input, the return type will be an integer. **Examples:** Subtract(42) Subtract(:float, 42) Subtract(:date1, :date2) Subtract(:integer, 1000, :float)
Use	**Explanation:** When having multiple source fields mapped to a CDM field, or having created multiple local field variables in a calculation this rule selects the one that will go in to the CDM field. This command is special as it will always remove the input fields and only leave one field as an output of the rule. Even if it has an output field defined. **Parameters:** 1 – The field or local field variable in :-notation to commit to the CDM field. **Input type:** - Date - Float - Integer - String **Output type:** - The same as the input type. **Requires:** - Doesn't have any hard requirements. **Examples:** Use(:Result)

	Use(:location_id) Use(:Result) => :Result The line above will leave the :Result field in the rule chain, and it can be referenced as :Result.
Years	**Explanation:** Years is a helper function to create the integer value for a specified number of years. The integer value is in seconds. The Years command either accepts a single input or it accepts a static parameter. **Parameters:** 1 – The number of years to calculate the number of seconds for. If not present a single integer input must be present. (Optional) **Input type:** - Integer **Output type:** - Integer **Requires:** - Either a single value must be presented as an input to Years or a single integer parameter must be provided. - A single year is expected to be 365 days. **Examples:** Years() Years(2)

	Issue	Preprocessing solution	Solution in RiaH files	Decision
CPR				
t_person	Duplicate rows are present	Remove duplicate rows		Preprocessing
municipality_list (created	Municipality list is needed for Locations	Add a table with municipality numbers and names. Note: use comma separation for input to ETL engine and semicolon separation for input to WhiteRabbit		Preprocessing
dansk_ophold_periode	We need unique DK stays (dk_adresse_start ,dk_adresse_slut) for each person for Observation_period.	Create new table with the unique periods	Use normalize function to get unique periods	Instructions in RiaH files
dansk_ophold_periode	dk_adresse_slut is 2500-01-01 for present residence. It should be changed to data cutoff date for use in Observation_period	Change 2500-01-01 to 2024-03-15	Use SetValue and Selector functions to change 2500-01-01 to 15/3-2024.	Instructions in RiaH files
dansk_ophold_periode	dk_adresse_start can be earlier than data cutoff date. It should be changed to 1995-01-01 for use in Observation_period.	Change dates earlier than 1995-01-01 to 1995-01-01	Change dates earlier than 1995-01-01 to 1995-01-01	Instructions in RiaH files
dansk_ophold_periode	dk_adresse_slut (and then also dk_adresse_start) can be earlier than 1995-01-01.	Remove records with dk_adresse_slut <1995-01-01	Do not insert a record in observation_period if dk_adresse_slut <1995-01-01	Instructions in RiaH files
DAR				
t_dodsaarsag_1, t_dodsaarsag_2	There are duplicates of CPR +date, when the two tables are combined	Remove records with CPRs present in t_dodsaarsag_1 from t_dodsaarsag_2		preprocessing
c_dod1 and c_dod_tilgrundl_acme	the ICD10 code is not standard format. Punctuation mark is missing.	Create new columns with preprocessed diagnosis code (insert punctuation mark)	Insert punctuation mark	Instructions in RiaH files
CAR				
t_tumor: Create new ICDO3 column	The ICDO3 code needs to be constituted from c_morfo and c_topo	Create new variable with concatenated c_morfo and c_topo, preprocessed to correct ICDO3 code		Preprocessing
t_tumor: create new index column	The records in Measurement table should be linked to the corresponding records in Condition_occurrence. An index is needed for this.	Create id_index column in t_tumor		Preprocessing

	Issue	Preprocessing solution	Solution in RiaH files	Decision
LPR2				
All tables	There are corresponding psyk tables for all LPR2 tables. How can we map them in the most easy way?	Append the psyk tables to the corresponding LPR2 tables (t_psyk corresponds to t_bes)		Preprocessing
t_diag: c_diag	The ICD10 code is not standard format: Prefix 'D', no punctuation mark, uncommon suffix letter and digits.	Create columns with preprocessed diagnosis codes: Diag_std1 (punctuation mark inserted, suffix letters removed), Diag_std2 (also concatenated to 5 char)		Preprocessing
t_diag	There are duplicates of diagnosis codes if there is a 'tillægskode'. Also some codes can have another prefix than D.	Keep only records with prefix 'D' in diagnosis code	Use AssertNot to discard records with C_diagtype not in (A, B).	Preprocessing and RiaH instructions
t_adm	some inpatient visits are overlapping, which is not expected in OMOP (but there is no check for it). Overlaps are especially seen between somatic and psyk visits, but also within t_adm.	Overlaps between inpatient visits in t_adm can be solved by concatenating the overlapping visits. Use the first start date and last end date. Keep the visits in t_psyk_adm even if they overlap with t_adm		Not done. Can be done in next ETL'ing
LPR3				
Diagnoser: diagnosekode	the ICD10 code is not standard format	Create new columns with preprocessed diagnosis codes (std1, std2) as for LPR2		Preprocessing
Diagnoser	Diagnosekode can be A/B/+ types. We want only A and B.	Keep only records with prefix 'D' in diagnosis code	Use AssertNot to discard records with C_diagtype not in (A, B).	Preprocessing and RiaH instructions
DDV				
DDV_SSI	The table is in wide format	Transpose to long format: one row for each vaccination, one column for vaccinename, one column for vaccine_date. Remove empty rows.		Preprocessing
LSR				
Epikur	Replicates of the variables for ETL'ing are present. There are also -but fewer replicates in the full Epikur.	Remove duplicate rows or duplicates of the selected variables.		Not applied
lsr_calculations (derived table)	Quantity calculations needed	Add a column, quant1, with calculated quantity for a single pack (only for VNRs mapped to a Clinical Drug concept_id)	Calculate Quantity as quant1*apk (apk is number of packs)	Preprocessing and RiaH instructions