Human Brain Project SP8.2

Medical Informatics Platform (MIP)
MIP DATA FACTORY

Data Processing User Guide

Author (MIP Deployment and Support Team)

Version 02.30 (2020 07 07)

Document Control

Document Owner	HBP 8.2 MIP Deployment and Support Team
Document Type	MIP User Guide
Document Purpose	To provide a good enough description of the process to execute in order to, leveraging the MIP Data Factory set of tools, achieve the processing of hospital's data before the data is made available to the MIP as a MIP dataset.

Date of update	Updated by (author name)	Changes Made with Brief Description	Version #
2019 05 24	Iosif Spartalis Kostis Karozos	Created 1.0 version of the document	01.00
2019 08 29	Iosif Spartalis Kostis Karozos	Reorganized the Design of the Data Factory section. Explained in detail the mapping-tasks for a mock dataset.	01.10
2019 10 21	Iosif Spartalis Kostis Karozos	Updated Data Factory deployment and usage scripts	01.20
2019 12 11	Iosif Spartalis	Updated MRI pipeline execution	01.30
2019 12 12	Christian Dhondt	Documenting reformatting using the MIP Documentation Template	02.00
2019 12 13	Christian Dhondt	Document restructuring, revision (round 1)	02.00
2019 12 14	Christian Dhondt	Document restructuring, revision (round 2)	02.00
2019 12 15	Christian Dhondt	Document improvement (end to end)	02.00
2019 12 19	Christian Dhondt	All related to learning how to MIPMAP pushed in appendix as tutorial	02.00
2019 12 29	Christian Dhondt	Full revision / improvement before review with AUEB team	02.00
2020 01 15	Christian Dhondt Iosif Spartalis	Document improvement following full review with AUEB team member.	02.00
2020 02 19	Iosif Spartalis Kostis Karozos Laith Abu-Nawwas Jerome Chaptinel		
2020 02 24	Laith Abu-Nawwas	Correct conflicts and remove misleading paragraphs. In addition to document restructuring (page 3 and 7-15); I totally removed the old part "Types of Information and Data"	02.00

2020 03 04	Iosif Spartalis	Incorporated Laith proposals. Minor improvements and clarifications. Added the preprocessing design tool in appendix 2.	02.10
2020 05 07	Iosif Spartalis Kostis Karozos Laith Abu-Nawwas	Added some instructions in the Appendix 2. Made some small improvements and corrections.	02.20
2020 07 07	Kostis Karozos Laith Abu-Nawwas	Added Licenses for the Data Factory components	02.30

Acronyms

CHUV	Centre Hospitalier Universitaire Vaudois	AUEB	Athens University of Economics and Business
HBP	Human Brain Project	EEG	Electroencephalogram
MIP	Medical Informatics Platform	SME	Subject Matter Expert
EHR	Electronic Health Record	CDEs	Common Data Elements
JSON	JavaScript Object Notation; is open-standard file format or data interchange format	CSV	Comma-separated values file; is a delimited text file that uses a comma to separate values.
SQL	Structured Query Language	MRI	Magnetic Resonance Imaging
Db	Generic database file that stores data in a structured format	GUI	Graphical User Interface
ETL	Extract Transform Extract	QC	Quality Control
DICOM	Digital Imaging and Communications in Medicine	NIFTI	Neuroimaging Informatics Technology Initiative
DF	Data Factory	DC	Data Catalogue

Table of Contents

About this Document

Target Audience

This document is at the attention of members of the hospital MIP installation and setup project team (Technical Support and Data Manager) whose objective is to prepare data for ingestion to the MIP.

It provides them with the information needed to understand:

- The ETL (Extraction, Transform, Load) approach on which the data processing activities are based to achieve hospital's data ingestion to the MIP
- The role of the MIP Data Factory in the data processing activities to perform before the hospital data can be ingested into the MIP
- The major components of the MIP Data Factory that are available and can be used during the data processing activities

Document Overview

This document describes the MIP Data Factory architecture and for which purpose it was assembled. It also explains the overall data processing process and how the MIP Data Factory shall be used to achieve data ingestion to the MIP.

It is a step-by-step how to guide covering hospital data processing end to end to conform to MIP's standards before hospital data are added to the MIP. It provides links to additional documentation and detailed instructions described in GITHUB.

Using the Data Factory to process the hospital's data is a straightforward procedure. The only trivial thing is the configuration of the Data Factory EHR pipeline . This configuration is essential and must be performed by the hospital's data manager. In the appendix 2 of this document there is a tutorial for creating those configuration files (mapping tasks configuration files) with mock EHR data.

Concepts and Definitions

Electronic Health Records (EHR)

Health information and clinical records registered per each patient per visit in the hospital's database (Oracle, SQL or any database systems) and usually transferred in db or CSV format. EHR usually contain different levels of data; we might define them in this context as spaces, domain and sub-domain. For example; General space might include demographic, social status or patient's medical history as different data domains. On the other hand, EHR contain other data spaces related to the specific medical condition such as Dementia or Epilepsy where each space includes specific domain and sub-domain, such as medical assessments and tests, diagnoses, treatment and operations, etc.

Database Variables:

A variable or scalar is a storage address (identified by an index or address) paired with an associated symbolic name, which contains some known or unknown quantity of information referred to as a value. [Knuth, Donald (1997). The Art of Computer Programming. 1 (3rd ed.). Reading, Massachusetts: Addison-Wesley. p. 3-4.]

Data Models (Metadata)

Data Model (Metadata) describes the structure of database variables found in specific extract of a hospital's database; including descriptive metadata, structural metadata, administrative metadata, reference metadata and statistical metadata. [1] Zeng, Marcia (2004). "Metadata Types and Functions". NISO. Archived from the original on 7 October 2016. Retrieved 5 October 2016.

Data element

In metadata, the term data element is an atomic unit of data that has precise meaning or precise semantics. [Beynon-Davies P. (2004). Database Systems 3rd Edition. Palgrave, Basingstoke, UK]

Medical Conditions

Diseases are often known to be medical conditions that are associated with specific symptoms and signs. ["Disease" at Dorland's Medical Dictionary]

Common Data Elements (CDEs)

A set of standard variables defined by clinical experts and data scientists, which would be used by researchers to perform analysis on specific medical conditions at the federation level. In the MIP context, we use the term CDE to refer to the standardized federated data models only.

The MIP and the MIP Data Factory

The Medical Informatics Platform (MIP) is a system that integrates Electronic Health Records (EHR) from various hospitals and offers the capability to run medical experiments and statistical data analysis on specific medical conditions.

MIP consists of three major components:

- **MIP front-end:** User interface used by researchers to explore available data and perform medical experiments or statistical data analysis based on predefined algorithms. MIP front-end can be deployed on different scenarios depends on the machine and the network connectivity as below:
    - Local MIP: When MIP front-end installed in a hospital internal server or local machine, and reads only a specific hospital's data through an internal network or offline.
    - Federated MIP: using MIP FEDERATED, which is allowing a secure access to locally stored anonymized data from various federated hospitals.
- **MIP Data Catalog**: Web-based application that is used to explore and manage MIP Data Models and CDEs pre-defined by the federation's community or the use case group.
- **MIP Data Factory**: Toolbox used by hospital's data managers to perform various data preparation and processes (ETL) to make the data compatible with the MIP system.

The MIP Data Catalogue (One Source of the Truth)

All the existing Medical Conditions, Federations and Hospitals and their associated Data Models (Metadata) are available within the MIP Data Catalog, which is the UNIQUE SOURCE of the TRUTH for the MIP and the MIP DATA FACTORY.

The MIP Data Catalog is a web-based application that can be accessed at the following address*. It provides access to MEDICAL CONDITIONS and their data model, FEDERATIONS and HOSPITALS and their model (federated or standalone).

These models are maintained in the MIP Data Catalog as models defined by a:

- Federation of hospitals sharing the same Common Data Elements.
- Stand-Alone hospitals when dealing with non-federated and specific local Medical Conditions.

*The MIP Data Catalog web application is available HERE. It is recommended to use any other browser except Firefox.

More information on the MIP Data Catalog and on how to use it, look at the documentation in github (https://github.com/HBPMedical/DataCatalogue).

Supported Medical Conditions (as of December 2019)

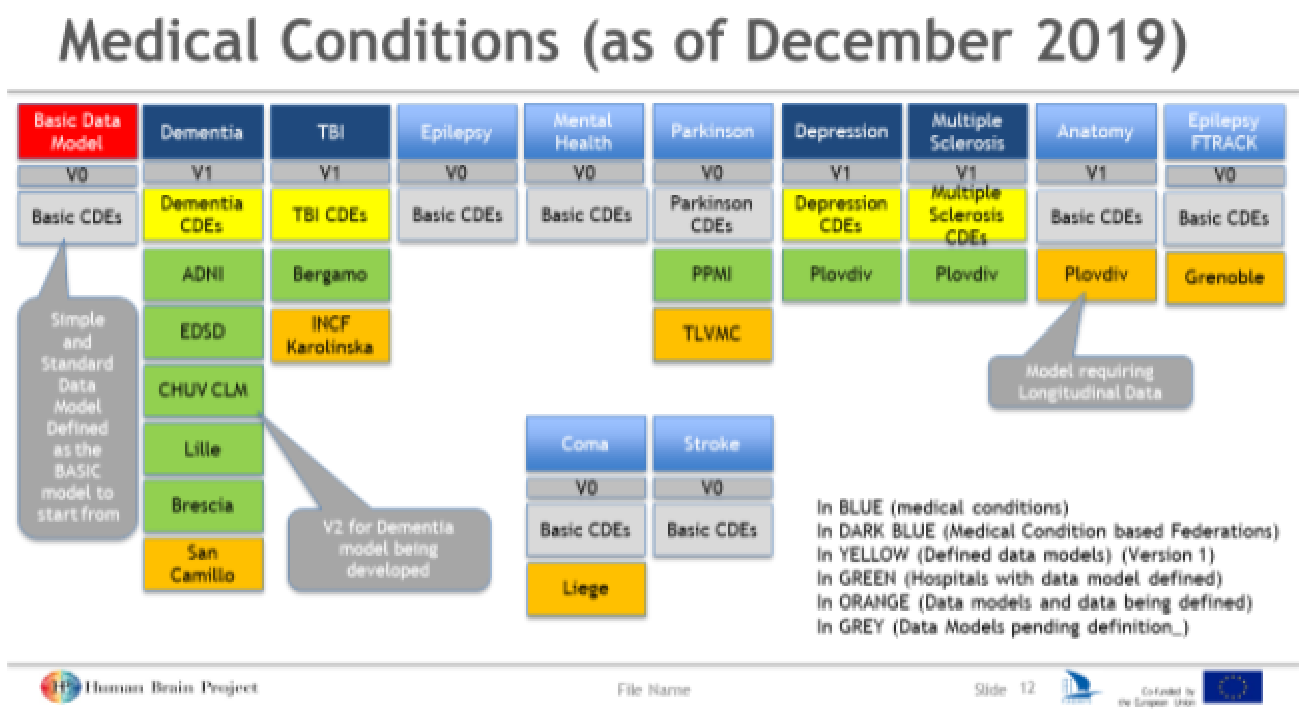
As of December 2019, the defined medical conditions on the MIP are the following:

Medical Conditions

- Federations
  - Dementia
  - Trauma Brain Injury (TBI)
  - Epilepsy
  - Mental Health
- Stand Alone Installations
  - Parkinson
  - Depression
  - Multiple Sclerosis
  - Anatomy
  - Coma
  - Epilepsy FTRACK

The graphic below gives an initial view of all existing Medical Conditions, federations and hospitals with the corresponding versions of Data Models (Metadata) in use as of the published date of this document.

For **updated info about the available defined medical conditions** please refer to the Data Catalog Web application.


Medical Conditions (as of December 2019)
Basic Data Model
V0
Basic CDEs
Simple and Standard Data Model Defined as the BASIC model to start from
Dementia
V1
Dementia CDEs
ADNI
EDSD
CHUV CLM
Lille
Brescia
San Camillo
TBI
V1
TBI CDEs
Bergamo
INCF Karolinska
V2 for Dementia model being developed
Epilepsy
V0
Basic CDEs
Mental Health
V0
Basic CDEs
Parkinson
V0
Parkinson CDEs
PPMI
TLVMC
Depression
V1
Depression CDEs
Plovdiv
Multiple Sclerosis
V1
Multiple Sclerosis CDEs
Plovdiv
Anatomy
V1
Basic CDEs
Plovdiv
Model requiring Longitudinal Data
Epilepsy FTRACK
V0
Basic CDEs
Grenoble
Coma
V0
Basic CDEs
Liege
Stroke
V0
Basic CDEs
In BLUE (medical conditions)
In DARK BLUE (Medical Condition based Federations)
In YELLOW (Defined data models) (Version 1)
In GREEN (Hospitals with data model defined)
In ORANGE (Data models and data being defined)
In GREY (Data Models pending definition_)
Human Brain Project
File Name
Slide 12
Co-funded by the European Union

MIP Data Factory

MIP Data Factory is a set of tools and components designed to process hospital's electronic Health Records (EHR) and MRI brain scans and store them into an i2b2 database. These records are usually extracted by hospital's data managers from the hospital's database system. As a data protection measurement, those data must be pseudo-anonymized by the hospital before processing them on MIP Data Factory.

Data Factory components are deployed in the Hospital's server where the actual data processing is performed through two major pipelines (EHR and Imaging). Not all of the Data Factory tools are supposed to be deployed in the hospital's server (see section for Data Factory components and tools).

The below summary explain the main operations MIP Data Factory can perform:

- Quality Control on EHR and MRI data
- Extract Brain NeuroMorphometric Features from MRI files.
- Transform the existing hospital Electronic Health Records (EHR) and the Brain NeuroMorphometric Features (from above)
- Store brain features and EHR data in an i2b2 database
- Harmonize hospital data so as to conform to a specific set of CDEs for the federation of a Medical Condition
- Anonymize hospital data in the process to make it available on the federated MIP.

Federations and Federation Leaders

Each Data Model (Metadata) associated with a Medical Condition is supported by a federation of hospitals under the responsibility of a "Federation Leader". The role of the "Federation Leader" is to ensure the Medical Condition Data Model (Metadata) evolves over time in a coordinated manner version after version.

As of today (December 2019), there are four Medical Conditions for which a Federation of Hospitals is being assembled. The federation leaders are listed here after.

Medical Condition	Leader	Hospital
Dementia	Prof. Jean Francois Demonet Jean-Francois.Demonet@chuv.ch	Center Leenardt for Memory, CHUV, Lausanne
Trauma Brain Injury (TBI)	Dottore Guido Bertolini guido.bertolini@marionegri.it	Hospedale Mario Negri, Bergamo, Italy
Epilepsy	Prof. Philippe Ryvlin Philippe.Ryvlin@chuv.ch	Neurosciences Department, CHUV, Lausanne
Mental Health	Prof. Pegah Sarkheil psarkheil@ukaachen.de	University Hospital RWTH, Aachen

When a hospital targets to join a FEDERATION of hospitals for a specific MEDICAL CONDITION, contacting the Medical Condition / Federation leader ahead of MIP installation and setup is strongly recommended.

Models defined for a MEDICAL CONDITION supported by a FEDERATION of Hospitals are made of Common Data Elements (CDEs) that are shared by all hospital's members of the federation. Thus, allowing federated data analysis to run on a shared set of variables, the CDEs.

Data Processing using the MIP Data Factory

Data originating from diverse hospital data sources are highly heterogeneous in nature, so, in order to be imported into the MIP, hospitals' data must be pre-processed to conform to MIP's way of expressing and storing clinical information.

Extract, Transform and Load (ETL) Approach

To achieve this purpose, the MIP Data Factory is defined to allow hospitals equipped with the MIP to perform the **TRANSFORMATION** part of the ETL (Extraction, Transformation, Load) approach on which the processing of hospital's data is defined:

- EXTRACT consists on selecting and extracting data from the hospital's systems
- **TRANSFORM** processes extracted data and converts them into the expected schema and format
- LOAD makes the transformed data available to the target system or application

To that effect, the MIP Data Factory:

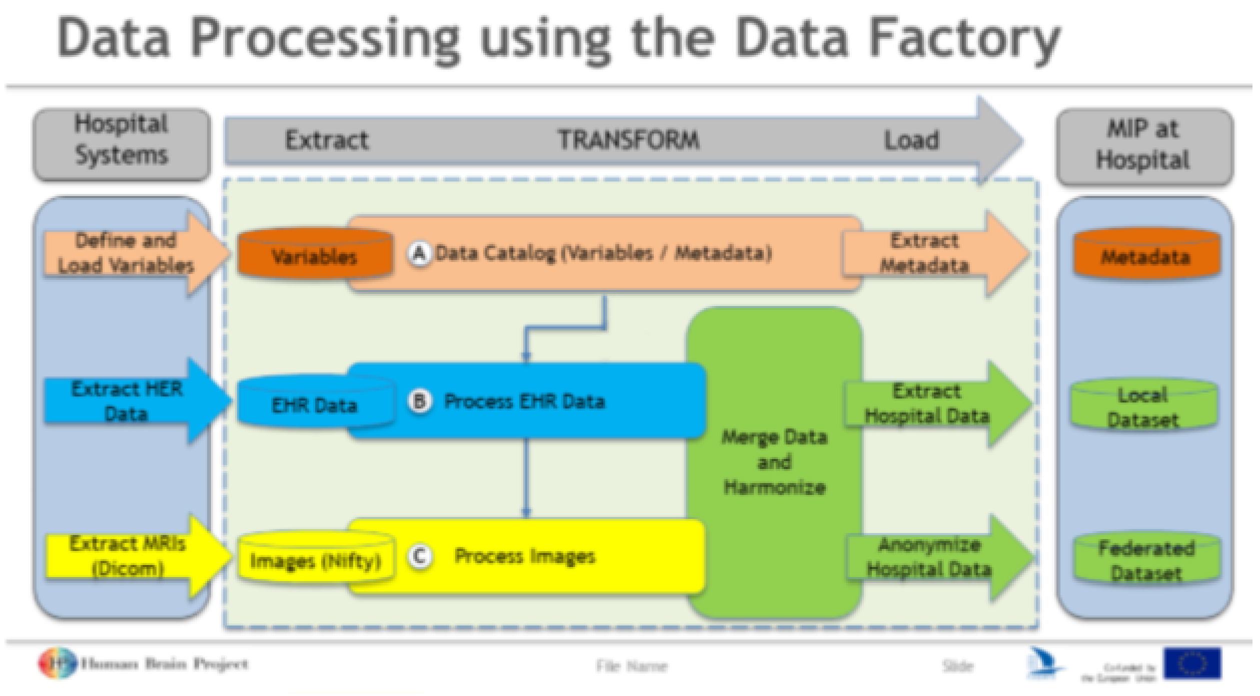
- Is built around a standard data schema (i2b2 data warehouse star schema). See appendix 1 for more details.
- Provides a set of tools and procedures for processing hospital's data in order the data to comply with the definitions of the minimum set of variables and be formatted to they be made available to the MIP in the expected format
- Allows hospitals and federations of hospitals to define the Data Model (Metadata) they want to use for data analysis, thus making the MIP highly configurable and adaptable to a specific set of data with specific constraints and values
- Is designed to collect over time and accumulate all hospital's data for one or more than one medical condition, thus providing the hospital with a way to safeguard their data at the use and scope of the MIP as the hospital progresses over time.

Threads of Work in Data Processing and pipelines

Three major threads of work are to consider when looking at the way the MIP Data Factory is designed:

- (A) – The "Variables" thread of work whose objective is to create the Data Model (Metadata) file that will be used by the MIP when performing analysis
- (B) – The EHR (Electronic Health Record) thread of work whose objective is to transform patients and clinical information according to the target Data Model (Metadata), variables definitions and MIP datasets format (EHR pipeline)
- (C) – The MRI (Magnetic Resonance Imaging) images thread of work whose objective is to transform patient scans images (NIFTI format) according to the target Data Model (Metadata), variables definitions and MIP datasets format (Imaging pipeline)

The following figure provides a high-level view of the MIP Data Factory processing activities in context of the ETL (Extract, Transform, Load) approach it is based upon.



Each of these threads of work:

- starts with an extraction step which consists in selecting and extracting data from hospital's systems and providing the extracted data is a predefined format so that it can be processed by the MIP Data Factory.
- ends with the creation of JSON (Metadata) and CSV files (Local dataset and Federated dataset) to copy to the MIP Local node and/or the MIP Federated Node to be accessed by researchers using the MIP.

Data Processing Critical Success Factors

Crucial to the success of the hospital's data processing activities, the following important critical success factors must be considered before even starting the process:

- Identification of variables and definition of the Data Model (Metadata) corresponding to hospital's data (EHR and Images processing) that will be made available to the MIP.
- Definition of period in scope and selection of patients and EHR data to extract before entering data transformation (patient information and clinical data collected during patient's visits for the period in scope).
- Selection of images of brain scans for every patient and every scan in scope and formatting of DICOM files in a NIFTI format before entering data transformation.

Components of MIP Data Factory

The main components of the Data Factory are the following:

Component	Functionality	License
MIPMapReduce	Schema mapping - Data Transformation	GNU General Public License v3.0
Ashburner's SPM12 package	MRI Brain's neuromorphometric feature extraction	GNU General Public License v3.0
LREN pipeline scripts	Pre-processing pipeline for SPM12	GNU Affero General Public License
MRI parallel neuromorphometric pipeline	Wrapper for SPM12	Apache License 2.0
NMM pipeline - NIFTI organiser?	Organizes MRI data so that it can be pre-processed properly	Apache License 2.0
NMM pipeline - Output merge	Merges individual outputs in one CSV file	Apache License 2.0
PostgreSQL v9.6	Data storage - i2b2 databases	PostgreSQL License
Anonymization module	EHR anonymization	GNU General Public License v3.0

Those components are deployed on Hospital's Server along with Local MIP. The execution of the DataFactory pipelines is performed through the terminal by a **python wrapper** script with CLI (see wrapper's github repo for DataFactory deployment and usage).

For Quality Control of the incoming data (EHR and MRIs) there are two optional tools that can be deployed in the hospital's server along with DataFactory's main components:

Component	Functionality	License
Data Quality Control tool	Statistical reports and quality control for EHR and MRIs (DICOM)	Apache License 2.0
LORIS for MIP	For DICOM quality control and selection	GNU General Public License v3.0

The *Data Quality Control* tool can be used as a stand alone application and has a simple GUI.

SPM12 is a specialized third party software developed by Ashburner's team in UCL.

Supporting Tools for the MIP Data Factory

Below are the supporting tools for MIP Data Factory:

Component	Functionality	License
MIPMap	Designing the mapping tasks	GNU General Public License v3.0
EHR-Mapping-Task design template	Configuring the Data Factory pipeline	GNU General Public License v3.0

The main supporting tool for Data Factory is *MIPMap* which is used for designing the mapping tasks that are going to be used in the Schema mapping and Data Transformation step. It has a very comprehensive Graphical User Interface and it can be deployed in any machine (it is not necessary to be installed in the hospital's server which probably will not have a graphical environment). The resulting mapping tasks configuration files are uploaded into the hospital server in specific DataFactory's configuration folders.

Other tools or sources of data are in the process of being added to the mix so as to increase the MIP Data Factory versatility and capabilities. Such as:

- REDCAP (see LINK) (for Epilepsy and Mental Health related data capture)
- Seizure Tracker System (Grenoble) (for Epilepsy related data capture)

Other third party Tools for the MIP Data Factory

dcm2niix DICOM to NIFTI format open source converter.

Data Processing Activities

The data processing process consists of 8 steps as described here after. It starts with extraction of data from hospital's systems (step 0) and leads to the creation of JSON and CSV files formatted so that they can be used by the MIP (steps 5, 6 and 7).

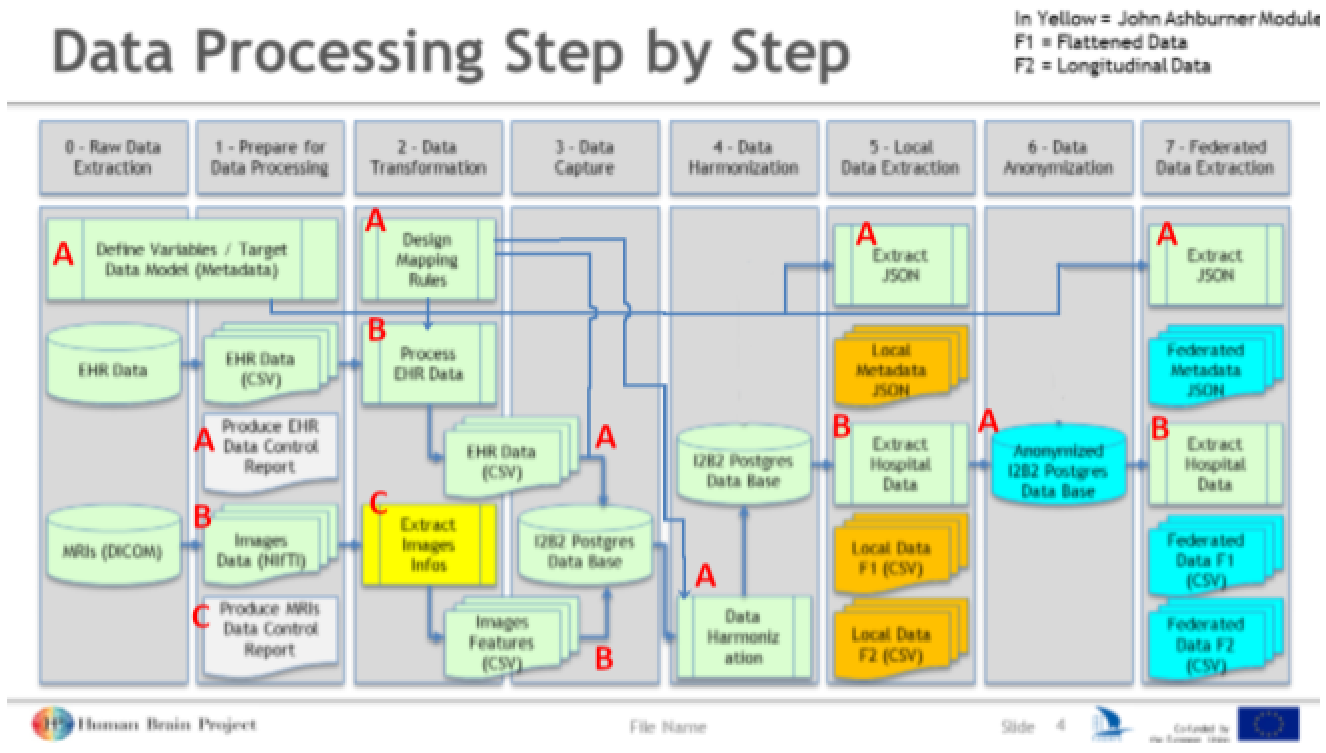


Figure 4

Tools that are used in each Data Processing Step							
0	1	2	3	4	5	6	7
A. Data Catalogue	A. DataQCtool EHR B. dcm2niix C. DataQCtool DICOM LORIS-for-MIP (optional for DICOM QC)	A. MipMap, Mapping task design template B. MipMapReduced using python wrapper (ingest ehr preprocess) C. SPM12 by using python wrapper (mri)	A. MipMapReduced using python wrapper (ingest ehr capture) B. MipMapReduced using python wrapper (ingest imaging)	A. MipMapReduced using python wrapper (ingest ehr harmonize)	A. Data Catalogue B. python wrapper (export)	A. Anonymization module using python wrapper (anonymize db)	A. Data catalogue B. Anonymization module using python wrapper (anonymize db)

These files to become visible to the MIP end users, must then be copied to the corresponding "medical condition" folder on the MIP local node and, when the hospital is part of a federation (federated hospital), to the corresponding "medical condition" folder on the MIP federated node.

Hospital's Data Format Requirements

Before starting to process data make sure the files that will be processed using the MIP Data Factory set of tools are defined and formatted correctly.

Variables and Target Data Model (Metadata)

Hospitals initiating the work related to data processing MUST decide on which Data Model (Metadata) to adopt in defining / processing their own data (EHR and MRIs).

Several options are to be considered before deciding:

- When targeting to join a federation (federated installation) for a medical condition
  - Ask the federation leader to provide the defined federated CDE as an xls or csv file to start from.
  - Select from the hospital's records the variables in common with the shared CDE file.
  - Assure the alignment between the enumerations defined in the federation CDE and the data extracted from the hospital's records.
- When targeting to use the MIP only locally (stand-alone installation), it can be done either ways below:
  - Define hospital own Data Model (Metadata) from scratch based on the data catalog metadata template (metadata.xls).
  - Or, Start from an existing Data Model (Metadata) and adapt to hospital's needs

The files needed are the following:

- *hospital_metadata.csv* - file with metadata about the hospital and Medical Condition (pathology) variables (created by the hospital data manager - see thread A)
- *cde_metadata.csv* - file with Medical Condition (pathology) metadata (preexisting when joining a federation or starting from an existing medical condition data model). Can be extracted from the Data Catalog or be provided by the federation leader of the Data Manager in charge of the Medical Condition Data Model.

Step 0 – Raw Data Extraction

Data Model (Metadata) CSV File format

The metadata CSV file, *hospital_metadata.csv*, contains metadata information about the hospital variables. It is a good practice to produce this file from the Excel metadata file which has been already filled in by the hospital for the Data Catalogue.

The templates to use for creating the metadata files can be found here:

https://github.com/aueb-wim/ehr-mapping-design-template/tree/master/source

The columns that need to have non-empty value, since they are completely necessary for the mapping task, are:

- name
- code
- type
- concept_path

The column **description** is not mandatory to be filled in. The columns **mapFunction** and **mapCDE** although they are not used directly in the **capture step** mapping task, contain information that is used in the **harmonization step**.

Once the variables and CDEs for the targeted Data Model (Metadata) have been defined (metadata.csv), the process of preparing hospital data for ingestion to the MIP can be started as per the following major steps:

Hospital's Data CSV Files Format (EHR)

Before entering the MIP Data Factory, hospitals data must be extracted from hospitals systems in CSV format for EHR data and NIFTI format for Brain Scans (MRIs in DICOM format transformed in NIFTI files).

The files needed are the following:

- EHR data in CSV files (1 or more)
    - Patients CSV (all the patients and patient ID codes in one file)
    - Visits CSV (all the visits and visit ID codes in one file for all the patients)
    - Examinations Data (one or several files)
    - Any other type of data needed
- Imaging data (MRIs) in a NIFTI format

The hospital CSV files are extracted from the hospital's EHR system and follow the hospital's local schema. In most cases, a hospital EHR system has a Patient and a Visit entity for storing a patient's demographic information and visit data.

Data in CSV files MUST contain:

- a field with a unique ID for each PATIENT
- a field with a unique ID for each VISIT
- a "Visit Start Date" field for each VISIT

Furthermore, hospital data CSV files will probably contain various examination results that a patient had during one or more hospital visits. This information must be linked with the Visit entities.

In terms of CSV files structure, the following basic conditions must be met:

- Header (variable) names
    - must not contain special characters like space, parentheses, hyphen, etc.
    - Use only underscore ("_") for word separation.
- Header (variables) names
    - must not start with any number character.
- The first column(s) must contain the Primary Key of the CSV
    - like patient id, visit id, etc.
- The delimiter must be the comma character (",").
- The encoding must be ASCII.

Hospital's MRI files

The specifications and the minimum requirements for the MRI files can be found in the following github link (https://hbpmedical.github.io/deployment/data/). In addition, each batch of MRIs must be accompanied by a csv file named **mri_visits.csv**. This file must have the headers PATIENT_ID, VISIT_ID, VISIT_DATE (the header names might as well be a bit different; the point is to have the information which visit was done by which patient and when) and be filled with this MRI information. The column VISIT_DATE must have the dd/mm/yyyy date format.

Step 1 - Quality Control and DICOM to NIFTI transformation

Once data have been given as input to the Data Factory, one can use the DataQualityControl tool to generate statistical reports for the EHR data (CSV files) and the brain scans (DICOM files) so as to evaluate their quality, detect possible errors and outliers so as to decide whether to continue in the Data Factory process with the existing set of extracted raw data or request for a new improved version. Then we can proceed with the transformation of MRI files from DICOM to NIFTI format with dcm2niix tool. The DICOM QC can be enhanced with the use of the optional LORIS-for-MIP feature, where an MRI specialist can visually inspect each MRI through LORIS web interface and exclude MRIs from being imported into the DataFactory's imaging pipeline (see the short demo video).

Step 2 - Mapping Tasks design - Data Processing

2A – Designing EHR data Mapping Rules (Tasks) using MIPMap

The output of this step are the configuration files for the preprocessing (see 2B), capture (see 3A) and harmonization step (see 4). This step is not performed in the hospital server but in a local machine with a desktop graphical interface. MIPMap is the main tool for designing the mapping tasks for the capture and the harmonization step. For making the design but also the testing of those mapping tasks easier in a local PC, a set up with a folder structure and execution scripts has been created (MIPMapReduced and PostgreSQL are included through docker containers for the testing of those config files). Below is the GITHUB repo link and more information can be found in the README file:

https://github.com/aueb-wim/ehr-mapping-design-template

A tutorial for this particular step can be found in the Appendix 2 of this documentation.

2B – Preprocessing EHR files

From the EHR side, the original CSV files run through a pre-processing stage with the objective to create auxiliary files that will be used in the capture step. Those files contain:

- the required information about the patient's unique ID codes(patientmapping.csv)
- the patients visit unique ID codes(encountermapping.csv)
- csv files from the original EHR csv files with unpivoted columns

The preprocessing step requires certain configuration files of the step 2A, for further details about those files please see the tutorial in the Appendix 2.

2C – Extract Imaging Features from MRIs - OPTIONAL

Note that not all Data Models (Metadata) defined for a specific Medical Condition require data extracted from Brain Scans (or MRIs). This step can therefore is skipped when there are no images to extract information from. For technical details about the input folder structure see the documentation in the github repos (https://github.com/HBPMedical/nmm-pipeline-nifti-organizer.git, https://github.com/HBPMedical/LORIS-for-MIP.git for LORIS-for-MIP option).

When entering the Data Factory pipeline, brain scans (DICOM files transformed in NIFTI file by using dcm2niix or the LORIS-for-MIP component) are processed by John Ashburner's tool which generates their brain neuromorphometric features. These features have numerical values and are stored in an output file "**volumes.csv"** in a subfolder for this specific MRI batch inside '/data/DataFactory/imaging' folder. The **"mri_visits.csv"** file which contains the MRI visit dates **must** be placed in the same batch subfolder.

Step 3 - Data Capture

3A – EHR Data Capture

The auxiliary files produced from the original EHR records, are imported to the I2B2 database with the use of MIPMapReduced by using the configuration files of the capture mapping task (step 2A). This database is referred to as "capture DB" since it captures hospital data values as they are.

3B – MRIs Brain Features Capture

The files volumes.csv and mri_visits.csv are being imported into the I2B2 "capture DB" database with the MIPMapReduced and thus the data from the imaging pipeline are linked with the data from the EHR pipeline.

Step 4 - Data Harmonization (skipped for non-Federated)

Afterwards, data is again processed by MIPMap which applies the harmonization rules (defined by clinicians) to conform data to MIP's schema. A new I2B2 database is populated with all values harmonized. This I2B2 database is called "harmonization DB".

Step 5 - Local Data Flattening

Data import into the MIP is done via the use of JSON files (CDEs and variable's definition), and CSV files (hospital's data).

In the first step and before importing the harmonized hospital's data into the MIP, the MIP must be set up with the right Data Model (Metadata) that corresponds to the set of variables and CDEs defined for the medical condition and the hospital. This metadata file (JSON format) is extracted from the Data Catalogue (see data catalog user guide) and then copied into the corresponding "medical condition" folder.

In a second step, the hospital's data (Information stored in the relational harmonized DB) is exported in a flat CSV file and then copied into the corresponding Medical Condition folder on the MIP Local node. For more details on the export process and strategies refer to https://github.com/HBPMedical/ehr-datafactory-template#export-flat-csv

Step 6- Anonymization (Federated Hospital Data) - OPTIONAL

When a hospital joins a federation of hospitals, the hospital's data MUST be anonymized before these data are moved to the MIP Federation node. To that effect an anonymized replica of the harmonized I2B2 database is created, sent and restored on the hospital's Federation node.

More detailed information and instructions can be found here:

https://github.com/aueb-wim/anonymization-4-federation

Step 7 - Federated Data Flattening - OPTIONAL

As for the installation of hospital's data and corresponding Data Model (Metadata) on the MIP Local node, a CSV File with ANONYMIZED data is exported from the I2B2 database and imported in the MIP Federation node along with its corresponding metadata file (extracted from the Data Catalog). These files are copied to the corresponding Medical Condition folder.

Setting Up the MIP Data Factory

Data Factory User Setup

Please refer to the repo's README file:
https://github.com/HBPMedical/ehr-datafactory-template

Data Factory Installation using Ansible

Please refer to the repo's README file:

https://github.com/HBPMedical/ansible-datafactory

Appendix 1 MIP I2B2 Database Description

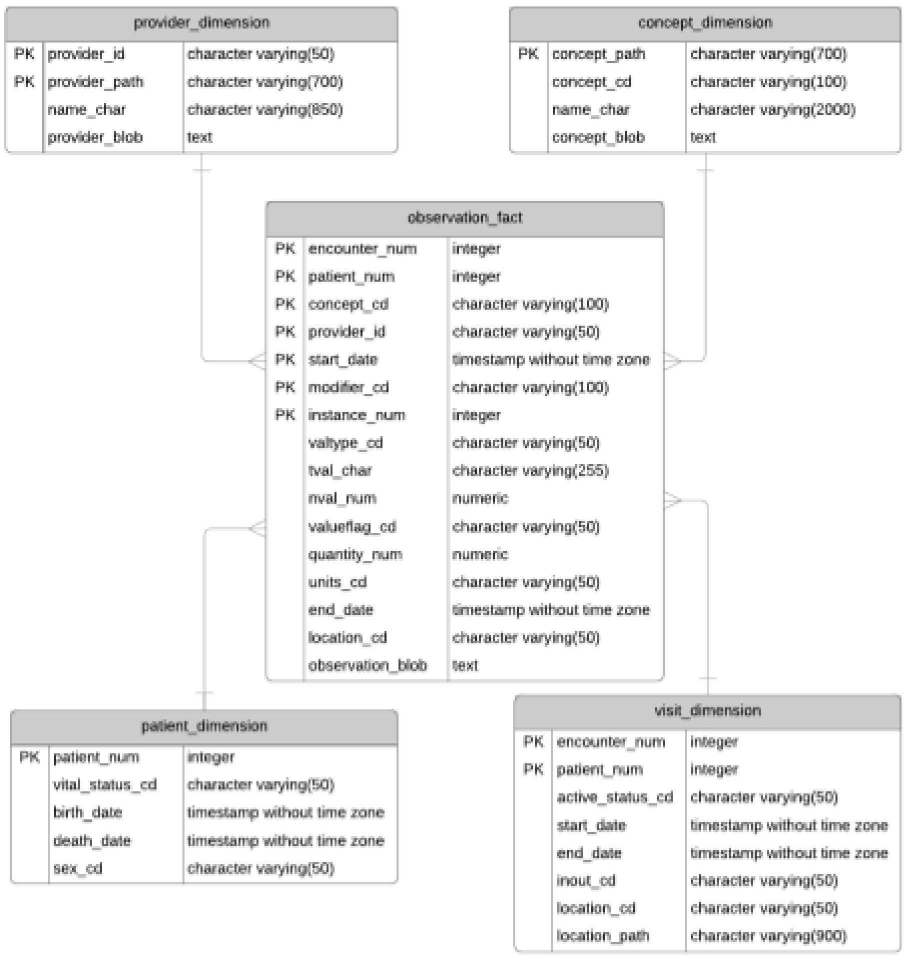
In order to process the hospital's data starting from the information of the hospital local CSV files we need to load the data into a relational database according to the I2B2 schema.

The I2B2 schema is basically a medical data warehouse star schema designed to store data from clinical trials, medical records and laboratory systems, along with many other types of clinical data from heterogeneous sources.

The main tables the I2B2 schema consists of are:

- concept_dimension
- provider_dimension
- observation_fact
- patient_dimension
- visit_dimension

Also, there are two mapping tables, **`encounter_mapping`** and **`patient_mapping`**, which map visit (encounter) and patient id to unique sequential integers.



provider_dimension		
PK	provider_id	character varying(50)
PK	provider_path	character varying(700)
	name_char	character varying(850)
	provider_blob	text

concept_dimension		
PK	concept_path	character varying(700)
	concept_cd	character varying(100)
	name_char	character varying(2000)
	concept_blob	text

observation_fact		
PK	encounter_num	integer
PK	patient_num	integer
PK	concept_cd	character varying(100)
PK	provider_id	character varying(50)
PK	start_date	timestamp without time zone
PK	modifier_cd	character varying(100)
PK	instance_num	integer
	valtype_cd	character varying(50)
	tval_char	character varying(255)
	nval_num	numeric
	valueflag_cd	character varying(50)
	quantity_num	numeric
	units_cd	character varying(50)
	end_date	timestamp without time zone
	location_cd	character varying(50)
	observation_blob	text

patient_dimension		
PK	patient_num	integer
	vital_status_cd	character varying(50)
	birth_date	timestamp without time zone
	death_date	timestamp without time zone
	sex_cd	character varying(50)

visit_dimension		
PK	encounter_num	integer
PK	patient_num	integer
	active_status_cd	character varying(50)
	start_date	timestamp without time zone
	end_date	timestamp without time zone
	inout_cd	character varying(50)
	location_cd	character varying(50)
	location_path	character varying(900)

Figure 5: I2b2 star schema

In the rest of the section we present how the I2B2 tables are used in the context of the MIP Data Factory. A few columns of the original I2B2 model are missing from our descriptions since they are not actually used in the process.

Concept Dimension

The **`concept_dimension`** table is the one containing the information about every type of observation stored in the **`observation_fact`**.

These observations might be CDEs or local hospital variables. For an observation to be inserted in the **`observation_fact`**, the **`concept_dimension`** table must contain its corresponding definition.

For example, for storing the score of the Montreal Cognitive Assessment (MOCA) of a patient, there must be a tuple in the `concept_dimension` defining MoCA's:

- `concept_path` ('/root/neuropsychology/montrealcognitiveassessment'),
- `concept_cd` ('montrealcognitiveassessment'),
- `name_char` ('MoCA Total')
- `import_date`. MoCA has also a description which is stored in the `concept_blob` column

Since not all variables come with a description, this column technically is not mandatory to be filled in for every tuple.

concept_dimension			
Fields used in Data Factory		Values	Mandatory (Y/N)
PK	concept_path	The hospital local or CDE variable concept path	Y
	concept_cd	The hospital local or CDE variable code	Y
	name_char	The hospital local or CDE variable name	Y
	concept_blob	The hospital local or CDE variable description	N
	import_date	Date data were imported	Y

Provider Dimension

provider_dimension		
Fields used in Data Factory	Values	Mandatory (Y/N)

PK	provider_id	Short name of the hospital with no space characters (for example “Niguarda” or “Niguarda_hospital”). Originally, in I2B2, used for the name of the person importing the data.	Y
PK	provider_path	“/Provider/Hospital/<provider_id>”	Y
	name_char	Full name of the hospital	N
	import_date	Date data were imported	Y

Patient Dimension

The patients are stored in the patient_dimension table. Originally, in I2B2, there are a lot of demographic columns portraying the patient (e.g. language, race, marital_status, religion). Due to privacy related issues the values for most of them are not available in the MIP.

patient_dimension			
Fields used in Data Factory		Values	Mandatory (Y/N)
PK	patient_num	A sequential integer mapped to the original patient id	Y
	birth_date	The patient’s birth date. Usually not available due to anonymization	N
	sex_cd	The patient’s gender (“M” or “F”)	Y
	import_date	Date data were imported	Y

Visit Dimension

The visit itself has its own dimension table. We expect a visit to have many observations.

visit_dimension			
Fields used in Data Factory		Values	Mandatory (Y/N)
PK	encounter_num	A sequential integer mapped to the original visit id	Y
PK	patient_num	A sequential integer mapped to the original patient id	Y
	start_date	Date of the start of the visit	Y

	end_date	Date of the end of the visit	N
	location_cd	Short name of the hospital with no space characters (for example “Niguarda” or “Niguarda_hospital”)	Y
	patient_age	The age of the patient at the time of the visit	Y
	import_date	Date data were imported	Y

Observation Fact Dimension

In the I2B2 star schema, **`observation_fact`** is the fact table. In our context and in healthcare in general, a fact is an observation on a patient such as a description of an event, a recording or a notation of something, a clinical measurement, a diagnosis et.al. Every fact must be related to a specific visit of a specific patient to the hospital.

observation_fact			
Fields used in Data Factory		Values	Mandatory (Y/N)
PK	encounter_num	A sequential integer mapped to the original visit id	Y
PK	patient_num	A sequential integer mapped to the original patient id	Y
PK	concept_cd	Code of the variable (may be CDE or local)	Y
PK	provider_id	Short name of the hospital with no space characters. E.g. “Niguarda” or “Niguarda_hospital”. Originally, in I2B2, used for the name of the person importing the data.	Y
PK	start_date	The visit’s start date	Y
PK	modifier_cd	The code for modifier of interest (e.g. “DOSE”). In DF we put ‘@’.	Y
PK	instance_num	Instance number. Usually we have one instance therefore we put ‘1’.	Y
	valtype_cd	‘N’ for numeric values ‘T’ for text (enum / short messages) ‘B’ for raw text (notes / reports)	Y
	tval_char	In correspondence with `valtype_cd`. When `valtype_cd` = ‘T’ ->	Y

		Stores the text value When `valtype_cd` = 'N' -> 'E' = Equals (most common in MIP) 'NE' = Not Equal 'L' = Less Than 'LE' = Less than and Equal to 'G' = Greater than 'GE' = Greater than and Equal to	
	nval_num	In correspondence with `valtype_cd` When `valtype_cd` = 'N' to store a numerical value	N
	units_cd	Units of measurement for the value in `nval_num`	N
	observation_blob	Used in correspondence with `valtype_cd` When `valtype_cd` = 'B' Holds any raw or miscellaneous data that exists	N
	location_cd	Short name of the hospital with no space characters (for example "Niguarda" or "Niguarda_hospital")	Y
	import_date	Date data were imported	Y
	text_search_index	Integer from a sequence	Y

Encounter Mapping Table

The **`encounter_mapping`** table maps every visit (encounter) to a unique sequential integer.

encounter_mapping			
Fields used in Data Factory		Values	Mandatory (Y/N)
PK	encounter_ide	The visit id	Y
PK	encounter_ide_source	Short name of the hospital with no space characters (for example "Niguarda" or "Niguarda_hospital")	Y
PK	project_id	Short name of the hospital with no space characters (for example "Niguarda" or "Niguarda_hospital"). Can be used in another	Y

		way in case of having multiple datasets of the same hospital.	
	encounter_num	A unique integer	Y
PK	patient_ide	the pseudo anonymized patient id	Y
PK	patient_ide_source	Short name of the hospital with no space characters (for example “Niguarda” or “Niguarda_hospital”)	Y
	import_date	Date data were imported	Y

Patient Mapping Table

The **`patient_mapping`** table maps every patient to a unique sequential integer.

patient_mapping			
Fields used in Data Factory		Values	Mandatory (Y/N)
PK	patient_ide	The pseudo anonymized patient id	Y
PK	patient_ide_source	Short name of the hospital with no space characters (for example “Niguarda” or “Niguarda_hospital”)	Y
	patient_num	A unique integer	Y
PK	project_id	Short name of the hospital with no space characters (for example “Niguarda” or “Niguarda_hospital”). Can be used in another way in case of having multiple datasets of the same hospital.	Y
	import_date	Date data were imported	Y

Appendix 2 – Designing and testing mapping tasks with Mipmap Tutorial

This Section is a step-by-step tutorial on how to **create (step 2A) and validate the configuration files for steps 2B (EHR preprocessing), 3A (EHR capture) and 4 (harmonization)** of the Data Factory's pipeline by using mock hospital EHR data. Each major step has a hands-on example and some basic guidelines. The following section essentially describes the step 2A of Data Factory's pipeline which is not performed in the hospital server, but in a local machine with a desktop graphical interface. The instructions can be extended and applied in any case of EHR data with the proper changes. We use csv files with mock data in order to design the mapping tasks and generate the configuration files. **In a real case scenario, we can use look-alike samples resembling the original hospital EHR csv files but with a very limited number of rows** in order to prevent any data privacy breach[1].

Before We Start

In the rest of the Section, after we have created the configuration files for each step, we execute the process on a mock set of data in order to validate the configuration files. The reader is prompted to download this set of data and keep up with the procedure on her own machine.

Needed Software

- docker (v18)
- docker-compose (v1.22 - v1.8)
- python3 (v3.5 or greater), python3-pip, python3-tk
- MIPMAP

EHR Mapping design template deployment

- Download the mock data CSV files from the GitHub repository
  - (https://github.com/aueb-wim/ehr-mapping-tutorial-files)
- Clone the mapping design template (see repo README file for requirements and further deployment instructions)
  - (https://github.com/aueb-wim/ehr-mapping-design-template)
- Go to the newly created mapping design folder, create "source" folder and put all hospital and metadata CSV files into it.
- Build the docker postgreSQL container and the necessary i2b2 databases by executing *build_postgres.sh* bash script.
  - `$sh build_postgres.sh`

Configuration files for Pre-processing EHR Data Step (2B)

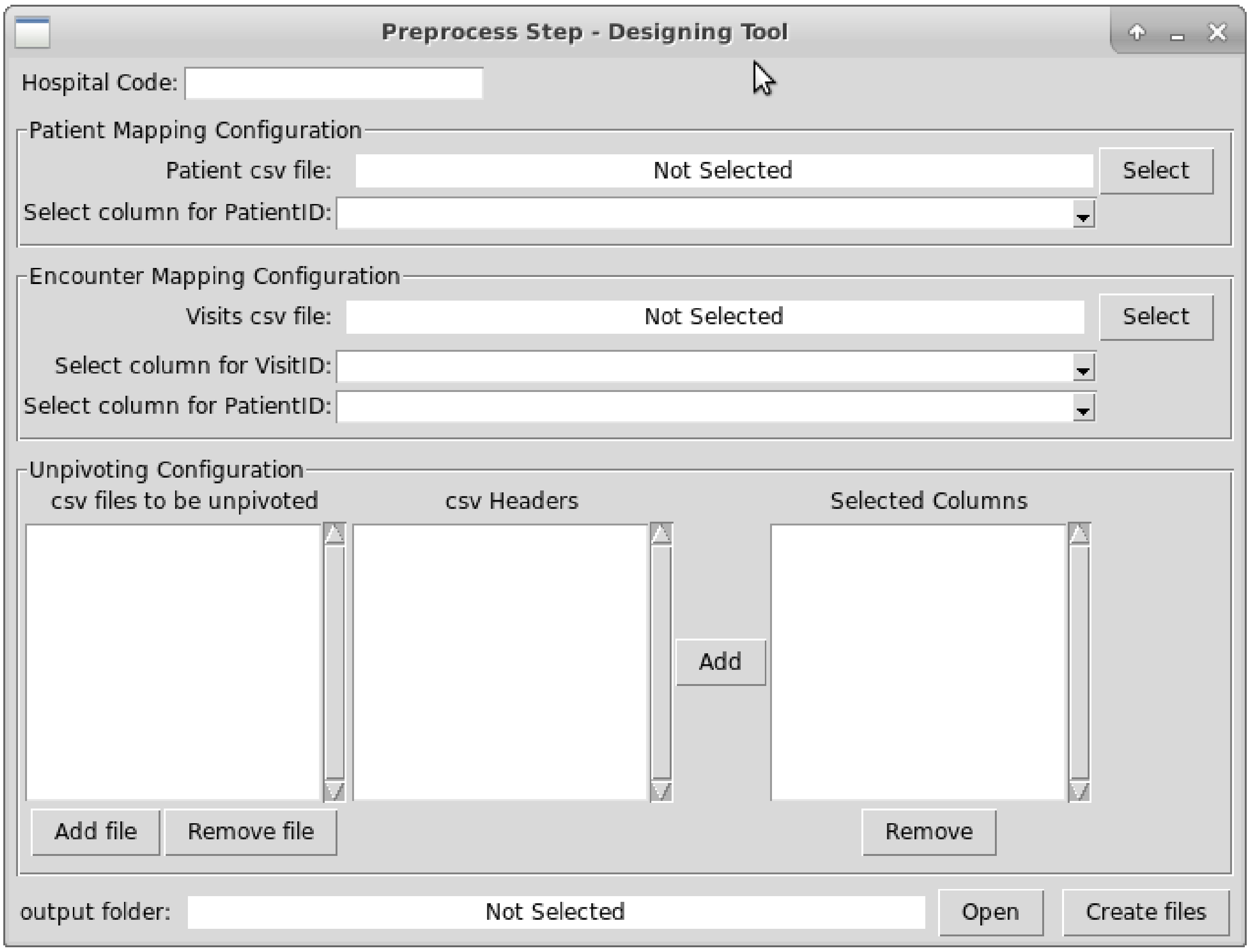
In this step we create the configuration files for producing two types of auxiliary files that are going to be part of the ***source schema*** in the following ***capture mapping-task tutorial***:

[1]Real hospital data are not allowed to leave the hospital server.

- Two CSV files that are essential for the i2b2 schema mapping:
    - EncounterMapping.csv (populates `encounter_mapping` i2b2 table)
    - PatientMapping.csv (populates `patient_mapping` i2b2 table)
- unpivoted CSV files that are produced from the initial hospital CSV files, in our case:
    - unpivoted_EHR_Exams1.csv
    - unpivoted_EHR_Patient_Family_history.csv

Those auxiliary files are going to be created by the MIPMapReduce module by giving as input the mock EHR csv files and the configuration files that we are going to create here. For the creation of those configuration files we use a tool with a GUI written in python (please refer to the ehr mapping design template repo's README file for how to install the required python packages). To start the program execute in the folder *preprocess_tool*:

```
$python3 prepro_gui.py
```


Preprocess Step - Designing Tool

Hospital Code:

Patient Mapping Configuration
Patient csv file: Not Selected
Select column for PatientID:

Encounter Mapping Configuration
Visits csv file: Not Selected
Select column for VisitID:
Select column for PatientID:

Unpivoting Configuration
csv files to be unpivoted
csv Headers
Selected Columns

output folder: Not Selected

Figure 6

1. In the *Hospital Code* we enter "**mock_hospital**".
2. In the *Patient Mapping Configuration* section we select the ***EHR_Exams1.csv*** which contains all the patient id codes and then select the ***patient_id*** column from the drop down list.

3. In the *Encounter Mapping Configuration* section we select the ***EHR_Exams1.csv*** which contains all the patient id codes and then select the ***visit_id*** and ***patient_id*** column from the drop down lists.
4. In the *Unpivoting* section we are going to create configuration files for unpivoting the ***EHR_Exams1.csv*** and ***EHR_Patient_Family_history.csv***.
    a. For ***EHR_Exams1.csv*** we select the columns:
        - patient_id
        - visit_id
        - S_Date_Suivi (contains each visit date)
    b. For ***EHR_Patient_Family_history.csv*** we select the columns:
        - patient_id
        - visit_id
        - P_Sexe (contains the patient's gender)
        - P_Ne_Le (contains the patient's date of birth)
5. Select *preprocess_step* folder as output folder and press the *Create files* button

In *preprocess_step* folder check if the following files have been created:
1. run.sh
2. patientmapping.properties
3. encountermapping.properties
4. selected_EHR_Exams1.txt
5. selected_EHR_Patient_Family_history.txt
6. unpivoted_EHR_Exams1.txt
7. unpivoted_EHR_Patient_Family_history.txt

Testing the configuration files and creation of unpivoted csv

- In mapping design folder give `$sh ingestdata.sh preprocess`
- Check folder "source" if the auxiliary files have been created:
    - o PatientMapping.csv
    - o EncounterMapping.csv
    - o unpivoted_EHR_Exams1.csv
    - o unpivoted_EHR_Patient_Family_history.csv
- Open each file and check if it is not empty

If everything is ok, we are ready to go to the next step.

Some notes about unpivoting csv files

In most cases the patient's examination results are stored in multiple columns in a CSV file. It's a good practice to produce an auxiliary CSV file by unpivoting the columns with the examination results of the initial CSV and keeping the columns with primary/foreign keys along with some other columns. For example let's say we have a csv file:

Column1	Column2	Column3	Column4

Value1	Value2	Value3	Value4

The unpivoted version with the columns 1 and 2 as "selected" columns will be:

Column1	**Column2**	**unPivoted**	**Value**
Value1	Value2	Column3	Value3
Value1	Value2	Column4	Value4

Although this unpivoting procedure adds an extra step to the data preparation, it simplifies the process when designing the mapping task in MIPMap.

Designing the Capture mapping task using MIPMap for Step 3A

We go through all the steps for the preparation and the configuration of the Data Factory pipeline for the mock dataset from https://github.com/aueb-wim/ehr-mapping-tutorial-files that is used in the examples of the previous steps. We assume that all the previous steps have been completed before we continue to the mapping task design. Please note that here we present some of the functionalities of the MIPMap tool. For a complete tutorial of MIPMap's features download the documents at:

https://github.com/HBPMedical/MIPMap/blob/master/MIPMap%20Tutorial.docx

For a demonstration of some basic MIPMAP features when mapping data see also the following VIDEO: Video Introduction to MIPMAP

Create a new mapping-task in MIPMap. Keep in mind that MIPMap stores the mapping task in an xml file. The mapping xml file must be stored in the mapping task design root folder. Select CSV as DataSource Type and put a Database Name (e.g. “HOSPITAL”) then select and add all the CSV files in the source folder. Although we have generated an unpivoted version for both EHR_Exams1.csv and EHR_Patient_Family_history.csv, it might be convenient to use the original files as well (sometimes it might not, depending on the case...).

For Target, choose Relational as DataSource Type. Choose the postgres Driver and fill in the URI and the db user credentials as follows:

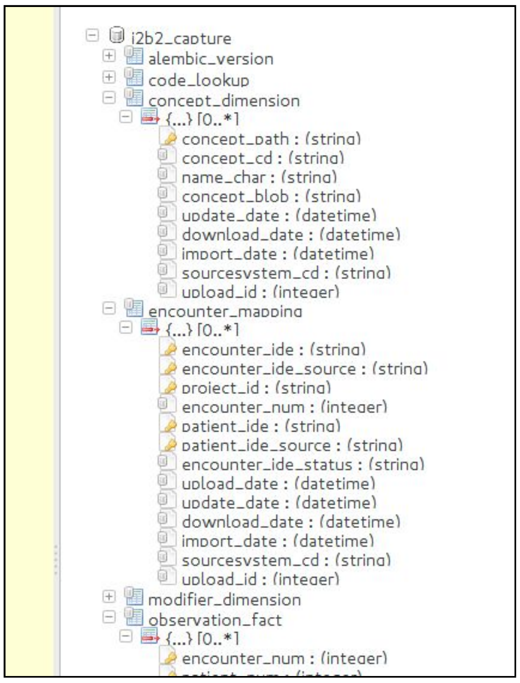
- URI: jdbc:postgresql://localhost:45432/i2b2_capture
- Username: postgres
- Password: 1234

Now we have to design the mapping-task so as to store all information from the EHR CSV files along with the metadata about the hospital variables and the CDEs to the Capture db. Obviously, there is not only one solution to our problem but there are some good practices and some no-no’s. In this User Guide, we elaborate on some of our mapping configurations and explain the logic behind them. We encourage the reader to continue and finish the mapping-task by herself instead of executing our ready-to-go solution.

At first, we look on the right side of the mapping-task the tables of the target i2b2_capture and search for the ones that we do not actually use (not currently at least...) in the context of the Data Factory.

These are `alembic_version`, `code_lookup` and `modifier_dimension`.

We shrink their columns for our ease.



```
i2b2_capture
  alembic_version
  code_lookup
  concept_dimension
    {...} [0..*]
      concept_path : (string)
      concept_cd : (string)
      name_char : (string)
      concept_blob : (string)
      update_date : (datetime)
      download_date : (datetime)
      import_date : (datetime)
      sourcesystem_cd : (string)
      upload_id : (integer)
  encounter_mapping
    {...} [0..*]
      encounter_ide : (string)
      encounter_ide_source : (string)
      project_id : (string)
      encounter_num : (integer)
      patient_ide : (string)
      patient_ide_source : (string)
      encounter_ide_status : (string)
      upload_date : (datetime)
      update_date : (datetime)
      download_date : (datetime)
      import_date : (datetime)
      sourcesystem_cd : (string)
      upload_id : (integer)
  modifier_dimension
  observation_fact
    {...} [0..*]
      encounter_num : (integer)
```

Figure 7: Shrink columns for tables that are not to be filled in

Also, there is one table which has a constant value: `provider_dimension`. We give to its one and only tuple the following values:
provider_id = “HOSPITAL”
provider_path = "/Provider/Hospital/UserGuideHospital"
name_char = "University Hospital for the User Guide"
import_date = datetime()

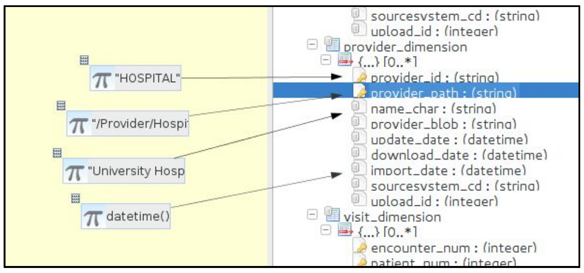


Figure 8: Give constant values to provider_dimension

A mapping that must be done pretty much in every Capture step is the one that maps the CDEs metadata to the `concept_dimension` table. We choose to label these as “CDE” and fill in `sourcesystem_cd` with that label.

As it has been said already, the `concept_dimension` table keeps all information about the variables used in the `observation_fact`. These variables may be CDEs or hospital local ones. Therefore, expect for the CDEs, we need also to store the local variables whose information is taken from the corresponding input CSV file.

To do that, since the original `concept_dimension` is “reserved” taking values from the CDEs metadata file, we need to duplicate it. Another thing we need to pay attention to, is the fact that in the given `hospital_metadata`.csv, the variables that map to one or more CDEs have no value in the `concept_path` column. This is done in order to avoid defining different `concept_path(`s) for the same element. The primary key in the `concept_dimension` table is `concept_path`; therefore, it cannot be left empty.

To deal with the situation we do a Selection Condition for the tuples in `hospital_metadata` having a not null `concept_path`. By doing this, we do not let any tuple without a `concept_path` try to be inserted into `concept_dimension`.

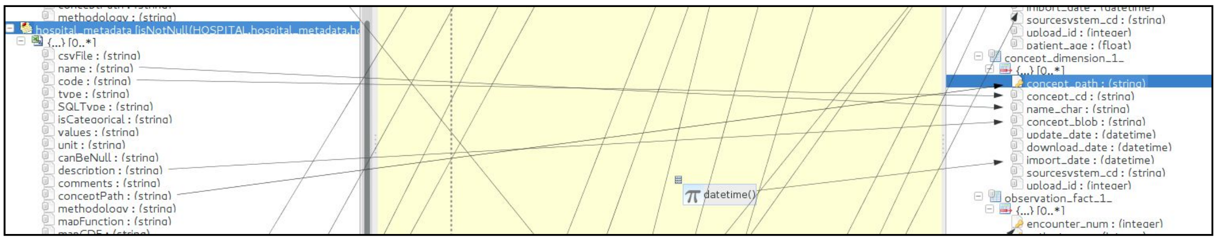


Figure 9: Mapping variables metadata to concept_dimension

We must fill in the mapping tables (`patient_mapping` and `encounter_mapping`). For that cause, we have already created the appropriate CSV files in the pre-processing step. We know `patient_mapping` links a patient’s `patient_ide` to her `patient_num` which is `patient_dimension`’s primary key. In other words, every patient has to have linked tuples in `patient_dimension` and `patient_mapping` having the same `patient_num`.

To do that, we feed `patient_dimension` from source’s `PatientMapping` and `EHR_Patient_Family_history` after we have joined them on `PatientMapping.patient_ide=EHR_Patient_Family_history.patient_id` (to design an inner join in MIPMap we create a connection and check the option ‘Mandatory’).

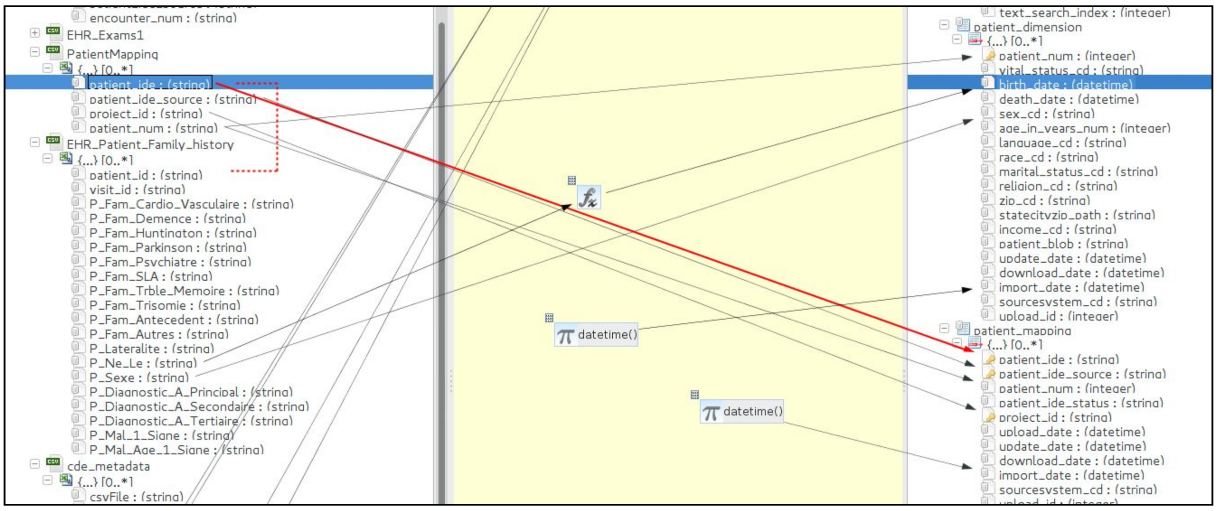
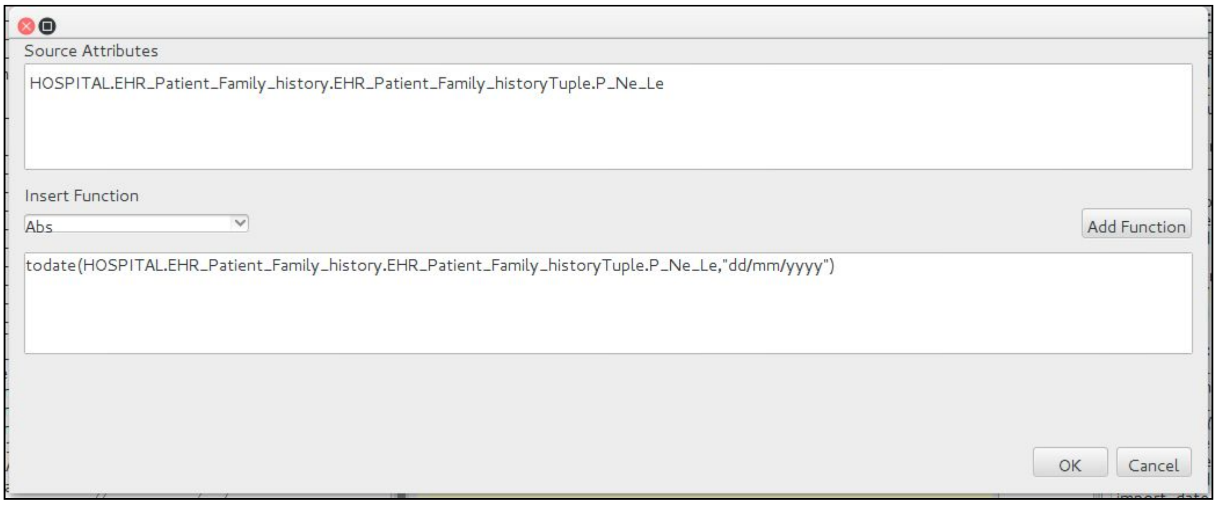


Figure 10: Mapping joined `PatientMapping` and `EHR_Patient_Family_history` to `patient_mapping` and `patient_dimension`

The function shown in Figure 6, mapping P_Ne_Le to birth_date is function todate(). We use it to ensure dates are imported into our target database in proper form.


Source Attributes

HOSPITAL.EHR_Patient_Family_history.EHR_Patient_Family_historyTuple.P_Ne_Le

Insert Function

Abs

Add Function

todate(HOSPITAL.EHR_Patient_Family_history.EHR_Patient_Family_historyTuple.P_Ne_Le,"dd/mm/yyyy")

OK Cancel

Figure 11: Using function todate() to transform dates into proper form

As it is depicted in Figure 6, in table `EHR_Patient_Family_history` we have left a lot of information unmapped since -leaving id's aside- from all 18 columns we mapped only `P_Ne_Le` and `P_Sexe` which are date of birth and gender. The reason for that is the fact that `patient_dimension` does not have corresponding columns to fill in with this information. Most of `EHR_Patient_Family_history` columns deal with the patient's family history in brain diseases.

We will use the unpivoted version of the CSV and store these variables in `observation_fact`. For filling in `observation_fact`'s encounter_num we join `unpivoted_EHR_Patient_Family_history` with (a duplicate instance of) `EncounterMapping`. We do likewise for `patient_num` whose value we take from

`PatientMapping` and for `start_date` which we take from `EHR_Exams1`. For these three last joins, we use duplicated instances of `PatientMapping`, `EncounterMapping` and `hospital_metadata` to separate the joins from other joins they already participate in.

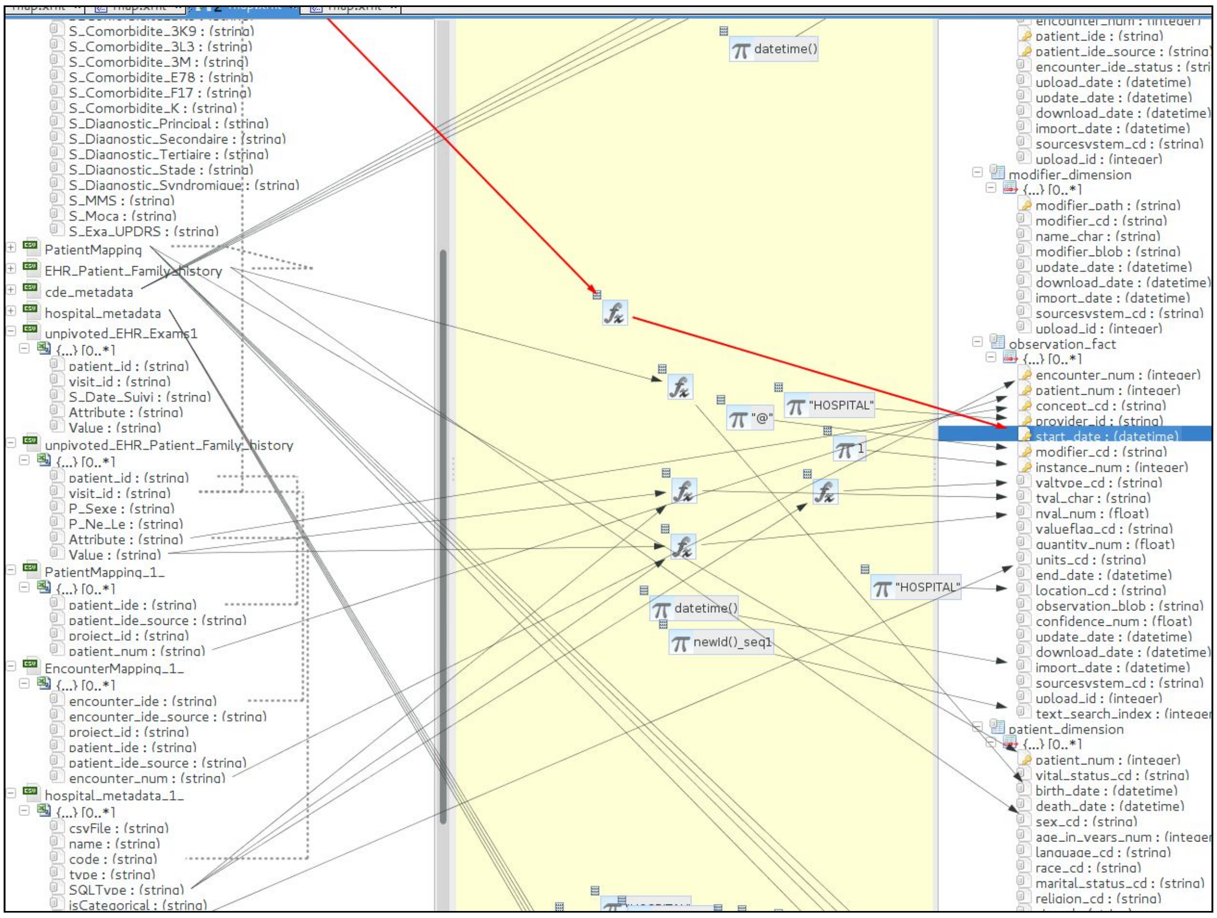


Figure 12: Mapping patient's variables about her family's history

The way the columns valtype_cd, tval_char and nval_num are filled in is pretty much standard. We use the information about the type of every variable given in `hospital_metadata`.

The functions defined in MIPMap are the following:

valtype_cd = if((HOSPITAL.hospital_metadata_1_.hospital_metadataTuple.SQLType == "text"), "T", "N")
tval_char = if((HOSPITAL.hospital_metadata_1_.hospital_metadataTuple.SQLType == "text"), HOSPITAL.unpivoted_EHR_Patient_Family_history.unpivoted_EHR_Patient_Family_historyTuple.Value, "E")
nval_num = if((HOSPITAL.hospital_metadata_1_.hospital_metadataTuple.SQLType == "text"), null(),

```
HOSPITAL.unpivoted_EHR_Patient_Family_history.unpivoted_EHR_Patient_Family_historyTuple.Value)
```

Towards the end of this mapping-task for the Capture step, we need to map the variables about the visits from `EncounterMapping` and the unpivoted version of `EHR_Exams1`. We map `EncounterMapping`'s variables to target's `encounter_mapping`. Information from the unpivoted version of EHR_Exams1 will be stored to the `observation_fact` table.

First, we realize we need to create a duplicate instance of `observation_fact` since the first one is already reserved for `unpivoted_EHR_Exams1`. Upon creating that, we see that we need to join `unpivoted_EHR_Exams1` to both `EncounterMapping` and `PatientMapping` to provide `observation_fact` with `encounter_num` and `patient_num`.

To be totally sure no other join is affected, we choose to join with two new duplicates for `EncounterMapping` and `PatientMapping`. To fill in the three aforementioned columns from `observation_fact` referring to the observation's type and value (`valtype_cd`, `tval_char` and `nval_num`) we will again need information from `hospital_metadata`. Therefore, we join a duplicate of that table as well. From this four-table join we give values to the duplicate of `observation_fact`.

To be more precise, we join `unpivoted_EHR_Exams1` with duplicates `EncounterMapping_2_` and `PatientMapping_2_` and map the join product to duplicate `observation_fact_1_` (the first `observation_fact` instance is already reserved for `unpivoted_EHR_Exams1`). We design the correspondences and the transformations with the same way of thinking as we did for the first instance of `observation_fact`.

Finally, in order to fill in the `visit_dimension` table, we join a duplicate of `EHR_Exams1` with new duplicates of `PatientMapping` and `EncounterMapping` and map the join product to `visit_dimension`.

If we have done all mappings correctly, the MIPMap engine will insert all the information in the Capture database along with the already imported imaging features.

Execute and test the capture mapping-task

In order to execute the designed mapping-task to the Capture db of the docker image we must do the following steps:

- Save the mapping task file in the main folder with the name map.xml
- In design folder give the command **`$ sh templator.sh capture`**
- In design folder give the command **`$ sh ingestdata.sh capture`**

This is the actual command where the MIPMap engine takes as input the designed mapping-task and generates the tuples in the target database.

If everything go smoothly, the i2b2_capture database is supposed to be filled in. Check if data have been imported correctly. You can use psql or pgAdmin or any other postgres client. For the specific mock dataset, `patient_dimension` table should now have 9 rows while `observation_fact` should have 543.

NOTE: From the explanation of the capture step mapping-task it may have been evident already that designing a mapping-task might be quite complex in some cases. Therefore, the MIPMap user is advised to build up on her map.xml configuration gradually, while testing its current state within a reasonable period of mapping steps, to locate errors early on.

Designing the Harmonization mapping task using MIPMap for Step 4

In this step we design the mapping-task for the harmonization of the values. This means that we implement all the harmonization rules that have been defined by researchers and clinicians to the local hospital variables to conform to the rules of the CDEs. Same as in Step_2, we elaborate on the mappings for our mock dataset.

Create a new mapping-task in MIPMap. For Source schema, choose Relational as DataSource Type. Choose the postgres Driver and fill in the URI and the db user credentials as follows:

- URI: jdbc:postgresql://localhost:45432/i2b2_capture
- Username: postgres
- Password: 1234

For Target, choose Relational as DataSource Type. Choose the postgres Driver and fill in the URI and the db user credentials as follows:

- URI: jdbc:postgresql://localhost:45432/i2b2_harmonized
- Username: postgres
- Password: 1234

In contrast to the previous mapping-task, where we went from CSVs to i2b2, in this one we translate an i2b2-schemaed relational database to another i2b2-schemaed one.

Starting off, we want the tuples in tables `concept_dimension` and `encounter_mapping` to be transferred as they are.

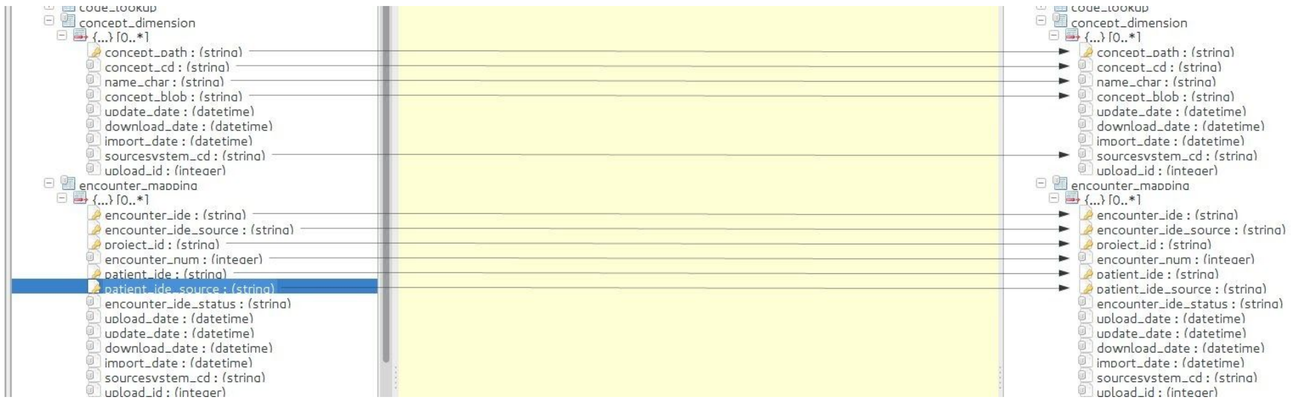


Figure 13: Passing values as they are

We do not need to draw corresponding lines for the rest of the columns since they do not have values. In case we do there will be no problem of course.

In the observation_fact table, where most variables are stored, we need to harmonize the values and/or the concept_cd's of a few of them (harmonization rules are described in hospital_metadata.csv in columns mapFunction and mapCDE). These are the following:

- ✓ For variable ‘P_Lateralite’ concept_cd must change to ‘‘handedness’ and values must change from ‘D’, ‘A’ to ‘R’, ‘L’ respectively.
- ✓ For variable ‘S_Exa_UPDRS’ concept_cd must change to ‘updrshy’.
- ✓ For variable ‘S_MMs’ concept_cd must change to ‘minimentalstate’’.
- ✓ For variable ‘S_Moca’ concept_cd must change to ‘montrealcognitiveassessement’.

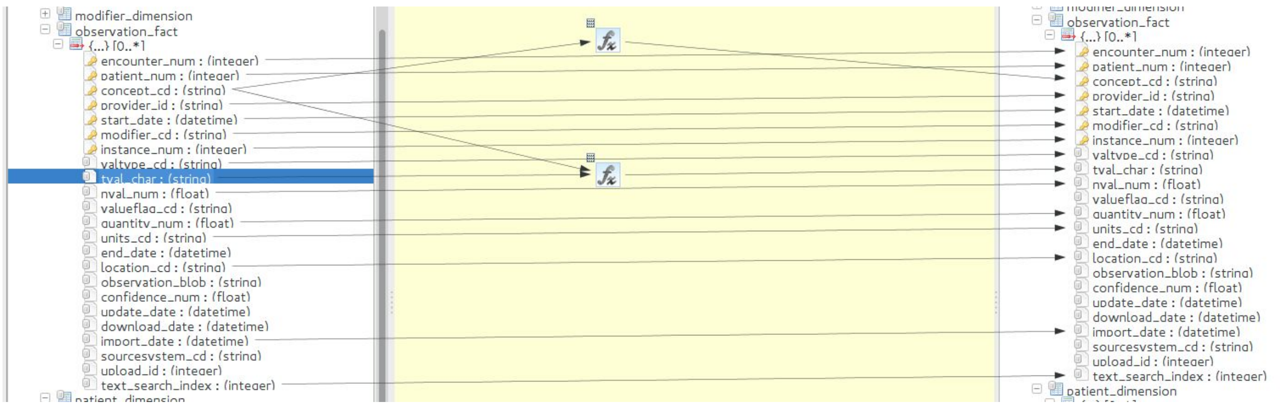


Figure 14: Harmonizing values in concept_cd and tval_char columns

To implement these harmonization rules we defined the following functions:

concept_cd = if(i2b2_capture.observation_fact.observation_factTuple.concept_cd=="P_Lateralite", "handedness", if(i2b2_capture.observation_fact.observation_factTuple.concept_cd=="S_Exa_UPDRS","updrshy", if(i2b2_capture.observation_fact.observation_factTuple.concept_cd=="S_MMS","minimentalstate", if(i2b2_capture.observation_fact.observation_factTuple.concept_cd=="S_Moca","montrealcognitiveassessment",i2b2_capture.observation_fact.observation_factTuple.concept_cd))))
tval_char = if(i2b2_capture.observation_fact.observation_factTuple.concept_cd=="P_Lateralite", if(i2b2_capture.observation_fact.observation_factTuple.tval_char=="A","L","R"),i2b2_capture.observation_fact.observation_factTuple.tval_char)

The rest of the mappings are plain correspondences as in Figure 13.

Execute and test the harmonization mapping task

- Save the mapping task file in the main folder with the name mapHarmonize.xml
- In design folder run `$ sh templator.sh harmonize`
- In design folder run `$ sh ingestdata.sh harmonize`

Check if the i2b2_harmonized is populated with data as expected. The number of tuples should not change from before.

<u>**NOTE:**</u> It is obvious that the design of the harmonization mapping task is way less complex and demanding from its precedent capture mapping task. This is normal and expected since in the harmonization step we transfer data from a database to another one with the exact same schema, while in the capture step there are a lot of differentiations in schema and in the way we store tuples (pivoted vs unpivoted).