# MIPMap Reduced

## *Requirements:*

* Operating System: Windows 7 (or newer) 64-bit, Linux, Mac OS
* RAM: 4GB or higher
* Java version: 8.0
* Latest release of PostgreSQL (version 9.2 +)

## *Before running the aplication:*

The presence of a Postgres Database is essential for MIPMap Reduced to run.

Before running the application you need to create a new or modify the existing PostgreSQL Database properties file (sample included in the application package) providing your credentials and connection information for the Postgres database.

## *Application:*

**- Translate instances and export to csv**

*java -jar* [path to jar]/MIPMapReduced.jar [path to mapping task]/mapping\_task\_simple.xml [path to database config file]/db.properties -csv [path to export folder]

where

* The 1st parameter *<Path To Mapping Task File/Mapping Task File.xml>* is the **absolute** path to a mapping task xml file created with the original MIPMap desktop application. Include the filename itself with its extension.
* The 2nd parameter *<Path To Database Configuration File/Database Configuration File>* is the **absolute** path to the PostgreSQL Database properties file (sample included in the application package). Include the filename itself with its extension.
* The 3rd parameter is the command **-csv** in order to define the exportation method.
* The 4th parameter *<Path To Export Translated Instances>* is the **absolute** path where the user wishes to export the translated instances (in CSV format).
* The 5th parameter is **optional** and its value can be either “true” or “false” (“t/T” and “f/F” are also accepted as input values). If set to true possible instances that violate target Primary Key constraints will be exported as well on a separate folder at the path specified as the 2nd parameter. If set to false instances that violate target Primary Key constraints will be ignored. The **default** value of the parameter is *true*.

Test Cases: ex1

**- Translate instances and export to csv**

java -jar [path to jar]/MIPMapReduced.jar [path to mapping task]/mapping\_task\_simple.xml [path to database config file]/db.properties -db [path to export database config file]/exportdb.properties

where

* The 1st parameter *<Path To Mapping Task File/Mapping Task File.xml>* is the **absolute** path to a mapping task xml file created with the original MIPMap desktop application. Include the filename itself with its extension.
* The 2nd parameter *<Path To Database Configuration File/Database Configuration File>* is the **absolute** path to the PostgreSQL Database properties file (sample included in the application package). Include the filename itself with its extension.
* The 3rd parameter is the command **-db** in order to define the exportation method.
* The 4th parameter *<path to export database config file>* is the **absolute** path of the configuration file of the database in which the translated instances will be exported.
* The 5th parameter is **optional** and its value can be either “true” or “false” (“t/T” and “f/F” are also accepted as input values). If set to true possible instances that violate target Primary Key constraints will be exported as well on a separate folder at the path specified as the 2nd parameter. If set to false instances that violate target Primary Key constraints will be ignored. The **default** value of the parameter is *true*.

Test Cases: ex2

**- GenerateID**

With this functionality can be generated a unique ID for a selected number of columns. In order to operate properly this functionality an input file with the following structure should be given.

commandSource ={csv | db}

sourceInputPath= {[path to csv file]/file.csv | [path to database properties file],[name of the table to extract data]

commandTarget={csv | db}

targetInputPath= {[path to csv file]/file.csv | [path to database properties file],[name of the table to extract data]}

targetColumns=[col11,col22,col33,generated\_id]

functionPerColumn=[split(col1,\_,0),"constant value",col3]

outputFile=[path to csv file]/file.csv

The source and the target can be either CSV or Database. In case of CSV, a path to CSV file must be given. In case of Database, a path to configuration file must be give. Furthermore, if the target is a database, a table name must be specified.

In the field **targetColumns** must be declared the columns which will be produced. The last column is the column which includes the generated id.

In the field **functionPerColumn** is defined the input values which will take the output columns. It supports three type of inputs.

1. Function inputs: An input could be a function output. The split function is the only one that is supported. The split function, takes a column from the source input as first parameter, a delimiter to split as second parameter and the part which will be taken after the function's operation. In this example the input is the column with column name “col1”, the delimiter is the underscore(“\_”) and after the split, the first part of the split is kept.
2. Constant values: A column could have the same constant value in all rows. In this case, a value must be declared inside double quotes.
3. Column name: In case that the user declares a column name, then the values of the source column will be exported to the respective column in the output file.

All the **targetColumns** are matched with the columns in **functionPerColumn** by position. So the **col11** will have the values which will be produced after the split of **col1**

In the field **outputFile** is specified the path and the name of the file which will be exported after the generation of the ID.

\* This functionality should be imported to Desktop MIPMap. It can be either implemented by uploading an input file with the structure which mentioned above or by developed panels appropriate to support this functionality.

Test Cases: ex3

**- Unpivot csv file**

java -jar [path to jar]/MIPMapReduced.jar -unpivot [path to csv]/inputCsv.csv [path to database config file]/db.properties nameOfNewColumn [path to selected columns file]/selectedColumns.txt {-i(gnore) or -u(npivot} [path to altered columns file]/alteredColumns.txt

where

* The 1st parameter is the command **-unpivot**.
* The 2nd parameter *<*path to csv*>* is the **absolute** path to the csv file which is selected for unpivoting.
* The 3rd parameter *<*path to database config file*>* is the **absolute** path to the PostgreSQL database properties file (sample included in the application package). Include the filename itself with its extension.
* The 4th parameter *<*path to selected columns file*>* is the **absolute** path of the file which includes the columns that will be kept in the final csv file. Should be noted that the columns must specified in a line by line way.
* The 5th parameter could be either the command **-i**(gnore) or **-u**(npivot). It defines if the columns which are specified in the alteredColumns file will be ignored or used during the unpivoting functionality.
* The 6th parameter *<*path to altered columns file*>* is the **absolute** path of the file which includes the columns that will be used (if the **-u** command is selected in 5th parameter) or not used (otherwise) in the unpivoting functionality. Should be noted that the columns must specified in a line by line way.

Test Cases: ex4

**- Change csv delimeter**

java -jar [path to jar]/MIPMapReduced.jar -csv\_delimeter [path to csv]/inputCsv.csv {";" | ":" | tab} {single | double}

where

* The 1st parameter is the command **-csv\_delimeter**.
* The 2nd parameter *<*path to csv*>* is the **absolute** path to the csv file which is selected for changing delimeter.
* The 3rd parameter is the current delimeter of the csv file.

The three choices are:

* ";" : If the current csv is separated with semi-colon

":" : If the current csv is separated with colon

tab : If the current csv is separated with tabs

* The 4th parameter is the current quote's style of the csv file.

The two choices are:

* single : If the current csv includes single quotes
* double : If the current csv includes double quotes

Test Cases: ex5