

MIPMap tutorial


MIPMap is a schema mapping and data exchange tool. It offers an easy to use graphical interface where the user, given a source and target schema, can define correspondences/mappings by simply drawing arrow lines between the elements of the two tree-form representations. In order to specify these mappings we use declarative representations, under the formalism of Tuple-generating dependencies (TGD), that are executed using an advanced, highly scalable and efficient mapping execution engine. The user except for creating new mapping tasks can also load ones previously created.

1. Tool configuration

Starting MIPMap *for the first time*, it creates a **local Postgres database** (the default name of the database is 'mipmaptask'). The database connection information, such as the username and password, the connection uri and port or the prefix of the name of the database that MIPMap will create, can be configured through a properties file (*postgresdb.properties*), which is found in *<MIPMap Installation path>\mipmap* installation folder.

Any change performed on the properties file will only take place after MIPMap has been restarted, since it loads the database properties file when it starts.

2. Creating/Loading a new mapping task

To start a **new mapping task** the user clicks on 'New' under the 'File' menu or on the  icon from the system bar. The supported input types for the source and target schema can be CSV or Relational (database). The source and target schemata in the same mapping task can have different input types.

- For CSV input (Figure 1), the user must enter a database/schema name and select and add the CSV files that will compose the source schema, each corresponding to a table. The user can also choose whether the files contain only the column names in their first line or data as well. The CSV delimiter used in input files should be the 'comma; (,)' character. The option to change the values' separator is provided by the tool under the 'Preprocessing' functions and is discussed in detail in Section 9.2.

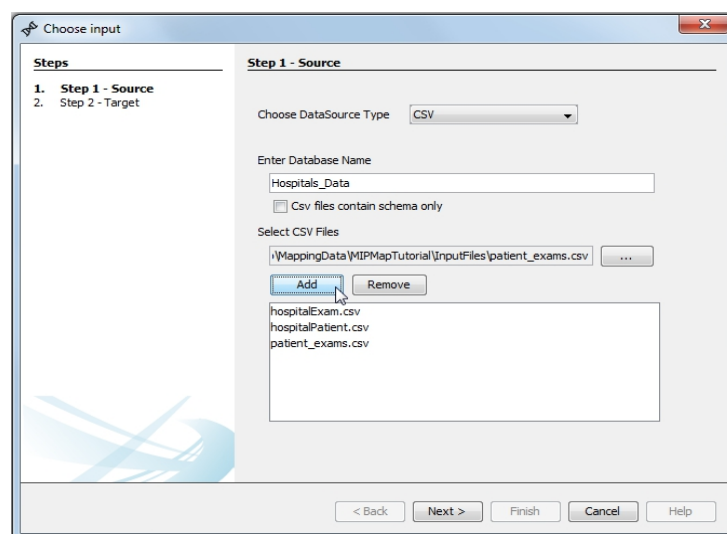


Figure 1- Choosing CSV files as source schema input


- For Relational input (Figure 2) a driver must first be selected. After that, most of the information in the 'Uri' field is filled automatically leaving to the user to fill in the host address and the database name. A username and a password for this database must also be entered.

The screenshot shows a 'Choose input' dialog box with a 'Steps' panel on the left and a 'Step 2 - Target' configuration panel on the right. The 'Steps' panel lists '1. Step 1 - Source' and '2. Step 2 - Target'. The 'Step 2 - Target' panel contains the following fields:

- Choose DataSource Type:** A dropdown menu set to 'Relational'.
- Driver:** A dropdown menu set to 'org.postgresql.Driver'.
- Uri:** A text field containing 'jdbc:postgresql://localhost/MIP_Database'.
- Username:** A text field containing 'postgres'.
- Password:** A text field with masked characters '.....'.

At the bottom of the dialog, there are five buttons: '< Back', 'Next >', 'Finish' (highlighted in blue), 'Cancel', and 'Help'.

Figure 2 - Providing database information for target relational input

To **load an existing mapping task** the user selects 'File' -> 'Open' or clicks on the  icon and then chooses a previously saved mapping task. Apart from the task itself, a tab containing the mapping task's information in xml format (the format in which tasks are saved) is created so that the user can review the configuration and the options of the loaded scenario.

After information for both the source and target schema has been provided, or a task has been selected for loading, MIPMap creates two *database schemata* in the local Postgres database, one for the original source schema and one for the target one, taking into consideration existing primary keys.

After the newly created mapping task is loaded it is represented in the Mapping Task Main panel (Figure 3), which consists of three areas. In the left area, the user is presented a tree-form representation of the source schema, where each leaf node corresponds to a table attribute followed by its data type. Primary key constraints -and possible foreign key saved on the mapping task xml- are also represented in the schema, the first by a 'key' icon on the appropriate element and the latter by a grey arrow connecting the referenced elements. The right-most area contains a respective tree-form representation for the target schema.

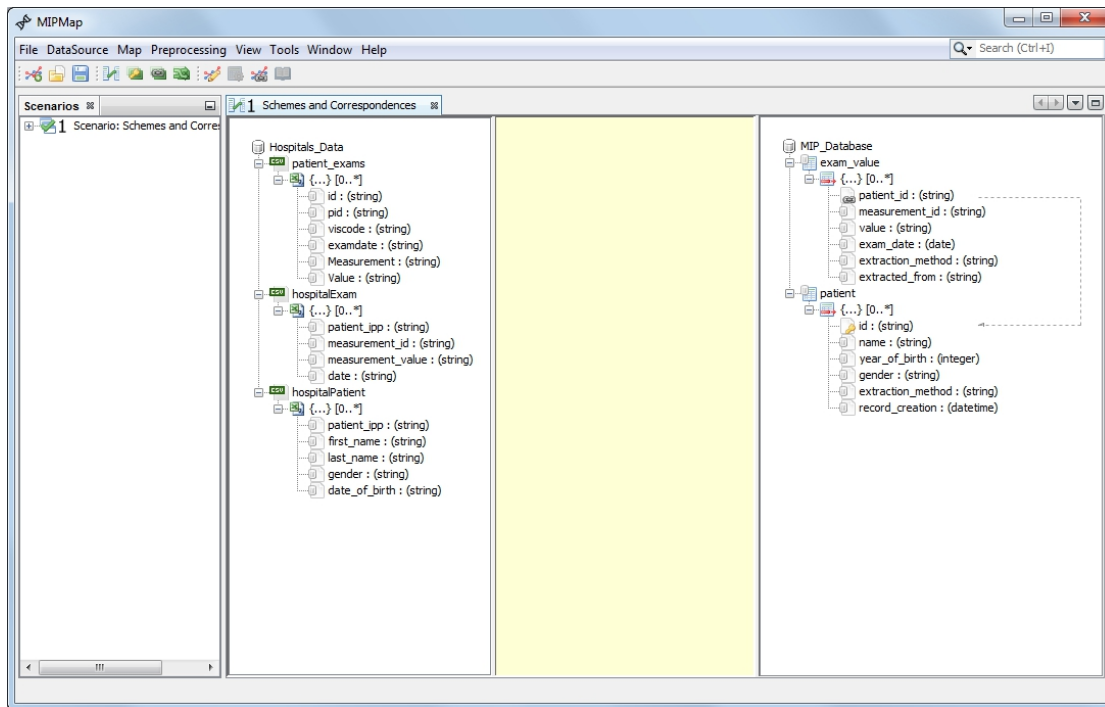


Figure 3 – New mapping task with CSV and relational input. No correspondences have been drawn yet.

It should be noted that more than one mapping tasks can be loaded. The ‘Scenarios’ panel on the left-most area of MIPMap offers navigation between the tasks and quick access to some of the most important features of the tool, which will be described in detail in Section 8.

3. Designing a mapping task

3.1 Correspondences between elements

In the mapping task Main panel the user can drag arrows from one leaf node to another. In case the nodes belong to the same tree (source or target) a foreign key constraint is created, while when the nodes belong to different trees a **correspondence** between the two attributes is created. Note, that a correspondence from a target tree node to a source tree node is not allowed and cannot be created.

An additional option to create simple correspondences is provided (Figure 4). By right clicking on a leaf node the user has the option to set the current node either as source or target of a simple correspondence connection. A node must have been set as source for the connection before another one is selected as target. Additionally, only the last node that has been set as source is taken into account for the connection. This option can be used to create foreign keys too, if the two nodes selected belong to the same tree.

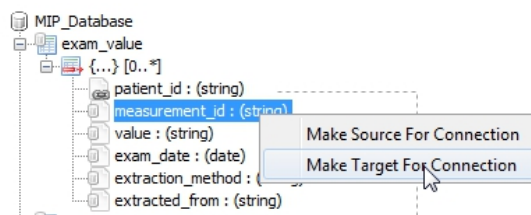


Figure 4 – Selecting a leaf node as the target point of a connection

When the user clicks on a leaf node the correspondences from -or to- that node are highlighted making it easier for her to distinguish the path of the mapped elements (Figure 5).

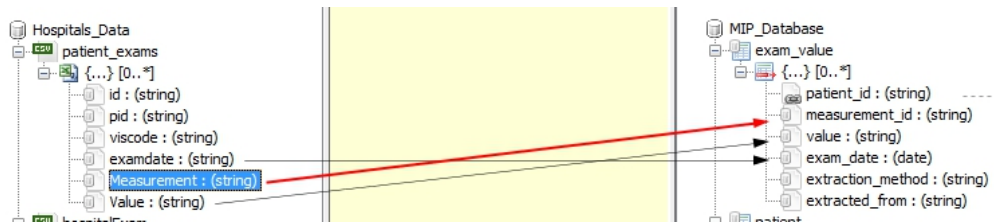


Figure 5 - Highlighted correspondence between two elements

By right clicking in the center area with the yellow background (Manipulation area) the following abilities are offered to the user:

- Define a **constant value** over the target instances (Figure 6). The constant value can either be a given *string* or *number* value or generated from a *function* (the date or date and time of the mapping task, a sequence number etc.). Constant value options are presented after double-clicking on the constant icon: π . An example of how this constant value is represented in the Manipulation area is shown in Figure 7.

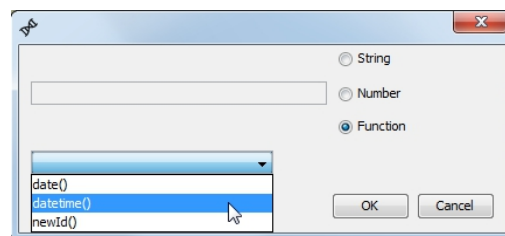


Figure 6 - Constant options

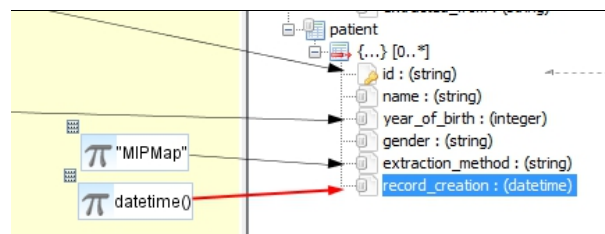


Figure 7 - Constant correspondence examples

- Create a complex **transformation function** (Figure 8) that uses one or more source attributes and maps the result to the target schema. Transformation function options are presented after providing input to and double-clicking on the function icon: \mathcal{F}

There is a big selection of functions that the user can choose from, from string (Append, Substring, Trim, Lower/Upper case) to mathematical ones (Round, Absolute value, trigonometric functions), even more complex ones, like 'If' function, and casting ones (to String, to Integer). The user must first drag at least one arrow from a source leaf node to the function icon. Then, by double clicking on the icon the function menu appears, in which available functions can be selected and inserted into the type area.

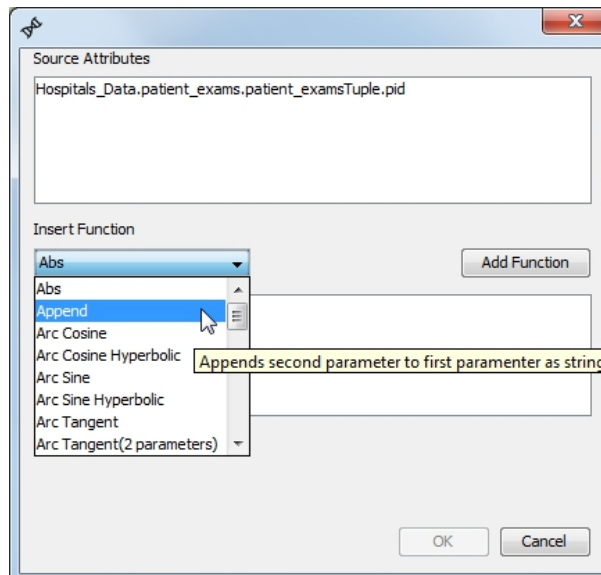


Figure 8 - Selecting a transformation function

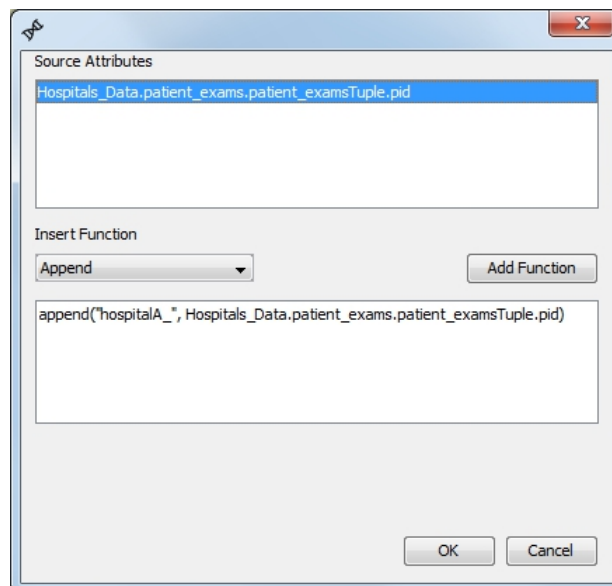


Figure 9 - Appending the 'hospitalA_' string text to the 'pid' attribute value using a transformation function

The user types the string, numeric or boolean values of the input variables in the type area. String values must be enclosed in double quotes (" ") and not in single quotes. Clicking on an item of the source attributes list adds it on the type area. Functions can be used inside other functions too.

For example (Figure 9), to concatenate first name with last name into one field and separate them with a blank space the transformation function can be:

```
append ( Hospitals_Data.hospitalPatient.hospitalPatientTuple.first_name,
append (" ", Hospitals_Data.hospitalPatient.hospitalPatientTuple.last_name) )
```

or to get the year from a date string:

```
substring( Hospitals_Data.hospitalPatient.hospitalPatientTuple.date_of_birth, (
len (Hospitals_Data.hospitalPatient.hospitalPatientTuple.date_of_birth) - 4 ) )
```

After that the user can drag an arrow from the function icon to one *or more* leaf nodes of the target tree to assign its value to it.

- Define a **functional dependency** (egd) over the attributes. The elements related with incoming arrows compose the determinant set, while the elements pointed from the functional dependency icon compose the dependent attributes.

3.2 Join conditions and Foreign key constraints

MIPMap also offers users the ability to apply **join conditions** (Figure 10) to the source sets by specifying a join dependency between two elements as 'Mandatory'. Join dependencies are defined with drag-and-drop operations among elements of the same schema and are represented by a grey line –or arrow in case they are not bidirectional and corresponding to a foreign key constraint as previously noted.

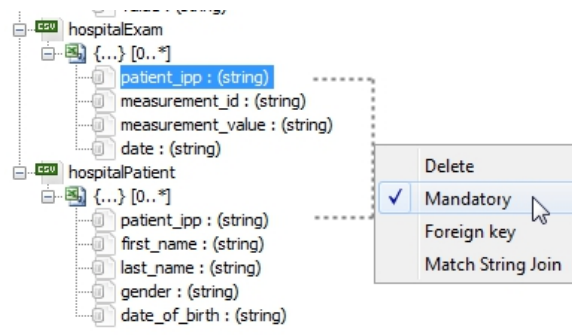


Figure 10 - Applying a mandatory join condition between two attributes

The user can right click on the connecting line and define the dependency as *mandatory* or *optional*, making the line bold, as well as choose if it is bidirectional or not ('Foreign Key' option). The last option provided ('Match String Join') can be selected if the user wishes for the joined values to be substrings of each other, rather than match the whole string value. This is denoted by making the line blue.

3.3 Additional options

In addition, **selection conditions** (Figure 11, Figure 12) can be applied to source sets by right clicking on a set node (the ones representing the tables), choosing to 'Edit Selection Condition' and typing the appropriate condition which will be then shown next to the table name. Typing the desired column name (leaf node name) is adequate, meaning it isn't necessary for the user to type the whole tree path to the leaf node. Supported symbols are equality (`==`, `!=`), comparison (`<`, `>`, `<=`, `>=`), arithmetic (`+`, `-`, etc) and conditional operators (`&&`, `||`), while, as in transformation functions, string values must be enclosed in *double quotes* (`" "`) and *not* in single quotes.

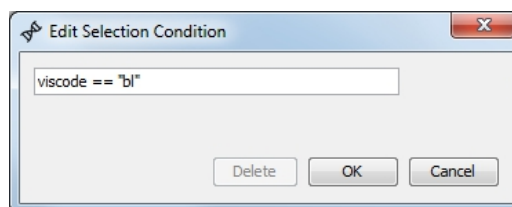


Figure 11 - Typing a selection condition, so that only baseline ("bl") exams will be translated.

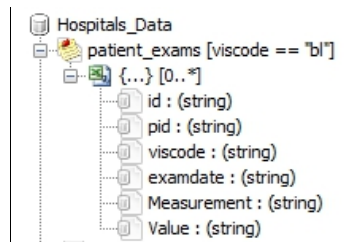


Figure 12 - Created selection condition appearing on node

Users are allowed to **duplicate sets** by right clicking on a set node of either schema and selecting to 'Duplicate Node'. This way, self-joins can be applied, for example, or different selection conditions on the same table can be used.

For instance, in the above example, if the user wishes to map exam data from both 'hospitalExam' and 'patient_exams' tables to the target 'exam_value' table while in the field 'extracted_from' have a constant value indicating the source of the mapping, the table 'exam_value' has to be duplicated as shown in Figure 13.

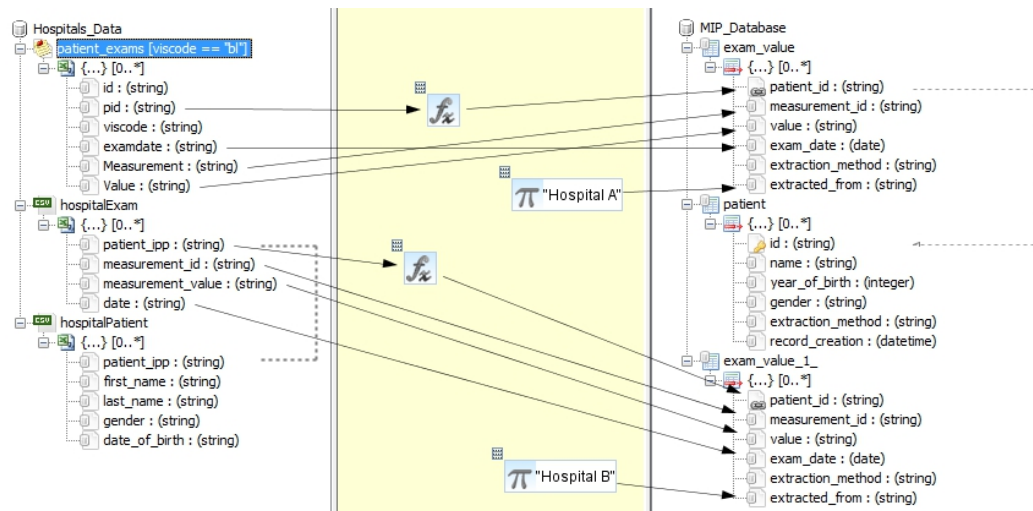


Figure 13 - Duplication of target table 'exam_value'

4. Loading/Adding Instances

Instance data (a sample of 100 rows per table maximum) can be viewed on the 'View Instances Window' panel that users can access by selecting 'View Instances Sample' under the 'DataSource' menu from the systems bar or via the 'Scenarios' left-most panel of the tool. The instances are represented in tree-form as well, where each set node still corresponds to a table, but its children represent the table tuples and leaf nodes correspond to the values of each column.

MIPMap also allows users to add instances both to source and target schema. This utility can be accessed through the systems bar under the 'DataSource' menu. The data must be in CSV format and the user can choose whether the files include column names or not (Figure 14).

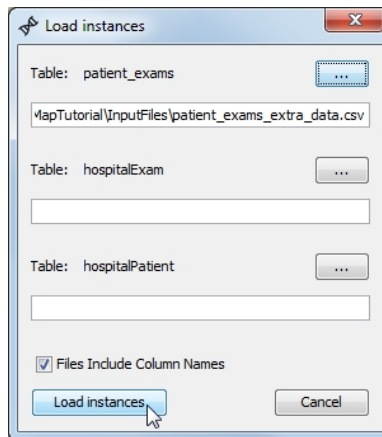



Figure 14 - Loading new instances for source schema table

When the new instance data are loaded they are shown in a separate Instance tab than initial data -if any- in the 'View Instances Window' panel. However, all instances participate during the data exchange process between schemata (Section 6).

5. TGD Generation

After correspondences between elements have been defined the user can compile the mapping scenario and **generate TGDs** (Tuple Generating Dependencies) [1] by clicking on the  icon or on 'Map'→'Generate Transformations'.

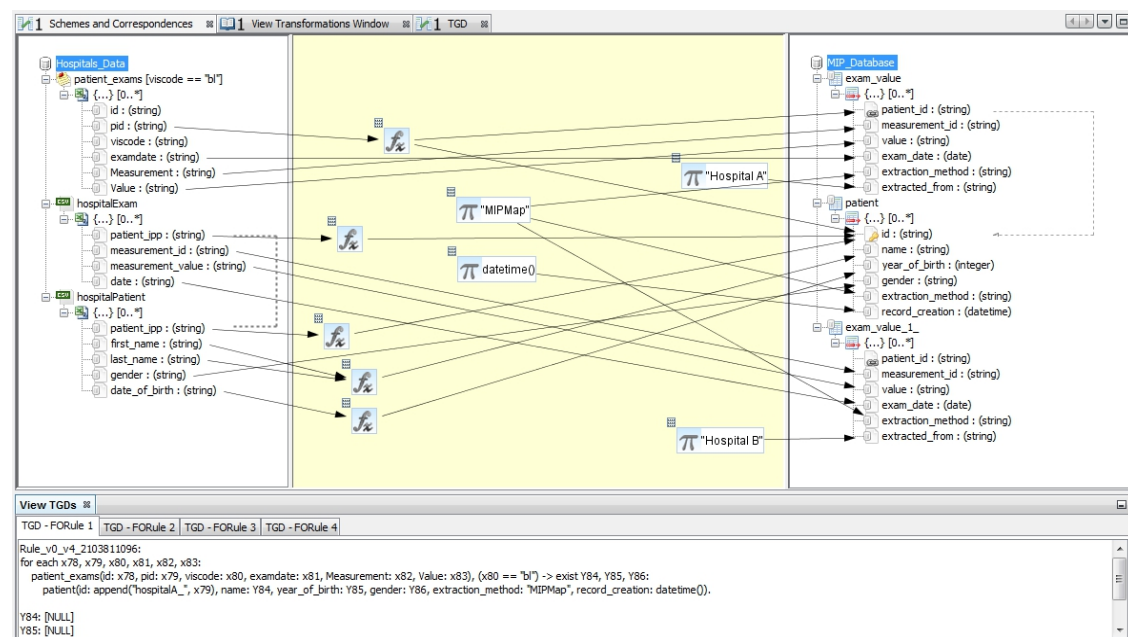


Figure 15 - Full mapping task with generated TGDs

The following three panels are shown (Figure 15):

- The 'View Transformations Window' which includes all the details about the schemata, their constraints, the original source-target dependencies and their rewritings to enforce target egds and compute optimal solutions.
- The 'View TGDs' panel that appears below the main panel and lists all TGD rules that compose the final mapping between the two schemata.
- The 'TGD' panel that contains a graphical representation of each TGD rule. This panel is non-editable, however by clicking on a tab of the above mentioned 'View

TGDs' panel the 'TGD' panel changes automatically and shows the correspondences that refer to that rule only (Figure 16).

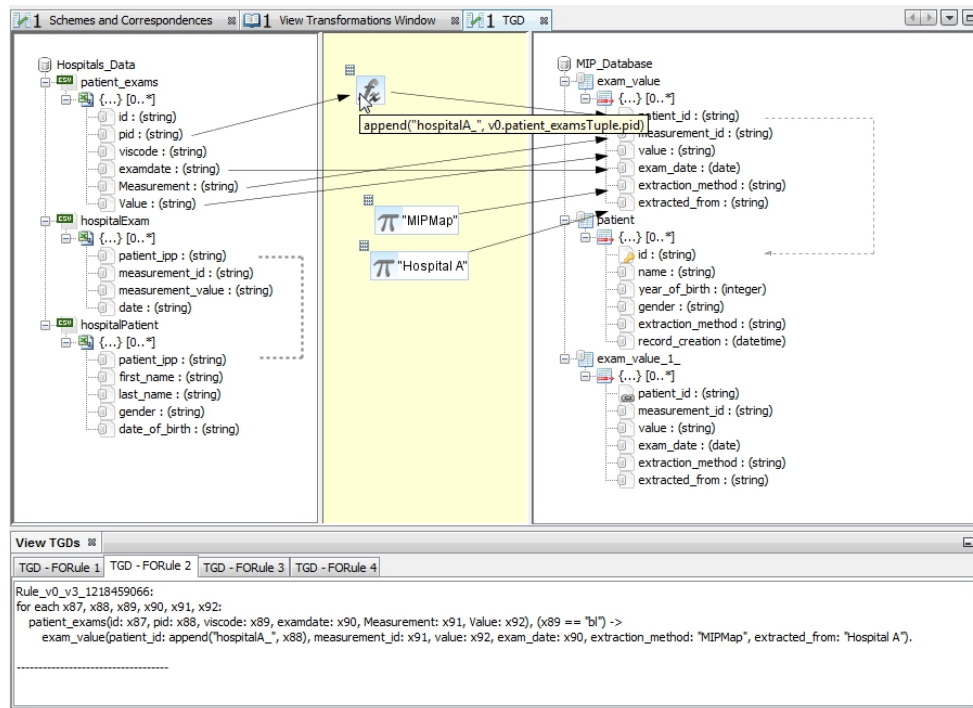


Figure 16 - TGD view for 'TGD - FORule 2'.



On hovering over function icons the corresponding functions are shown.

In case *only constant values* are mapped to a target table a corresponding pseudo-TGD rule, named '**Constant FORule**', is generated and represented in the graphical interface.

If changes in the mapping task have been made (e.g. a new correspondence, constraint etc. is added, edited or removed) and one of those panels is clicked, MIPMap notices the changes and notifies the user asking to refresh its content.

The TGDs that are produced are **rewritten** by the MIPMap engine in order to ensure that the solutions generated are optimal. Once the mapping has been compiled in its logical form, it is possible to **generate the corresponding executable scripts** (XQuery or SQL) from the 'Scenarios' panel or from the system bar under the Map menu. The SQL script is based on these rewritten TGDs and is the one that is going to be executed in the PostgreSQL database (*see Section 1*) during the 'Translation' step.

6. Instance Translation and Data Exchange

After the mapping task has been created the user has the ability to **translate instances** from source to target schema. This can be done by clicking on the  icon or on 'Translate' under the 'Map' menu from the system bar. Moreover, clicking on the  icon allows users to generate transformations and translate instances at once.

What happens during the translation step is that the SQL script that is generated based on the rewritten TGD rules runs on the Postgres database of the appropriate mapping task and inserts the translated instances to the 'target' schema of the database. The user is notified when the translation process has finished successfully. Samples (100 instances per table) of the translated instances which form the solution are shown in the target side of the 'View Instances Window', under the 'Solution' tab (Figure 17).

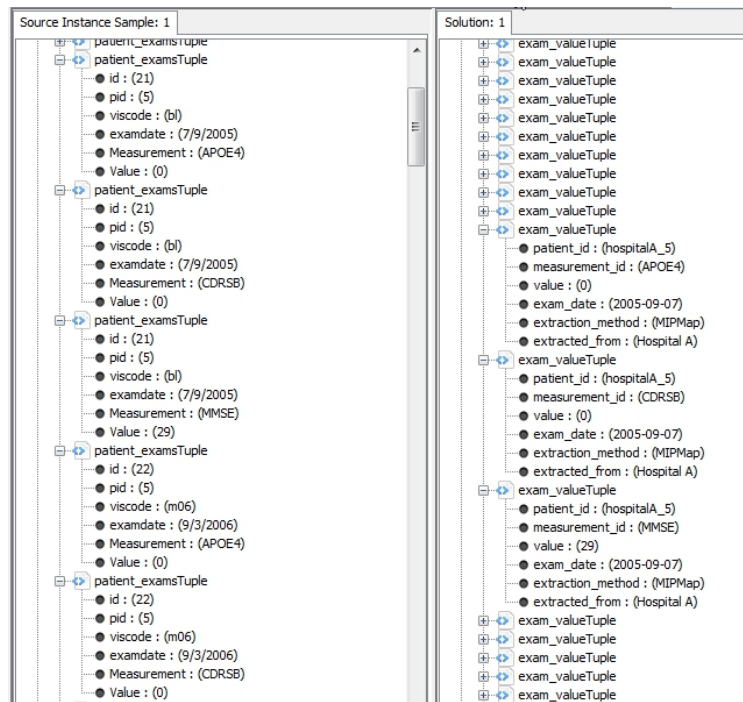


Figure 17 - Translated instances for patient with pid 5. Notice that only 'bl' measures were translated

The user then has the ability to **export the translated instances** in *CSV* or *JSON* format, either by creating a new file or by appending them to an existing CSV file that has the same structure after selecting the *directory* where the files will be exported to. The above commands can be found under the 'Map' menu section.

In case, during the data exchange process, **primary key constraints** on the target database are violated, MIPMap informs the user, who is then given the choice to export the instances that could not be loaded to the target schema. If, for example, two or more different instances with the same values in primary key fields are to be inserted into the target database only the first one is actually translated, while the other ones can be chosen by the user to be exported separately either in CSV or JSON format.

7. Closing/saving a mapping task

After closing the corresponding mapping task all Postgres schemas created by MIPMap are automatically deleted.

Mapping tasks can be **saved in xml format**, so that they can be easily accessed in the future.

8. 'Scenarios' configuration area

If more than one mapping tasks are opened simultaneously the TGD Generation and Translation actions correspond to the mapping task that is currently selected. By default the last opened mapping task is the one selected, but the user can **select a task** via the rightmost Scenarios panel by right clicking on the desired scenario (Figure 18). An option to remove or close a mapping task is also provided.

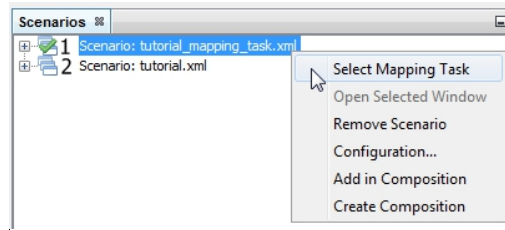


Figure 18 - Mapping task selection

By opening the **Configuration** menu (Figure 19), another option provided in the same panel, the user can choose for the translation process to implement EGDs (Equality Generated Dependencies), so that the optimal generated solutions will also take into account the dependencies between target elements. It must be noted that not all mapping scenarios can produce solutions with EGDs being taken into consideration. In such cases the user is informed with a warning message.

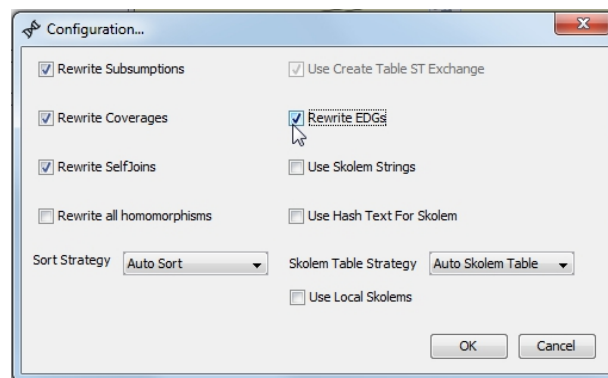


Figure 19 - Mapping task configuration options

More information on designing a mapping task with target constraints can be found at [2][3]. The configuration options are also saved with the mapping task.

9. Preprocessing

9.1 Un-pivot table (CSV input)

It has been mentioned that MIPMap supports CSV input, where each CSV file corresponds to a table. Therefore, in order to avoid cases when some of the files do not form a relational database, an additional feature has been implemented. Under the 'Preprocessing' menu on the systems bar the user can choose to '**Un-pivot tables**' that are given in CSV format.

Un-pivoting refers to transforming selected columns into attribute-value pairs where columns become rows. The user selects which columns to un-pivot and types the name of the new attribute column, while the value column name will automatically be selected by the system as 'Value'. These columns are inserted at the end of the file. The user also has the option to select some of the original columns to be completely omitted in the newly created file by unchecking the corresponding 'Include column' checkbox. The newly created CSV file is saved at the folder location of the original file as 'unpivoted_' plus the name of the original CSV file.

For example, let's assume that the CSV file 'patient_exams_original' contained the original data in the following form (Figure 20):

id	pid	viscode	examdate	APOE4	CDRSB	MMSE	MOCA
10	3	bl	12/9/2005	1	4,5	20	NA
11	3	m06	13/3/2006	1	6	24	NA
12	3	m12	12/9/2006	1	3,5	17	NA

Figure 20 - First three rows of the 'patient_exams_original' file

Since 'measurement id' is included in the form of column names (the last four columns) the data needs to be reshaped into a format useful for relational operations. Selected columns (the 'MOCA' column in this case) can also be discarded in the new table (Figure 21).

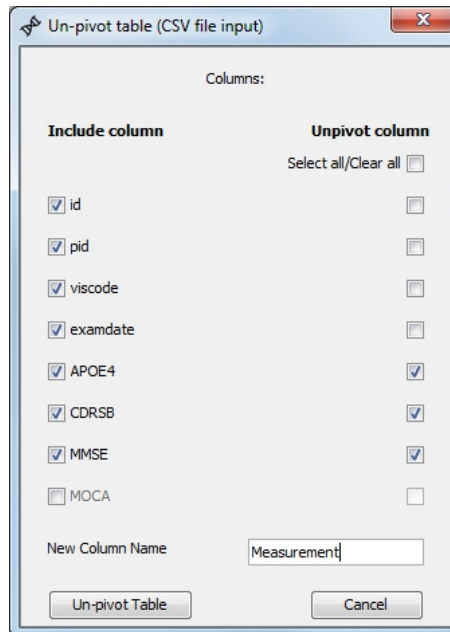


Figure 21 – Un-pivoting three columns into rows and omitting another column

The result of the un-pivoting process (for the initial first three rows) is shown below:

id	pid	viscode	examdate	Measuremer	Value
10	3	bl	12/9/2005	APOE4	1
10	3	bl	12/9/2005	CDRSB	4,5
10	3	bl	12/9/2005	MMSE	20
11	3	m06	13/3/2006	APOE4	1
11	3	m06	13/3/2006	CDRSB	6
11	3	m06	13/3/2006	MMSE	24
12	3	m12	12/9/2006	APOE4	1
12	3	m12	12/9/2006	CDRSB	3,5
12	3	m12	12/9/2006	MMSE	17

Figure 22 - Un-pivoted table rows

The new file will be automatically saved as 'unpivoted_patient_exams_original' and was renamed manually as 'patient_exams' in our mapping example for display reasons.

The un-pivoting process is executed in a temporary schema of the local PostgreSQL database.

9.2 Change CSV delimiter

The delimiter that is used to separate different values in the input CSV files must be the 'comma' (,) character. If a different delimiter is used, the option to create a new CSV file that is compatible with MIPMap is offered.

After clicking on '**Change CSV delimiter**' under the 'Preprocessing' menu and selecting a CSV file, the user chooses between available options which delimiter is used as a separator in the original file (Figure 23). The user can also choose between double quotes and single quotes being used in the original file as containers. Output options both for delimiters and value containers are fixed with only one option provided for each one.

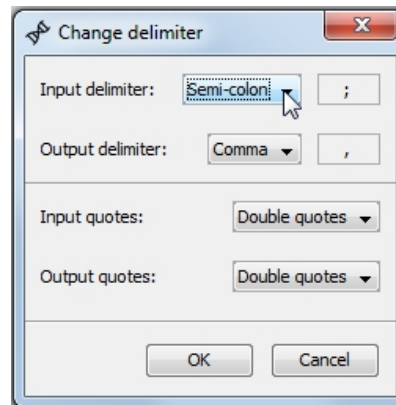


Figure 23 – Selecting input/output delimiters and value containers.

The output is a CSV file containing the same values with commas used as delimiters and double quotes as value containers, saved in the original file directory as the name of the original file concatenated with the string '*_withChangedDelimiter*'.

10. References

- [1] Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995).
- [2] B. Marnette, G. Mecca, P. Papotti - Scalable Data Exchange with Functional Dependencies. In VLDB Conference, 2010
- [3] G. Mecca, P. Papotti, S. Raunich - Core Schema Mappings. In SIGMOD Conference, 2009