Sii-Mobility

Supporto di Interoperabilità Integrato per i Servizi al Cittadino e alla Pubblica Amministrazione

Trasporti e Mobilità Terrestre, SCN_00112

Deliverable ID: DE4.2a
Titolo: Sistema di acquisizione e aggregazione dati

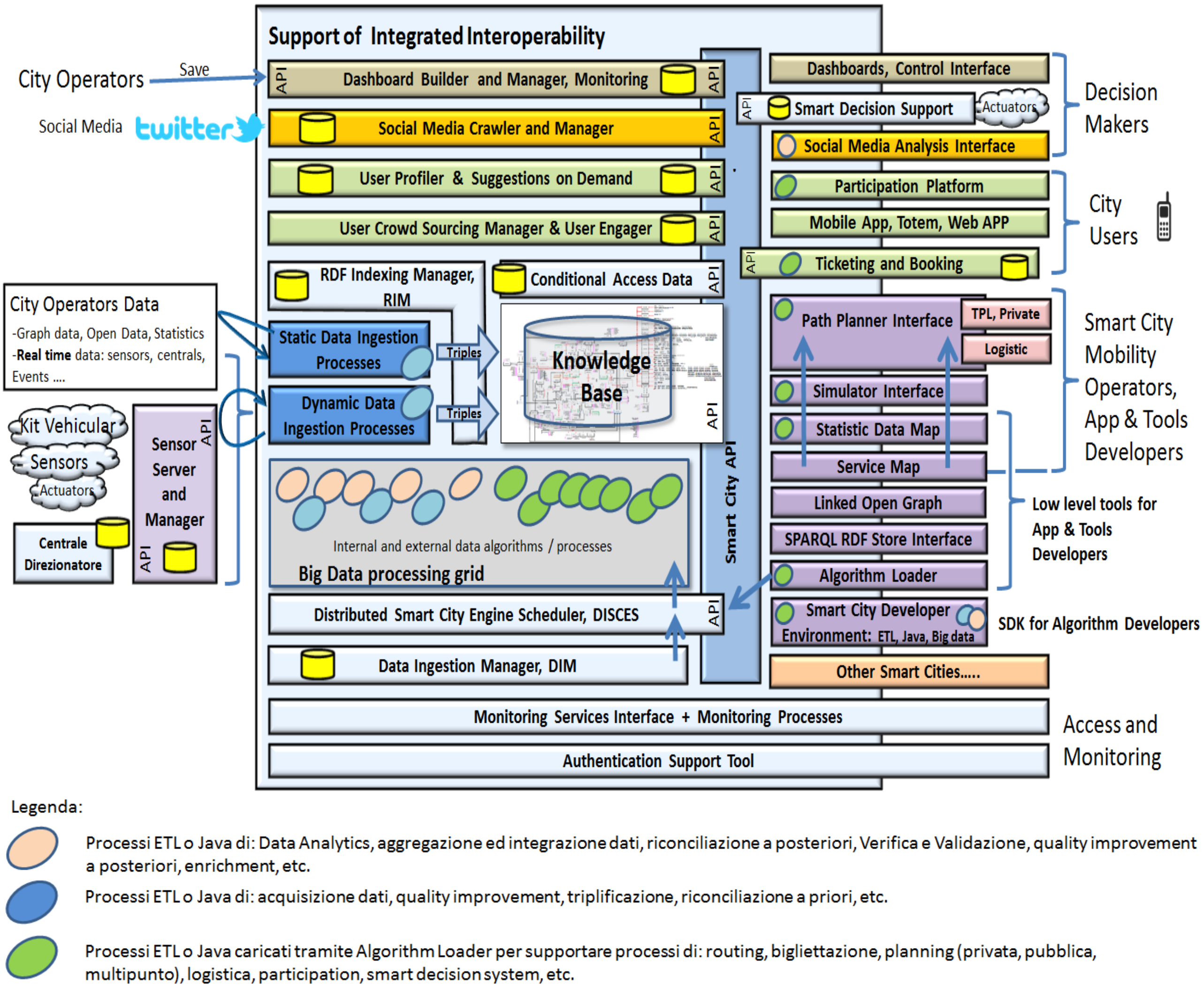
Data corrente	xxxxx
Versione (solo il responsabile puo' cambiare versione	0.1
Stato (draft, final)	Draft
Livello di accesso (solo consorzio, pubblico)	Pubblico
WP	WP4
Natura (report, report e software, report e HW..)	Software e Documentazione
Data di consegna attesa	M12 - Dicembre 2016
Data di consegna effettiva	xxxxxx
Referente primario, coordinatore del documento	UNIFI: DISIT
Contributor	Michela Paolucci, michela.paolucci@unifi.it Paolo Nesi. <Nome> <Cognome> <email>, <Nome> <Cognome> <email>
Coordinatore responsabile del progetto	Paolo Nesi, UNIFI, paolo.nesi@unifi.it

Sommario

1 Executive Summary

Sii-Mobility intende creare una soluzione che possa abilitare un'ampia gamma di servizi al cittadino e commerciali in connessione e integrati con il sistema di mobilità: collezionando dati puntuali e attualizzati in tempo reale da varie fonti; analizzando i flussi di dati con varie tipologie di algoritmi producendo azioni e informazioni tramite applicazioni web e mobili, totem informativi, ecc.; mettendo a disposizione dati elaborati e puntuali, che potranno essere usati da PA, gestori, e imprese per produrre servizi più efficaci ed efficienti, e anche nuovi servizi integrati. Permettendo a PA e PMI di caricare ulteriori algoritmi sul sistema per erogare servizi verso gli utenti finali e verso le PA. Per esempio algoritmi di routing, di valutazione e predizione di condizioni critiche, di ottimizzazione delle risorse, di personalizzazione dei percorsi, di guida connessa, etc.

Nell'architettura del progetto **Sii-Mobility** si possono notare le interfacce per la connessione con altri sistemi di Smart city, con il sistema di mobilità nazionale, la rilevazione dati ambientali, le ordinanze, etc.


Support of Integrated Interoperability
City Operators
Save
Social Media
twitter
API
Dashboard Builder and Manager, Monitoring
Social Media Crawler and Manager
User Profiler & Suggestions on Demand
User Crowd Sourcing Manager & User Engager
RDF Indexing Manager, RIM
Conditional Access Data
City Operators Data
-Graph data, Open Data, Statistics
-Real time data: sensors, centrals, Events ....
Static Data Ingestion Processes
Dynamic Data Ingestion Processes
Triples
Triples
Knowledge Base
Kit Vehicular
Sensors
Actuators
Sensor Server and Manager
Centrale Direzionatore
Internal and external data algorithms / processes
Big Data processing grid
Distributed Smart City Engine Scheduler, DISCES
Data Ingestion Manager, DIM
Smart City API
Dashboards, Control Interface
Smart Decision Support
Actuators
Social Media Analysis Interface
Decision Makers
Participation Platform
Mobile App, Totem, Web APP
Ticketing and Booking
City Users
Path Planner Interface
TPL, Private
Logistic
Simulator Interface
Statistic Data Map
Service Map
Smart City Mobility Operators, App & Tools Developers
Linked Open Graph
SPARQL RDF Store Interface
Algorithm Loader
Low level tools for App & Tools Developers
Smart City Developer Environment: ETL, Java, Big data
SDK for Algorithm Developers
Other Smart Cities.....
Monitoring Services Interface + Monitoring Processes
Authentication Support Tool
Access and Monitoring
Legenda:

● Processi ETL o Java di: Data Analytics, aggregazione ed integrazione dati, riconciliazione a posteriori, Verifica e Validazione, quality improvement a posteriori, enrichment, etc.

● Processi ETL o Java di: acquisizione dati, quality improvement, triplificazione, riconciliazione a priori, etc.

● Processi ETL o Java caricati tramite Algorithm Loader per supportare processi di: routing, bigliettazione, planning (privata, pubblica, multipunto), logistica, participation, smart decision system, etc.

Figura 1: Architettura Sii-Mobility

Le tipologie di dati che devono essere acquisiti sono molteplici e si differenziano per vari aspetti: Provider (PA, Aziende trasporti, PMI, Aziende pubbliche, etc.), tipo di dato (in termini di argomento, ad esempio: mobilità, servizi, etc.), tipo di formato (shp, json, xm, html, etc.), licenza d'uso (Open Data, Private Data, etc.), frequenza di aggiornamento (statici o semi statici, periodici, real-time), etc. La mole di dati che ogni giorno deve essere ingerita genera quindi un problema risolvibile grazie ad una analisi in termini di Big Data.

Scopo di questo Deliverable è quello di descrivere il prototipo del software realizzato per raggiungere gli obiettivi per la fase di acquisizione e aggregazione dati del progetto Sii-Mobility che prevede: implementazione di opportuni processi ETL e di una serie di tool per la gestione degli stessi processi.

2 Introduzione ed obiettivi

Sii-Mobility is a Smart City tool for implementing the city vision, monitoring the city evolution, diffusely providing new services for improving quality of life of city users, for the city economic grow; stimulating city users; since an attractive city is a “city that produces” and in which users are happy and proud.

The new generations of smart city services enable to aggregate heterogeneous data (static and real time data, open and private data), as derived and provided by city, city operators, IOT sensors, social media, and mobile applications with the city user behavior. They are capable to take advantages from multi domain data: mobility, energy, government, welfare, wellness, tourism and culture, environment, weather, education, governmental, etc. Despite to the huge amount of potential services and data accessible on the city; thus, losing for the city the possibility of controlling the data flow evolution and value; and in part also the control, the view. On the contrary sentient cities, should provide a data aggregation platform supporting data analytics and producing user suggestions and engagements on demand for implementing city strategies as well as requests of city operators/stakeholders.

Most of the market open data aggregation tools, just based on ingesting and aggregating open data, are unsuitable for proving smarter and sentient services. Classical open data collection and indexing, browsing and visualization tools are unsatisfactory tools for producing value from data via services, and thus for enabling the construction of smart services and making them sustainable for city and service providers.

Also, solution that harmonize the services among the several operators are unsatisfactory for creating value from data, since data still remain non-interoperable, and the development of new service and tools, too expensive for the operators.

State of the art solutions for smart city have large problems for the (i) lack of data semantic interoperability and reconciliations among the hundreds of data sets, (ii) difficulties in managing real time data flow and putting them in connection with the rest of static data, (iii) lack of support for managing complexity of licensing mechanisms, (iv) lack of tools for social innovation producing strategic suggestions, user engagement, taking into account people and traffic low, origin destination analysis, user flow analysis, etc. (v) lack of easy to use tools for developing mobile and web Apps for the city services and events, (vi) lack of integrated solution for setting up flexible smart city control room dashboard and resilience assessment.

In this document the purpose Is to pose attention on the ingestion/aggregation of the datasets, and provide a set of guidelines to develop the ETL processes necessary to connect the Sii-Mobility data to Knowledge Base in Km4Cit. However, you can find more general details, related to the KM4City tools following this link.

3 Description of the ingestion Architecture

One of the most relevant aspects of the Smart City ingestion tool is the possibility to ingest the data coming from the cities adding, when necessary, details to improve the quality of the data and aggregating them thanks to the Km4City sematic engine. In this way, the datasets are connected each other, can interact and can be viewed as a strong resource for the citizens.

A data aggregation platform for smart city is a very complex system. In this document, we intend to illustrate the ETL processes that are at the basis of the Sii-Mobility framework, developed by DISIT Lab of the University of Florence, Fig.1. This solution is based on a semantic model, Km4City Ontology also developed at DISIT Lab (highlighted in red box). Its main goal is to collect a very large number of public (often released as Open Data) and private datasets for providing integrated services in multiple domains to citizens, developers and Public Administrations.

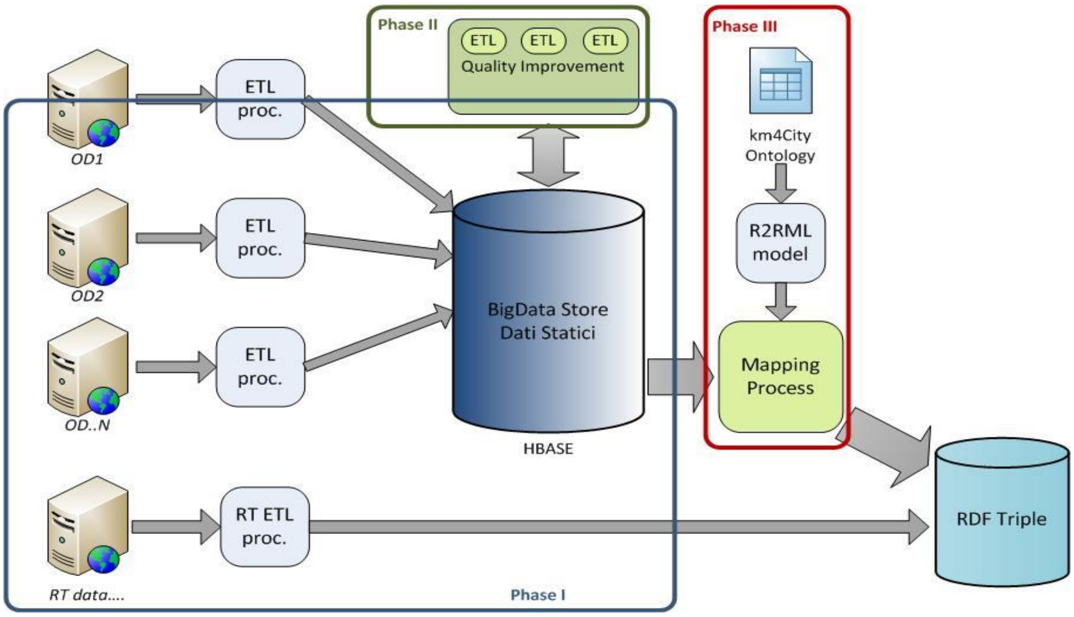


Fig.3.1: Km4City data aggregation platform.

As you can see in Fig.3.1, the datasets collected come from different sources in terms of classification (Point of interest, events, public Transports, traffic flows, etc.), format (csv, json, xml, html, shape, etc.), associated license (Open or Private Data), refresh rate (Static, Periodic, Real Time), etc. and are characterized by high heterogeneity, for this reason it is necessary a process of data analysis and transformation in order to make them interoperable and reusable. The goal in fact is to map the data on a semantic model and store this information into an RDF store, to have data semantically most significant of the raw data ingested.
The main data sources to which we have referred are:

- Website for ***Open Data of Florence Municipality*** (*http://opendata.comune.fi.it/*)
- Website for the ***Open Data of the Tuscany Region*** (*http://dati.toscana.it/*)
- ***Regional Observatory for Mobility and transport*** (*http://www501.regione.toscana.it/osservatoriotrasporti/*)
- ***MIIC*** (Mobility Integration Information Center of the Tuscany Region)
- ***LAMMA Consortium*** *(http://www.lamma.rete.toscana.it/)*

- ***Data crawled*** from website with a list of Italian companies (*http://www.informazione-aziende.it/*)

In most cases these datasets are published as Open Data. This mean they are freely accessible to anyone who wants to use them, which are not constrained by patents or other forms of control that limit the reproduction. The only possible limit related to copyright is to quote the source or release changes and improvements still the same open format. The data provided from the sources listed above are of several kinds and collect information related to mobility, tourist information, weather forecast, green areas, monuments, shops, wifi hot-spots, cultural events, statistical data of public administration etc. Obviously, these datasets are delivered in different formats, the most common are: CSV, XML, KMZ, JSON, PDF, RDF, SHP.

As a design choice, we have divided this information into two categories:

Static and semi-static data: consists of information that is constant in time or that changes with a very low frequency (in the order of a few months). For this reason, the processes that perform the acquisition and the storage of these kinds of information is launched manually. Some examples of data belonging to this category are:

- Road graph of the Tuscany region, which includes about 137,745 single street and 1,432,223 civic numbers.
- Wi-Fi hot-spots, public toilets, bike racks, drinking fountains, car parks position and bus stops in Florence.
- Position and general information related to: monuments, museums, theatres, hotels, pubs, restaurants, sports facilities, government offices, first aid, hospital, emergency etc.

Real Time data: collects data that changes over time at a rate ranging from a few minutes to several hours in a day. In fact, the processes that allow their acquisition are not manual but automated through the use of a scheduler that starts the execution of each of these processes on the basis of the time period defined as configuration parameter. Some examples of data belonging to this category are:

- Car parks occupancy status (updated every 20 minutes).
- Traffic flows detected by the sensors (updated every 2 hours).
- AVM (Automatic Vehicle Monitoring) information for bus routes.
- Weather forecast of all 287 Tuscan municipalities (updated 2 times a day).
- Public Transport Lines (timetables, trips, etc.)
- Events in Florence Municipality (updated 1 time a day)
- Fuel stations in the whole Tuscany
- Status of the triage for the hospitals
- Etc.

4 How to create your own open dataset

The first step is to design the schema of the table that will hold all the data.
The example refer to the data related to facilities of the Univeristy of Florence, just for example.

The **mandatory fields** are:

- class: the main class to which the facility belongs, that allows to connect it to the KM4City ontology ([km4CitySchema])
- categoryENG: subclass of the "class" field
- nameENG: the name of the facilty in english
- latitude: latitude coordinates in WGS84 coordinates system
- longitude: longitude coordinates in WGS84 coordinates system

Other fields that should be added, but that are not mandatory are:

- otherCategoryITA: other possible subdivision or category; it depends on the kind of facility/service your are dealing with
- otherCategoryENG: the translation of the previous entry
- nameITA: the italian name of the facility
- abbreviationITA (shortNameITA): the name's abbreviation in italian
- abbreviationENG (shortNameENG): the transalted abbreviation
- descriptionShortITA: a brief description of the facility
- descriptionShortENG
- descriptionLongITA: a more detailed description
- descriptionLongENG
- phone: this field must contain only one phone number; possible additional numbers must be written in the specific field
- fax: the previous rule applies also to this field
- url: the url of the facilty website; it must start with "http://" or "www"
- email: the same behaviour as for the phone field
- RefPerson: the name of the person in charge of the facility/department
- province
- city
- postalcode: the CAP/ZIP code of the city
- streetaddress: the full name of the street, without the house number
- civic: the house number
- secondPhone: possible additional phone numbers
- secondFax: possible additional fax numbers
- seconEmail: possible additional email addresses
- secondEntrance: possible additional street addresses/secondary entrances
- orario: opening days and hours
- photo: a link to a picture of the facilty

All the entries must be filled up in text format, even phone numbers or postal code. Information about the "class" and "categoryENG" fields can be found at the DISIT Knowledge Model Schema page ([km4CitySchema]). The workflow to fill the latitude and longitude fileds will be explained later in the document. It's strongly recommended to compile all the additional fields, in order to have the most detailed and complete dataset.

4.1 Gather the data

In this example we started from the official site of UNIFI, [UNIFI], in order to have consistent and reliable information. Than we began to browse through the pages to find all the schools, [UNIFISchools], and the related departments, [UNIFIDep]. The school and department names, joined together, will form the "nameENG" entry.
The subdivision of departments in areas (e.g. Biomedical Area, Scientific Area...) is an example of the kind of data that could be inserted in the "otherCategory" field.
At this point we can complete all the table fields. This phase could be different in each case:
sometimes you can find a downloadable dataset that you can just modify accordingly to your needs, sometimes (that's our case) you have to manually copy the information to fill out the table.

4.2 Geocoding

In this phase we'll describe the workflow to fill up the "latitude" and "longitude" fields using a method called geocoding.
Geocoding is the process of converting addresses into geographic coordinates. For this process we used QGis ([QGIS]), an open source GIS software, and the MMQGIS Plugin ([MMQGIS]), a suite of tools to work with vector layers.

4.2.1 Geocoding: Install the Plugin

The firts step is to download and install the plugin. To do this you have to open QGis -> Plugins menu -> Plugin Manager (fig. 4.1).

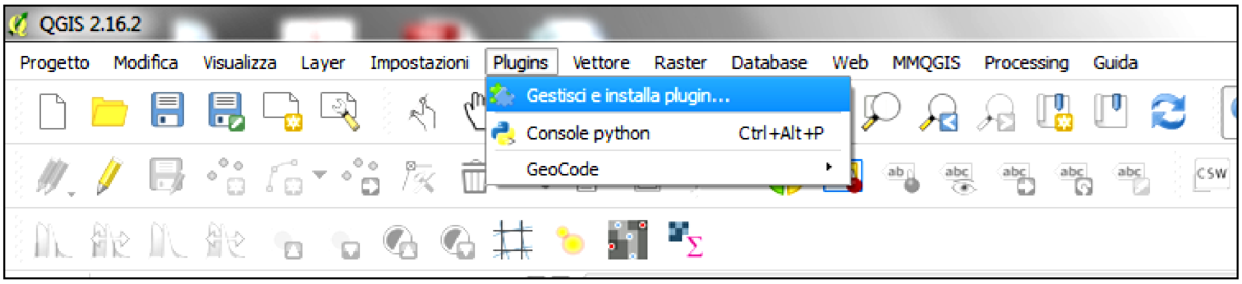
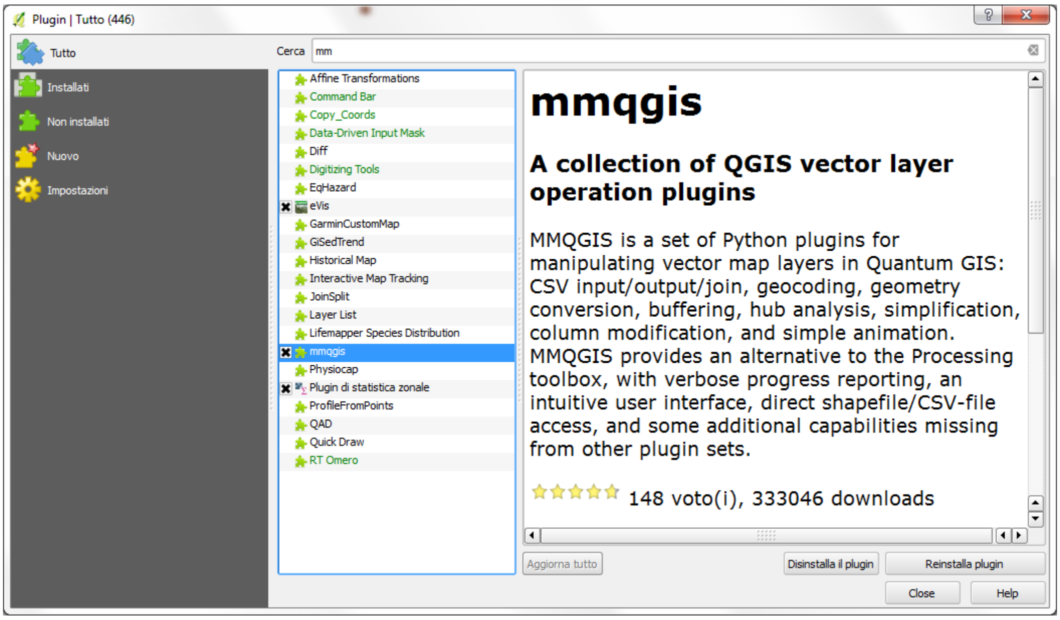


Fig. 4.1: Plugins menu

A window will open, allowing you to type the name of the plugin to find and install it (Fig. 4.2)



mmqgis

A collection of QGIS vector layer operation plugins

MMQGIS is a set of Python plugins for manipulating vector map layers in Quantum GIS: CSV input/output/join, geocoding, geometry conversion, buffering, hub analysis, simplification, column modification, and simple animation. MMQGIS provides an alternative to the Processing toolbox, with verbose progress reporting, an intuitive user interface, direct shapefile/CSV-file access, and some additional capabilities missing from other plugin sets.

☆☆☆☆☆ 148 voto(i), 333046 downloads

Fig. 4.2 Plugin repository window.

4.2.2 Geocoding: Run the tool

After the installation ends you we'll find the MMQGIS menu that allows you to open the geocoding tool (Fig. 4.3)

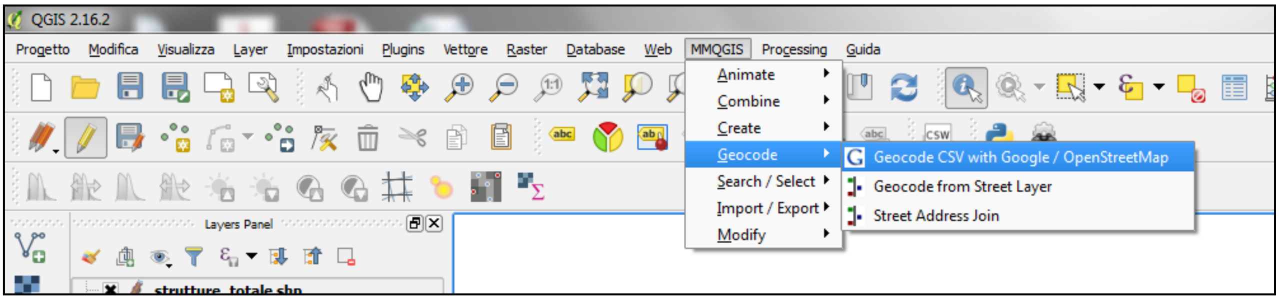
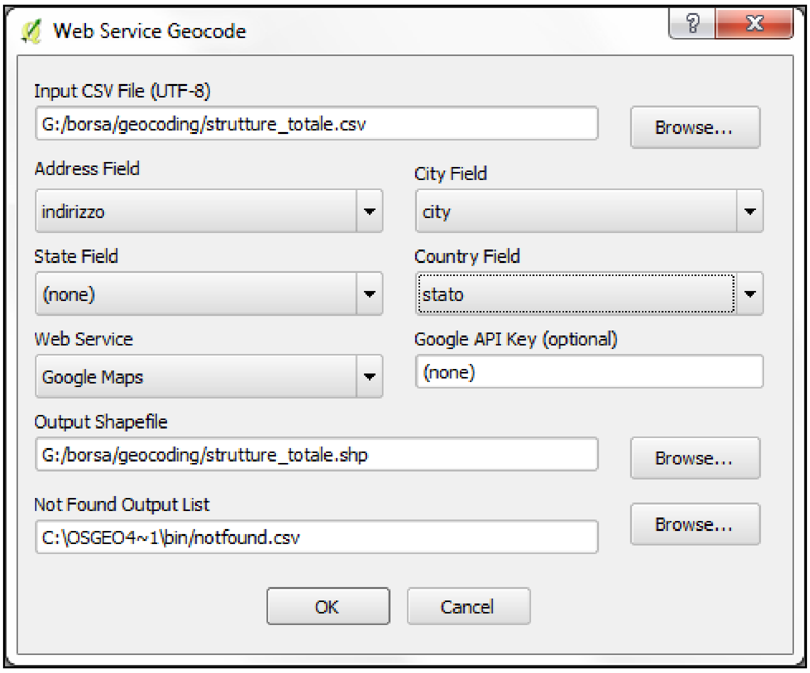


Fig. 4.3 MMQGIS menu

A dialog window will open (Fig. 4.4), with the following parameters to insert (each box refers to a field in your csv file):

- Input CVS File: the input file has to be a csv
- Address Field: must contain the street name and house number (you can simply build this filed concatenating the "streetaddress" and "civic" fileds of your table)
- City Field: e.g. Firenze
- State Field: correspond to our province; we left it empty because this information is more suitable for the american meaning of state (e.g. Louisiana)
- Country Field: the nation



Web Service Geocode

Input CSV File (UTF-8): G:/borsa/geocoding/strutture_totale.csv

Address Field: indirizzo

City Field: city

State Field: (none)

Country Field: stato

Web Service: Google Maps

Google API Key (optional): (none)

Output Shapefile: G:/borsa/geocoding/strutture_totale.shp

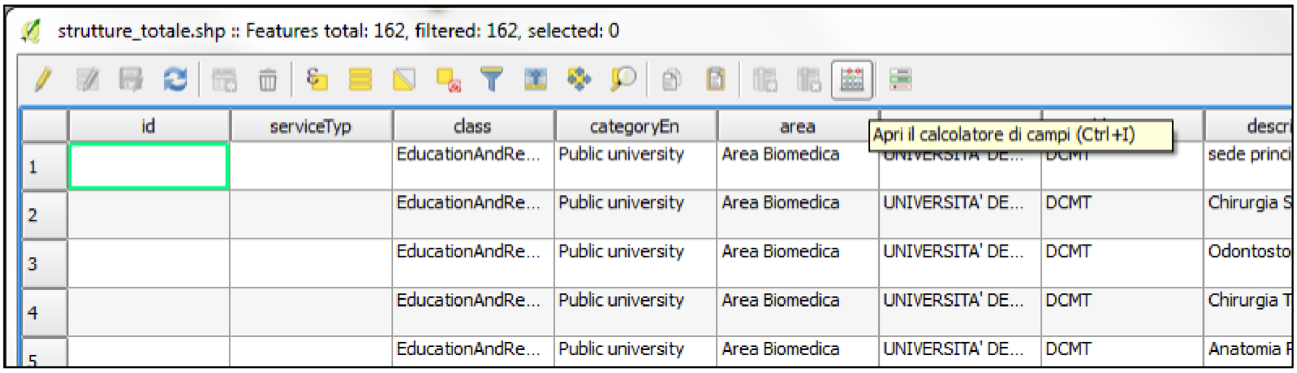
Not Found Output List: C:\OSGEO4~1\bin/notfound.csv

Fig. 4.4 Plugin parameters

You can choose between Google Maps Services ([GOOGLEAPI]) or OpenStreetMap/Nominatim web services. We found that the first is the most suitable to geocode complete addresses.
A "notfound.csv" file is also created, containing all the records that the tool was not able to geocode. The output is shapefile (.shp) that you can open with QGis. Visualizing the shapefile helps you to verify the accuracy of the results. In our example the input table listed 164 records, 161 of which were correctly geocoded, 2 were not found and 1 was placed in the wrong position. We then manually corrected the position of the wrong placed facilities.

4.2.3 Geocoding: Add the coordinates values to the table

To get the coordinates in your table a further step is needed. Load the shapefile in QGis, open its attribute table (Fig. 4.5) and open the field calculator (Fig. 4.6)

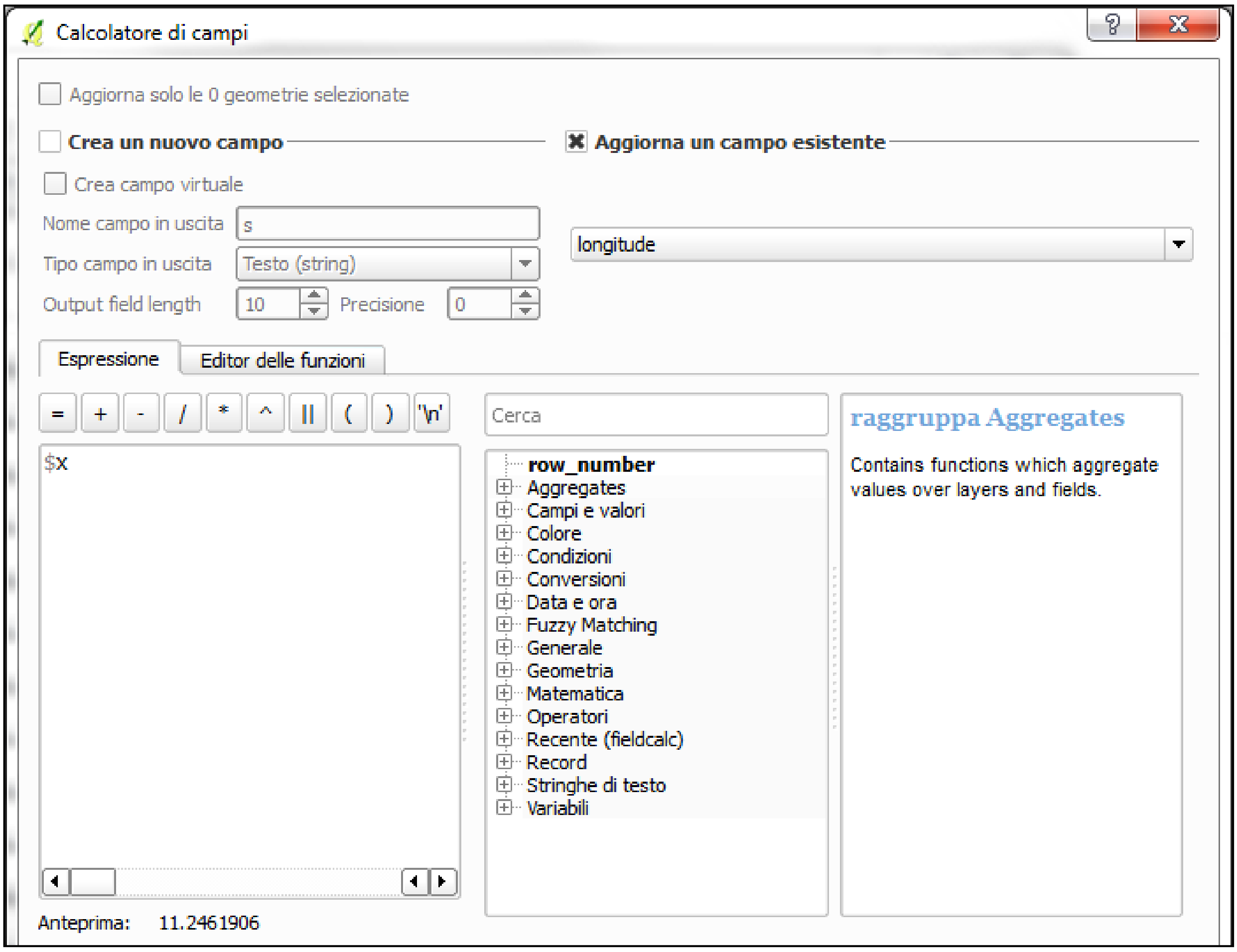


strutture_totale.shp :: Features total: 162, filtered: 162, selected: 0

Apri il calcolatore di campi (Ctrl+I)

	id	serviceTyp	class	categoryEn	area			descri
1			EducationAndRe...	Public university	Area Biomedica	UNIVERSITA' DE...	DCMT	sede princi
2			EducationAndRe...	Public university	Area Biomedica	UNIVERSITA' DE...	DCMT	Chirurgia S
3			EducationAndRe...	Public university	Area Biomedica	UNIVERSITA' DE...	DCMT	Odontosto
4			EducationAndRe...	Public university	Area Biomedica	UNIVERSITA' DE...	DCMT	Chirurgia T
5			EducationAndRe...	Public university	Area Biomedica	UNIVERSITA' DE...	DCMT	Anatomia F

Fig. 4.5 Shape File attribute table.



Calcolatore di campi

□ Aggiorna solo le 0 geometrie selezionate

□ **Crea un nuovo campo**

□ Crea campo virtuale

Nome campo in uscita: s

Tipo campo in uscita: Testo (string)

Output field length: 10 Precisione: 0

☒ **Aggiorna un campo esistente**

longitude

$x

Anteprima: 11.2461906

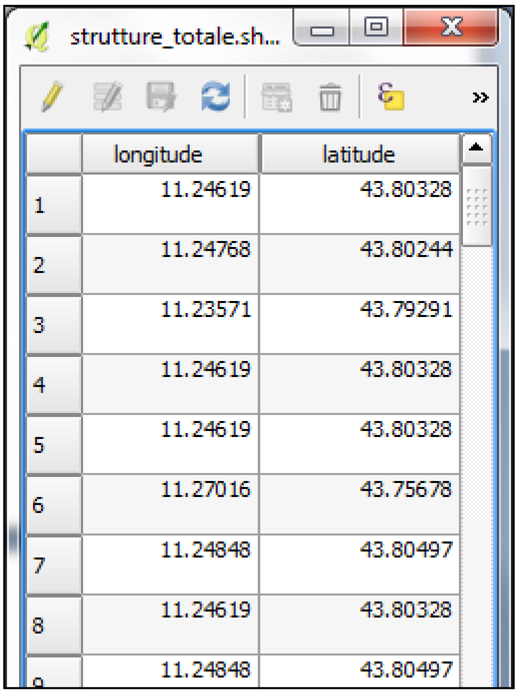
raggruppa Aggregates

Contains functions which aggregate values over layers and fields.

Fig. 4.6 The Field Calculator window

As shown in the above picture you can update an existing field (created in the first step) with a new value automatically computed.

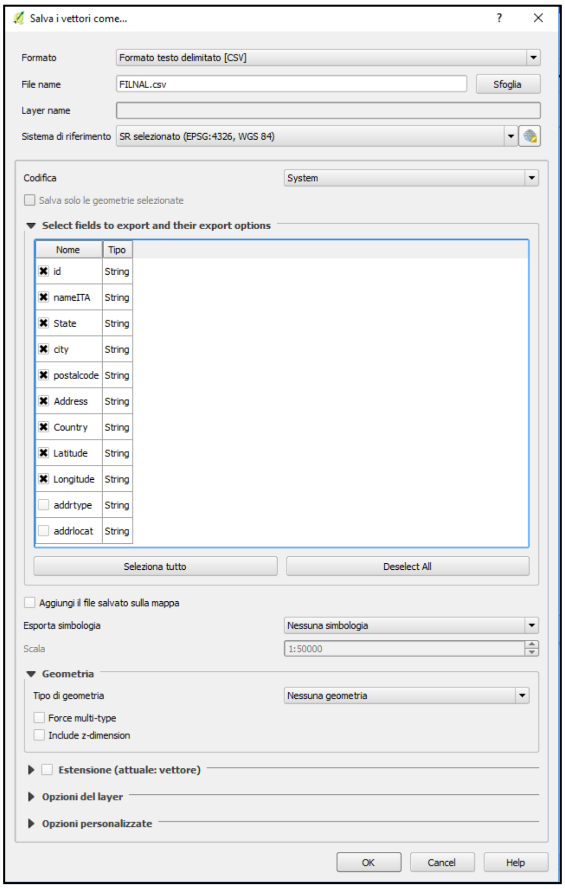
The expression to be written in the expression tab of the window in order to fill the "longitude" field of your table with the real longitude value is "$x", while for the latitude value is "$y". Fig. 4.7 shows the output columns.



strutture_totale.sh...

	longitude	latitude
1	11.24619	43.80328
2	11.24768	43.80244
3	11.23571	43.79291
4	11.24619	43.80328
5	11.24619	43.80328
6	11.27016	43.75678
7	11.24848	43.80497
8	11.24619	43.80328
9	11.24848	43.80497

Fig. 4.7 Longitude and latitude columns



Salva i vettori come...

Formato: Formato testo delimitato [CSV]
File name: FILNAL.csv
Layer name:
Sistema di riferimento: SR selezionato (EPSG:4326, WGS 84)

Codifica: System
☐ Salva solo le geometrie selezionate

Select fields to export and their export options

Nome	Tipo
☒ id	String
☒ nameITA	String
☒ State	String
☒ city	String
☒ postalcode	String
☒ Address	String
☒ Country	String
☒ Latitude	String
☒ Longitude	String
☐ addrtype	String
☐ addrlocat	String

☐ Aggiungi il file salvato sulla mappa
Esporta simbologia: Nessuna simbologia
Scala: 1:50000

Geometria
Tipo di geometria: Nessuna geometria
☐ Force multi-type
☐ Include z-dimension

Fig. 4.8 Saving as a csv.

Last step is to export the table in the desired format by right-clicking the layer in QGis and saving as a CSV or Excel file (Select the correct fields and Geometry, Fig. 4.8).
The obtained file related to the facilities of the University of Florence is available in the Km4City Data Ingestion Guidelines and ETL Examples web page ([ETLGuide]).

5 ETL Guidelines

5.1 ETL different Phases

A set of ETL processes can be created and used to collect these different types of data. These processes are realized considering the following main Phases, detailed in Fig.5.2:

- Phase I: ***Data Ingestion (mandatory)***
  - These ETL processes make calls to web services authenticated or not, and store the raw files in the disk according to a file system with the following structure:
    - **Sources/Year_Month/Day/Hour/MinSec.** In this way, it is possible to keep track of multiple versions of the raw data if they suffer changes or improvements over time. It was also designed a mechanism that compares the downloaded files with the latest version on the disk storing it only when changes have been detected
  - The ETL transformations are performed manually for static data, while for real-time data, processes are launched automatically by a Process Scheduler according to the repetition interval present into a MySQL table (process_manager2, table containing the information characterizing the processes).
  - Due to the large amount of data to compute, and whereas the presence of real-time data leads to an increase of the repository size over the time, it was decided to store the data collected on a NoSQL Database (suitable for Big Data problems), specifically the choice fell on HBase [HBase].
- Phase II: ***Quality Improvement (its presence depends on the quality of the dataset)***
  - This is still a series of ETL transformations that aim to increase the accuracy and consistency of information through a standard format identified for each specific field.
  - At the end of these operations the data are stored again in a new HBase table.
- Phase III: ***Triplification (mandatory)***
  - a data Integration Tool ([Karma]) it is used to define several R2RML models with the aim to map the data stored on a SQL database on a semantic model. In this case, we refer to the semantic model defined by the KM4City Ontology.
  - These R2RML models, are used in the ETL to generate the RDF triples that will be indexed and uploaded to the RDF store (Virtuoso).
  - Periodically a task of re-indexing is executed to index new additional content

After these phases, a set of queries can be done to verify the completeness and consistency of the RDF repository, after new data integration and re-indexing operation. At the end the triples stored can be used to produce in short time web and mobile applications, accessing the data through a SPARQL endpoint.

Relation among kind of datasets and ETL Phases:

- ***Static and semi-static ETL processes can be:***
  - Data Ingestion
  - Quality Improvement (facultative)
  - Triplification
- ***Real Time data:***
  - Data Ingestion
  - Triplification

5.2 Download the Virtual machine

To proceed with this guide, you should download and install the Virtual Machine in which you can find all the instruments we will use to realize the ETL ([Guide_VM]).
The VM contains a development system prepared with the following tools:

- Oracle Java 7 JDK (requisito per Penthao Data Integration e per Apache HBase, [Oracle], [Ubuntu])
- Penthao Data Integration (PDI) ver. 5.0.1 (tool ETL, [PDI])
  - Start from the folder “data-integration” with the command: "**./spoon.sh .**" *(shell)*
- XAMPP (Database MySQL, [XAMPP])
  - Start with the command: “***sudo /opt/lampp/lampp start”*** *(shell)*
  - Stop with the command: “***sudo /opt/lampp/lampp stop”*** *(shell)*
  - Access from PDI with **username=disit | password=Ubuntu** *(shell)*
- Apache HBase ver. 0.90.5 (Database NoSQL, [HBase])
  - Start from the folder “/bin” with the command: ***start-hbase.sh*** *(shell)*
  - Stop from the folder “/bin” with the command:***stop-hbase.sh*** *(shell)*
  - Verify the execution with the command “***jps”*** *(shell)*
  - Verify the execution with the **http://localhost:60010/master.jsp *(browser)***
- H-Rider ver. 1.0.3.0 (to manage data on HBase, [HRider])
- Karma data integration ver. 2.024 (for the Triplification Phase, [Karma])
  - Start from the folder “/programs/Web-Karma-master/karma-web” with the command ***mvn -Djetty.port=9999 jetty:run***
  - Access from web ***http://localhost:9999***.

5.3 Details on ETL

Both the data ingested in input and those produced as output, must be saved following a specific nomenclature:

- Input (where the sources must be saved)
  - /Sources/*KM4CitySourceCategory(*)*/ProcessName/Year_mounth/Day/Hour/MinutesSeconds/filename.xxx
- ETL code (where the process must be saved):
  - [Phase I]: /Trasformazioni/NomeTrasformazione/**Ingestion**/...
  - [Phase II]: /Trasformazioni/NomeTrasformazione/**QualityImprovement**/...
  - [Phase III]: /Trasformazioni/NomeTrasformazione/**Triplification**/...
- Final output (triples produced in Phase III):
  - */Triples/KM4CityOutputCategory(**)/ProcessName/*Year_mount/Day/Hour/MinutesSeconds/filename.n3

5.3.1.1 *NOTES on KM4CityCategories:*

- *Main KM4CitySourceCategory(*):* (in the ETL process, it is necessary specify one of these categories)
  - GeolocatedObj
  - Services
  - Other (if necessary and in accordance with the Km4City ontology)
  - …
- *Main KM4CityOutputCategory(**):*
  - GeolocatedObject
  - Servizi

- Sensori
- Other (if necessary and in accordance with the Km4City ontology)
- …

5.4 ETL Sample

This document details how some ETL processes are realized, using the Pentaho - Kettle tools for data integration (Open source tool), in particularly the Spoon Open source tool ([Penthao]).

<u>Spoon Introduction:</u>

1. In Spoon, you have different kinds of blocks:
    a. Elementary blocks, classified basing on their use, such as:
        i. File management
        ii. Repository
        iii. Big Data
        iv. Scripting
        v. …

    b. Transformations:
        i. Can contain elementary blocks and can be reused
    c. Job: The main Process is a job, has a start ('start' ▶ block) and one or many ends ('success' block ✓) . A job can contain:
        i. Other Jobs
        ii. Transformations
        iii. Elementary blocks

5.4.1.1 Example: Pharmacies in Tuscany

This dataset can be accessed on the Open Data Portal of the Tuscan Region, http://dati.toscana.it (Specific Link: http://www301.regione.toscana.it/bancadati/farmacie/index.xml). The KM4City ETL process is available here: http://www.disit.org/drupal/?q=node/6975

5.4.1.1.1 Phase I: Data Ingestion (spoon Job)

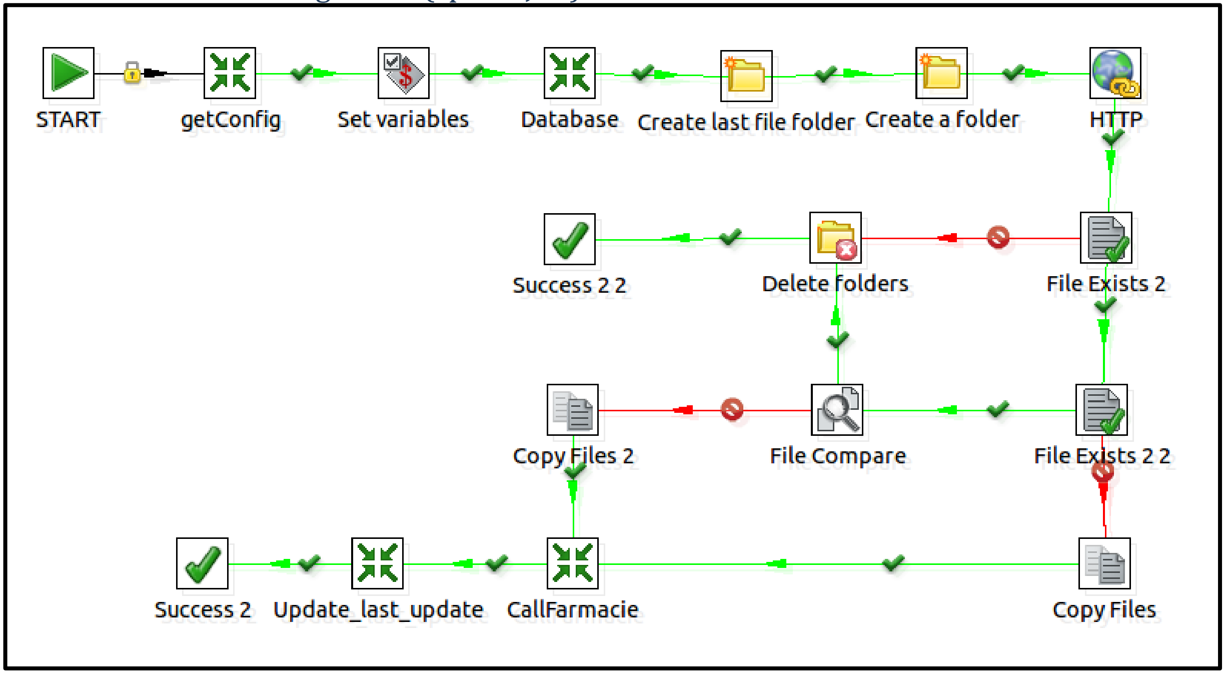


Fig. 5.2: Pharmacies in Tuscany - Data Ingestion phase.

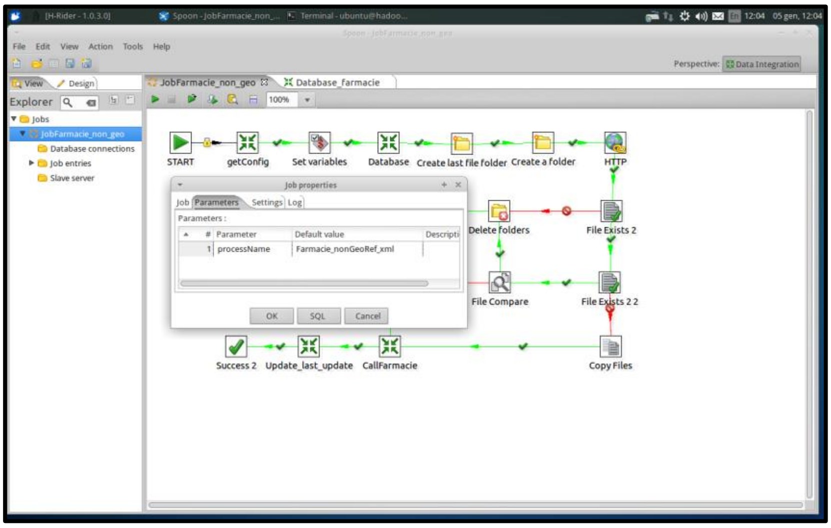
The Data Ingestion Phase is composed of the following actions (referring to Fig. 5.2):

- **Process parameters** (click on the green triangle to start the Process/Spoon Job):
  - In some processes (you find the 'processName' parameter that must be specified at the execution time and saved in the 'process_manager_2' Mysql table)
- **Configuration blocks:** blocks containing the configuration of the variables that can be used in the processes:
  - 'getConfig' and 'Set Variables' (Spoon Transformations)
- **MySQL Database Connection** ('process_manager_2' table):
  - 'Database' (Spoon Transformation)
- **Blocks for the Creation of the folder** into which the dataset will be uploaded (according with the KM4City nomenclature previously described):
  - 'Create last file folder'
  - 'Create a folder'
- **Connection with the web server** (that provides the dataset):
  - 'HTTP'
- Set of simple blocks to establish if some differences from the last upload exist:
  - 'File exist..' | 'Delete folders' | 'Success .. ' | 'File Compare' | 'Copy files'
- Block to: i) get data coming from the file uploaded and ii) insert them on HBase:
  - 'CallFarmacie'
- Block for the final connection to the Mysql database (the same 'process_manager_2' table) in order to update the info related to the process status (particularly relevant is the field 'actualDate' that registers the date in which the last download of the dataset has been realized):
  - Update_last_update
- Blocks to exit from the process:
  - 'Success..' (the output is '1' if the ETL process ends with success)

Expanding the jobs and the transformations, it is possible to describe more details

PROCESS PARAMETERS:

- Spoon windows > 'View Tab' > Jobs > click on the Job name > 'Parameters' tab:
- Here it is possible to put the 'Main job' input parameter and its Default value
- In the example you find:
    - Parameter name: *processName*
    - Parameter default value: Farmacie_non_georef
    - Note that in our convention this parameter corresponds to the field 'process' of the
- previously described MySQL table 'process_manager2')
    -



Parameters :

#	Parameter	Default value	Descripti
1	processName	Farmacie_nonGeoRef_xml	

Fig. 5.3: 'Main job' input parameter

CONFIGURATION BLOCKS:

Fig. 5.4 > Expanding the ***getConfig*** Transformation (click on right mouse > Open referenced Object > Transformation)

- *getConfig* Transformation (Fig. 5.4 e 5.5 - NOTE: it MUST be present in each process)
    - Click on *getConfig* (right button of the mouse > Open referenced Object > Transformation)
    - You will see the blocks:
        - Text file input
        - Set Variables

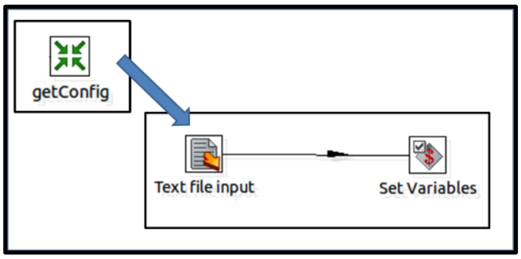
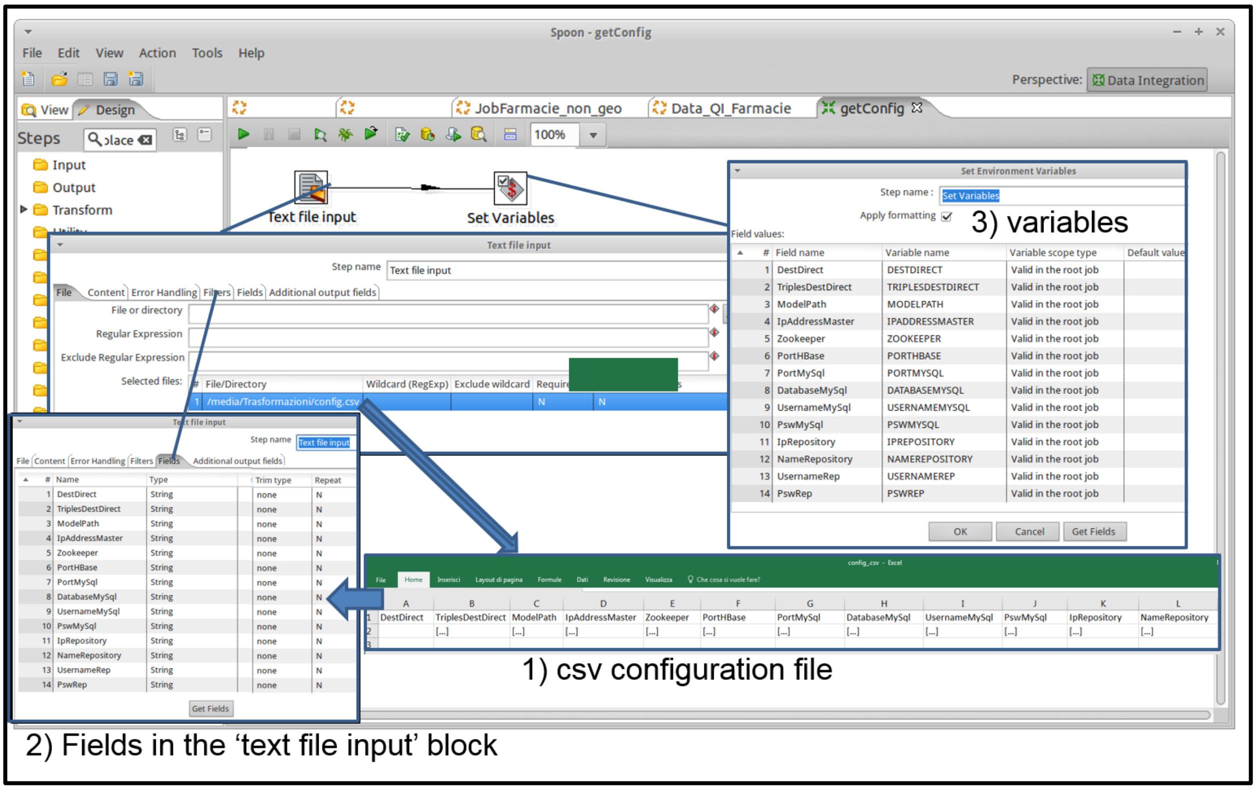


Fig. 5.4: getConfig Transformation



Set Environment Variables

Step name : Set Variables

Apply formatting ☑

Field values:

#	Field name	Variable name	Variable scope type	Default value
1	DestDirect	DESTDIRECT	Valid in the root job	
2	TriplesDestDirect	TRIPLESDESTDIRECT	Valid in the root job	
3	ModelPath	MODELPATH	Valid in the root job	
4	IpAddressMaster	IPADDRESSMASTER	Valid in the root job	
5	Zookeeper	ZOOKEEPER	Valid in the root job	
6	PortHBase	PORTHBASE	Valid in the root job	
7	PortMySql	PORTMYSQL	Valid in the root job	
8	DatabaseMySql	DATABASEMYSQL	Valid in the root job	
9	UsernameMySql	USERNAMEMYSQL	Valid in the root job	
10	PswMySql	PSWMYSQL	Valid in the root job	
11	IpRepository	IPREPOSITORY	Valid in the root job	
12	NameRepository	NAMEREPOSITORY	Valid in the root job	
13	UsernameRep	USERNAMEREP	Valid in the root job	
14	PswRep	PSWREP	Valid in the root job	

OK Cancel Get Fields

Text file input

Step name Text file input

File | Content | Error Handling | Filters | Fields | Additional output fields

File or directory

Regular Expression

Exclude Regular Expression

Selected files:

#	File/Directory	Wildcard (RegExp)	Exclude wildcard	Required
1	/media/Trasformazioni/config.csv			N	N

Text file input

Step name Text file input

File | Content | Error Handling | Filters | Fields | Additional output fields

#	Name	Type	Trim type	Repeat
1	DestDirect	String	none	N
2	TriplesDestDirect	String	none	N
3	ModelPath	String	none	N
4	IpAddressMaster	String	none	N
5	Zookeeper	String	none	N
6	PortHBase	String	none	N
7	PortMySql	String	none	N
8	DatabaseMySql	String	none	N
9	UsernameMySql	String	none	N
10	PswMySql	String	none	N
11	IpRepository	String	none	N
12	NameRepository	String	none	N
13	UsernameRep	String	none	N
14	PswRep	String	none	N

Get Fields

config_.csv - Excel

File Home Inserisci Layout di pagina Formule Dati Revisione Visualizza Che cosa si vuole fare?

	A	B	C	D	E	F	G	H	I	J	K	L
1	DestDirect	TriplesDestDirect	ModelPath	IpAddressMaster	Zookeeper	PortHBase	PortMySql	DatabaseMySql	UsernameMySql	PswMySql	IpRepository	NameRepository
2		[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]	[...]
3												

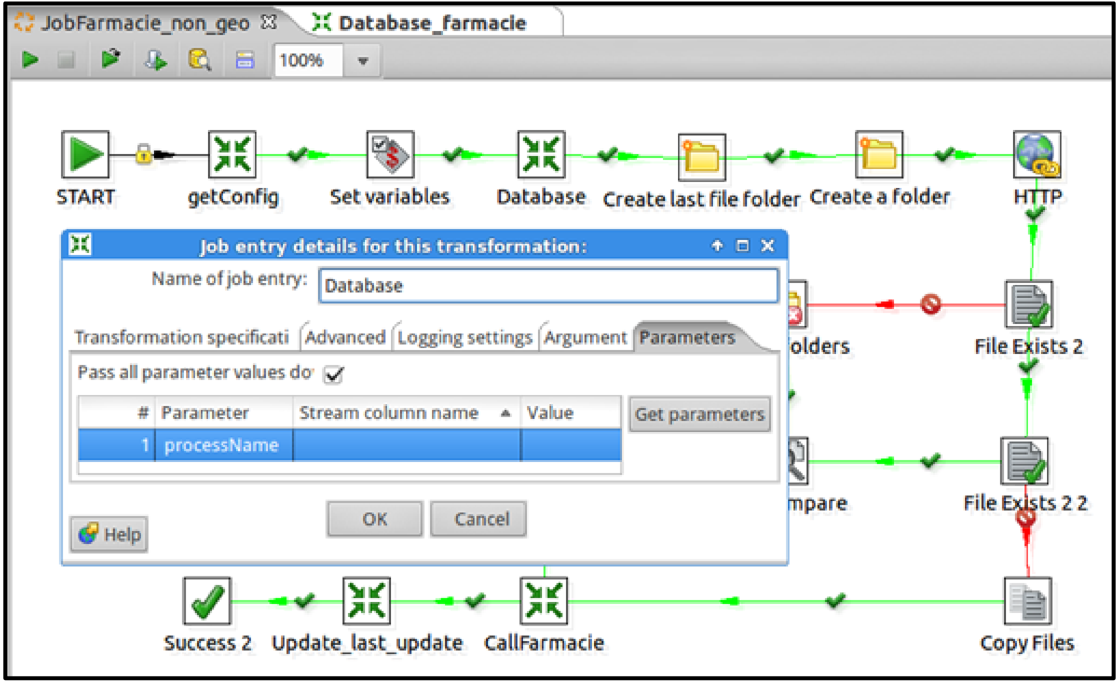
Fig. 5.5: getConfig Transformation - Details.

Looking at Fig.5.5 it is possible to see the following flow:

1. A csv configuration file is available in an external directory (this file can be used by all the transformation created and it is common to all the ETL processes created)
2. The *Text file input* block, reads the data from the csv and takes the fields and values present in it (visible in the tab 'fields' if you click on 'get fields')
3. The 'Set Variables' block transforms the fields (coming from the csv, point 2) in variables that can be used in Spoon (if you want to use the variables in the whole Job, choose the selection 'Valid in the root job').
4. NOTE that the variables so defined, can be reused in the whole process thanks to the following convention:
   - Example: variable = 'DESTDIRECT' | variable value = **${**DESTDIRECT**}**

MYSQL DATABASE:

Fig. 5.6 > Expanding the *Database* Transformation (click on: right mouse > Open referenced Object > Transformation)

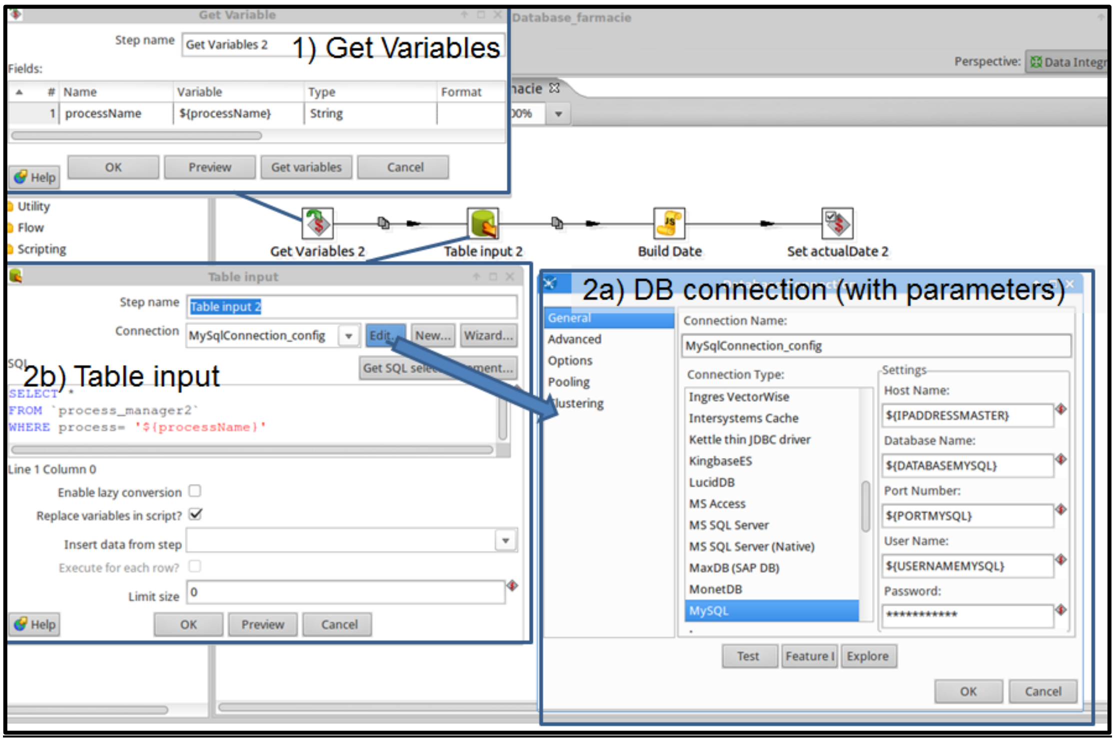


Name of job entry: Database

Pass all parameter values do ☑

#	Parameter	Stream column name	Value
1	processName		

Fig. 5.6a: Database Transformation - Details.



Step name: Get Variables 2

Fields:

#	Name	Variable	Type	Format
1	processName	${processName}	String	

Step name: Table input 2

Connection: MySqlConnection_config

SQL

```
SELECT *
FROM `process_manager2`
WHERE process= '${processName}'
```

Line 1 Column 0

Enable lazy conversion ☐

Replace variables in script? ☑

Insert data from step

Execute for each row? ☐

Limit size: 0

Connection Name: MySqlConnection_config

Connection Type: Ingres VectorWise, Intersystems Cache, Kettle thin JDBC driver, KingbaseES, LucidDB, MS Access, MS SQL Server, MS SQL Server (Native), MaxDB (SAP DB), MonetDB, MySQL (selected)

Settings

Host Name: ${IPADDRESSMASTER}

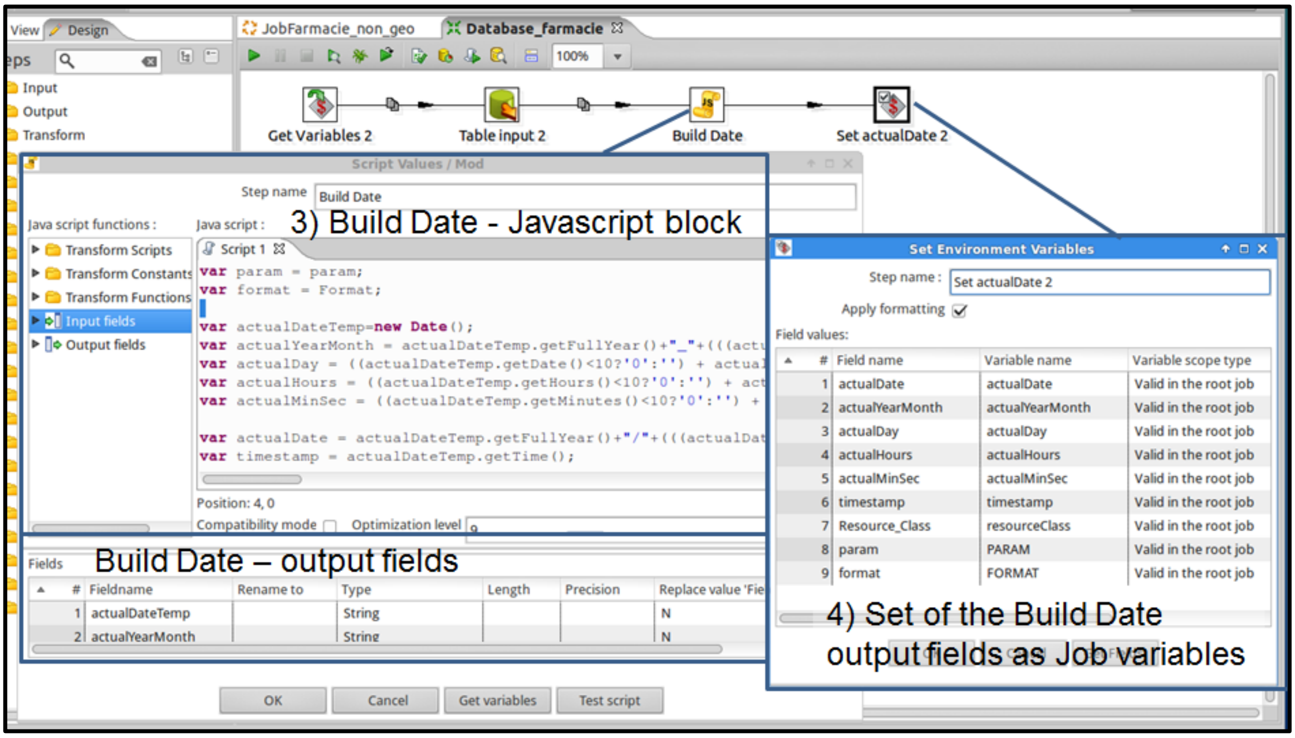
Database Name: ${DATABASEMYSQL}

Port Number: ${PORTMYSQL}

User Name: ${USERNAMEMYSQL}

Password: ***********

Fig.5.6b: Database Transformation - Details.



3) Build Date - Javascript block

Step name: Build Date

```
var param = param;
var format = Format;

var actualDateTemp=new Date();
var actualYearMonth = actualDateTemp.getFullYear()+"_"+(((actu
var actualDay = ((actualDateTemp.getDate()<10?'0':'') + actual
var actualHours = ((actualDateTemp.getHours()<10?'0':'') + act
var actualMinSec = ((actualDateTemp.getMinutes()<10?'0':'') +

var actualDate = actualDateTemp.getFullYear()+"/"+(((actualDat
var timestamp = actualDateTemp.getTime();
```

Position: 4, 0

Compatibility mode ☐ Optimization level 9

Build Date – output fields

Fields

#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fie
1	actualDateTemp		String			N
2	actualYearMonth		String			N

4) Set of the Build Date output fields as Job variables

Step name : Set actualDate 2

Apply formatting ☑

Field values:

#	Field name	Variable name	Variable scope type
1	actualDate	actualDate	Valid in the root job
2	actualYearMonth	actualYearMonth	Valid in the root job
3	actualDay	actualDay	Valid in the root job
4	actualHours	actualHours	Valid in the root job
5	actualMinSec	actualMinSec	Valid in the root job
6	timestamp	timestamp	Valid in the root job
7	Resource_Class	resourceClass	Valid in the root job
8	param	PARAM	Valid in the root job
9	format	FORMAT	Valid in the root job

Fig. 5.6C: Database Transformation - Details.

Looking at Fig. 5.6a) 5.6b) 5.6c) it is possible to see the following flow (NOTE: this transformation MUST be present in each process):

1. (main job > double click on left mouse > Tab Parameters) Transformation edit/input definition/etc.:
    - the input parameter of the main job (*processName*), is passed as input to this *Database* Transformation
2. Get Variables:
    - The block sets the input parameter as a variable that can be used in this transformation (see the Table input)
3. Table input, throught this block it is possible to:
    - Set the Database connection (click on 'Edit'):
        i. Here it is possible to insert the data necessary to establish the database connection (in this case ia MySQL connection). The settings:
            1. Host Name (es: 1) concrete: localhost; 2) with the parameters contained in the csv configuration file: ${IPADDRESSMASTER})
            2. Database Name (es: 1) concrete: Mydatabase; 2) parameters: ${IPADDRESSMASTER})
            3. Port Number: (es: 1) concrete: 3306 2) parameters: ${PORTMYSQL})
            4. User Name: (es: 1) concrete: root; 2) parameters: ${USERNAMEMYSQL})
            5. Password: (es: 1) concrete: root; 2) parameters: ${PSWMYSQL})

Note that: the parameters are fundamental once the transformation is fully operational: to get it started automatically (in a separate scheduler) thanks to the use of parameters. While in the TESTING phase, you must use the concrete modality

- Make Queries in the Database:
    i. Query Example:
        - 1) SELECT * FROM `process_manager2` WHERE process= 'Farmacie_nonGeoRef_xml'
        - 2) parameters concrete: “SELECT * FROM `process_manager2` WHERE process= '${processName}'

    ii. If you want a preview of the output (useful to check the query and to view the fields and their values, **that are also the output fields of the Table input block**), you can press on the ‘Preview’ button

<u>Note that:</u> also in this case, the parameters are fundamental once the transformation is fully operational: to get it started automatically (in a separate scheduler) thanks to the use of parameters. While in the TESTING phase, you must use the concrete modality

4. Build date Block (JavaScript block):
    - It has as input the output fields of the table input block
    - It does a set of elaborations to produce the following fields in output:
        i. param (received as input, because is one of the output fields of the Table input block. In fact, ‘param’ is one of the columns present in the ‘program_manager2’ table)
        ii. format (received as input, because is one of the output fields of the Table input block. In fact, ‘format’ is one of the columns present in the ‘program_manager2’ table)
        iii. actualDateTemp
        iv. actualYearMonth
        v. actualDay
        vi. actualHours
        vii. actualMinSec
        viii. actualDate
        ix. timestamp
5. Set of Actual Date (Set Variables block):
    - This block convert the fields in variables that can be used in the Main job (Variable scope type = ‘Valid in the root job’)

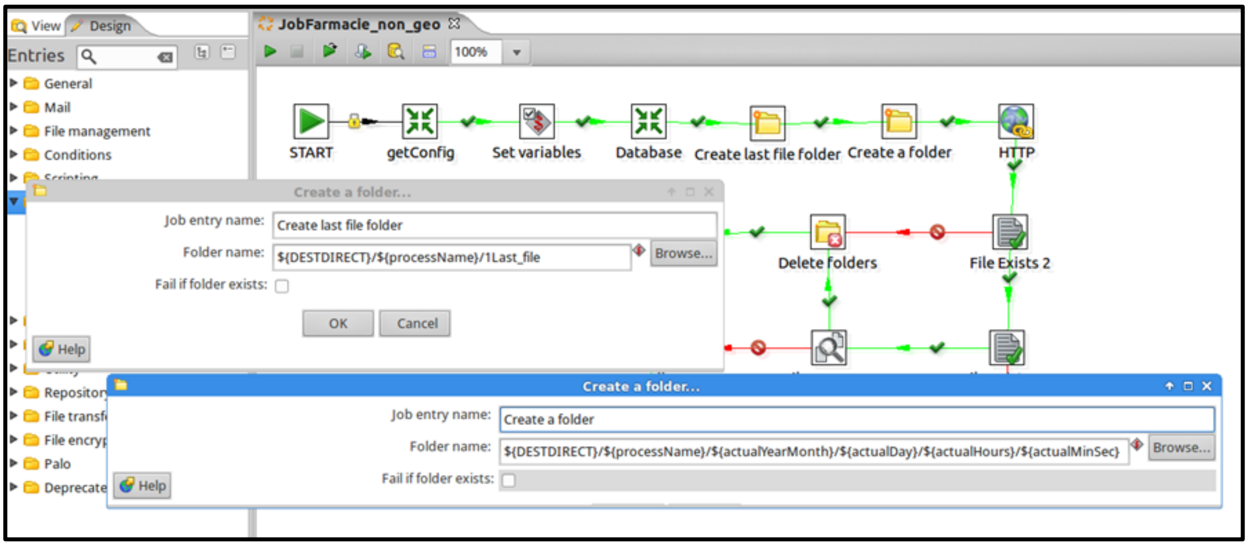
<u>BLOCKS FOR THE CREATION OF THE FOLDER INTO WHICH THE DATASET WILL BE UPLOADED/STORED</u>

Fig. 5.7 > Create_Last_file_folder AND Create a Folder

- Create_Last_file_folder: this block creates the folder in which the last file uploaded will be contained:
  Example: ${DESTDIRECT}/${processName}/1Last_file **|** In this case the path is written using the parameters (according with the Km4City nomenclature) and depends from:
    - o DESTDIRECT (one of the variables contained in the csv configuration file)
    - o processName (the input parameter of the Main Job)
- Create a Folder: this block creates the folder in which the file uploaded is stored and it is different each time the Main process it is launched to take traces of the data changes and the actions performed. In fact, it depends on the time in which the Main job is launched:

Example:
${DESTDIRECT}/${processName}/${actualYearMonth}/${actualDay}/${actualHours}/${actualMinSec}

- DESTDIRECT (one of the variables contained in the csv configuration file)
- processName (the input parameter of the Main Job)
- actualYearMonth, actualDay, actualHours, actualMinSec (variables calculated in the Database > Build date Block, Fig.5.6c)



Create a folder...

Job entry name: Create last file folder

Folder name: ${DESTDIRECT}/${processName}/1Last_file

Fail if folder exists: □

Create a folder...

Job entry name: Create a folder

Folder name: ${DESTDIRECT}/${processName}/${actualYearMonth}/${actualDay}/${actualHours}/${actualMinSec}

Fail if folder exists: □

Fig. 5.7: Folder Creation, basing on theKm4City nomenclature - Details.

CONNECTION WITH THE WEB SERVER

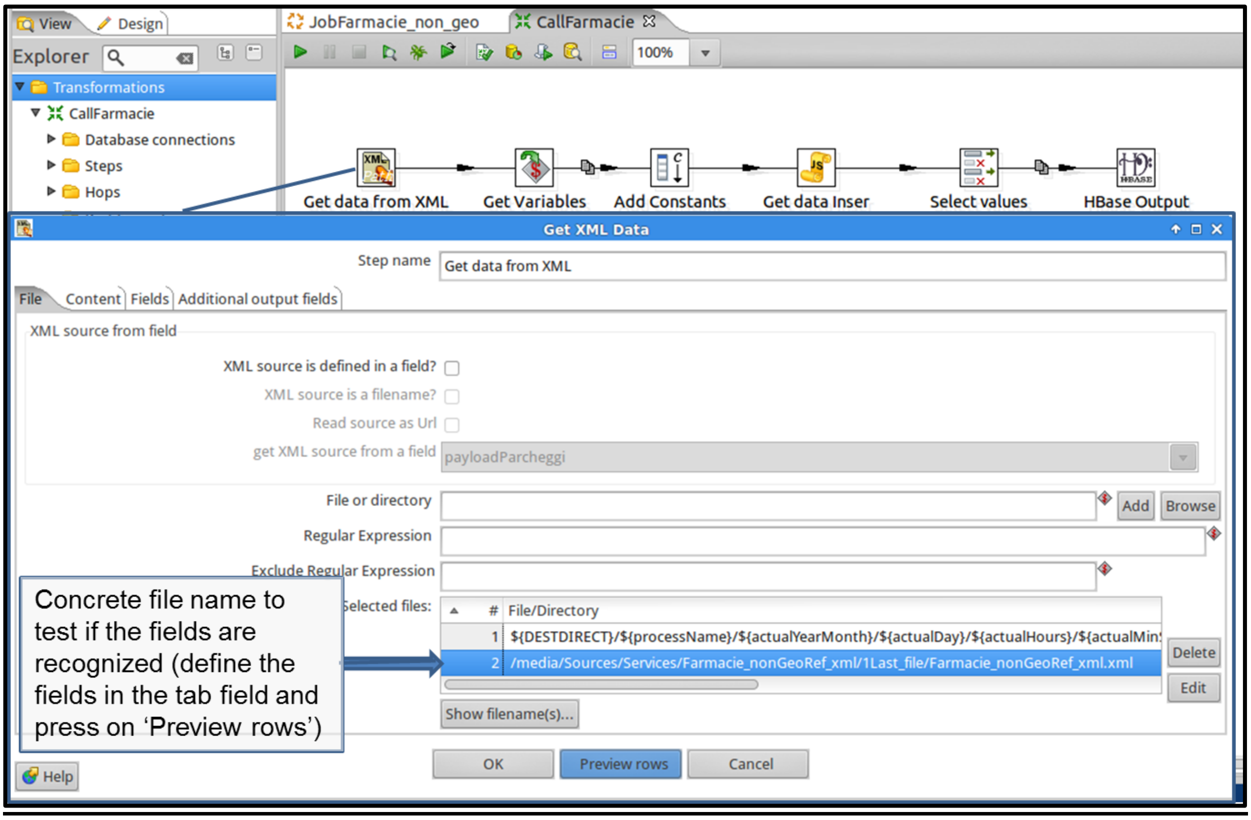
Simple Http Block with the following settings (Fig. 5.2):

- Tab General: URL = ${PARAM} (value of the field ‘param’ in the process_manager2 table obtained in the Database Transformation)
- Target File: is the name of the file into which the data are stored (also the format can be parametrized using ${FORMAT}, value of the field format in the process_manager2 table obtained in the Database Transformation):
  - Example:
    ${DESTDIRECT}/${processName}/${actualYearMonth}/${actualDay}/${actualHours}/${actualMinSec}/${processName}.xml

SET OF SIMPLE BLOCKS TO ESTABLISH IF SOME DIFFERENCES FROM THE LAST UPLOAD OF THE DATASET EXIST (Fig. 5.2)

- Simple blocks to: compare files, delete folders, copy files into folders (note the ‘Replace existing file’ checkbox on active), etc.
- Note that: *File Compare* and *File Exists* blocks ARE action that can have positive or negative consequences (the workflow can proceed in two different ways), so in this case you find the arrows colored in two different modalities:
  - RED: in case of negative match
  - GREEN: in case of positive match

BLOCK TO: i) GET DATA COMING FROM THE FILE UPLOADED AND ii) INSERT THEM ON HBASE (Call Farmacie) (Fig. 5.8)



Step name: Get data from XML

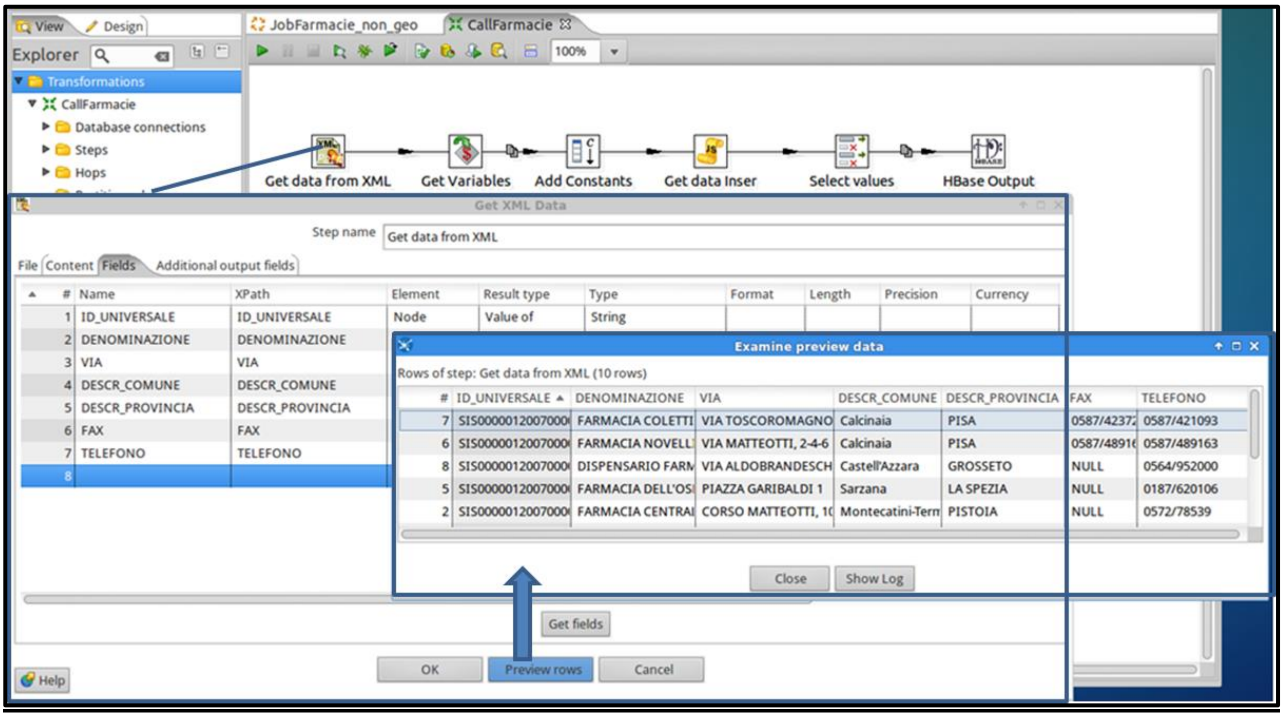
XML source is defined in a field? □
XML source is a filename? □
Read source as Url □
get XML source from a field: payloadParcheggi

File or directory:
Regular Expression:
Exclude Regular Expression:
Selected files:

#	File/Directory
1	${DESTDIRECT}/${processName}/${actualYearMonth}/${actualDay}/${actualHours}/${actualMin:
2	/media/Sources/Services/Farmacie_nonGeoRef_xml/1Last_file/Farmacie_nonGeoRef_xml.xml

Concrete file name to test if the fields are recognized (define the fields in the tab field and press on 'Preview rows')

Fig. 5.8a): Get data (single fields) from the input file.



Step name: Get data from XML

#	Name	XPath	Element	Result type	Type	Format	Length	Precision	Currency
1	ID_UNIVERSALE	ID_UNIVERSALE	Node	Value of	String				
2	DENOMINAZIONE	DENOMINAZIONE							
3	VIA	VIA							
4	DESCR_COMUNE	DESCR_COMUNE							
5	DESCR_PROVINCIA	DESCR_PROVINCIA							
6	FAX	FAX							
7	TELEFONO	TELEFONO							
8									

Rows of step: Get data from XML (10 rows)

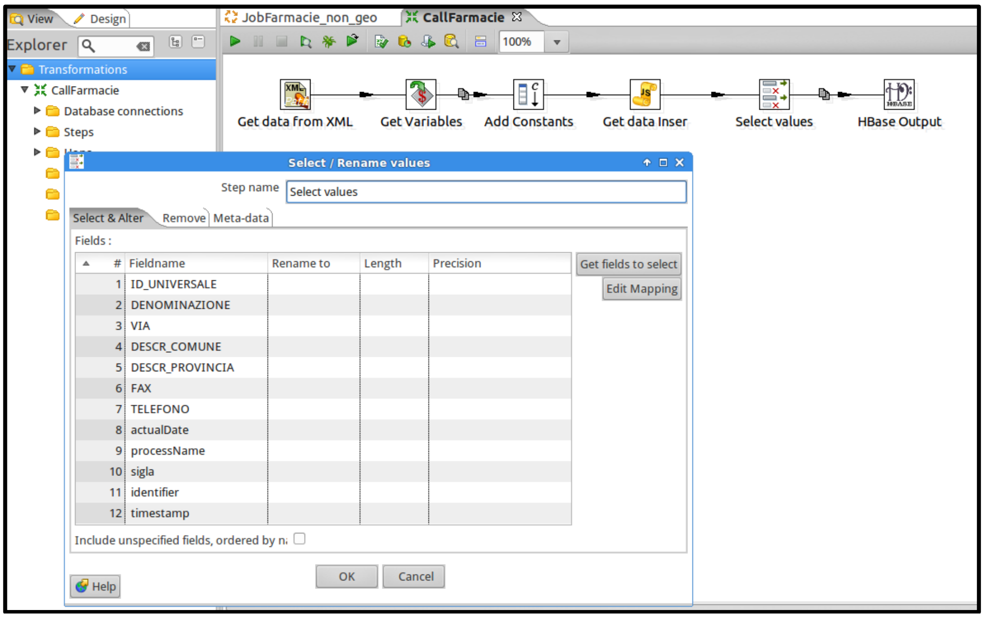
#	ID_UNIVERSALE	DENOMINAZIONE	VIA	DESCR_COMUNE	DESCR_PROVINCIA	FAX	TELEFONO
7	SIS0000012007000(	FARMACIA COLETTI	VIA TOSCOROMAGNO	Calcinaia	PISA	0587/42372	0587/421093
6	SIS0000012007000(	FARMACIA NOVELLI	VIA MATTEOTTI, 2-4-6	Calcinaia	PISA	0587/48916	0587/489163
8	SIS0000012007000(	DISPENSARIO FARM	VIA ALDOBRANDESCH	Castell'Azzara	GROSSETO	NULL	0564/952000
5	SIS0000012007000(	FARMACIA DELL'OSI	PIAZZA GARIBALDI 1	Sarzana	LA SPEZIA	NULL	0187/620106
2	SIS0000012007000(	FARMACIA CENTRAI	CORSO MATTEOTTI, 1(	Montecatini-Term	PISTOIA	NULL	0572/78539

Fig. 5.8b): How to select the interesting fields from the input file (xml).

- <u>Get Data from file (XML)</u>:
  - Fig. 5.8a) Tab File: put the path of the input file. In Fig 5.8 a) you find:
    - Parametrized path (is the same address you put in the block CONNECTION WITH THE WEB SERVER):

${DESTDIRECT}/${processName}/${actualYearMonth}/${actualDay}/${actualHours}/${actualMinSec}/${processName}.xml

    - Concrete path (used to make tests, click on Browse > check your file > click on Add): /media/Sources/Services/Farmacie_nonGeoRef_xml/1Last_file/Farmacie_nonGeoRef_xml.xml
  - Fig 5.8b), here you find the relevant fields that will be extracted from your file:
    - NOTE that the first time you open this tab, you must select the fields:
      1. Put the concrete file path (in the tab 'File')
      2. Click on Get fields
      3. Select the fields that you want extract
    - Press on 'Preview rows' to see if your selection is correct: you can see the results on the windows 'Examine preview data'
  - Click on 'Ok' buttons to save the selection
- Get variables: select the variables necessary in this transformation
- Add Constants: block to add constants as fields when necessary
- Get data insert: JavaScript block useful to calculate the timestamp (actual download instant). Not that this block has as input:
  - The fields coming from the input file (xml file)
  - The variables loaded in the 'Get Variables' block
  - The constant added in the 'Add constant' block
- Select values: this block is useful to select the final fields, that will be saved on HBase:
  - To define the fields, click on the button 'Get fields to select'



Step name: Select values

Fields :

#	Fieldname	Rename to	Length	Precision
1	ID_UNIVERSALE			
2	DENOMINAZIONE			
3	VIA			
4	DESCR_COMUNE			
5	DESCR_PROVINCIA			
6	FAX			
7	TELEFONO			
8	actualDate			
9	processName			
10	sigla			
11	identifier			
12	timestamp			

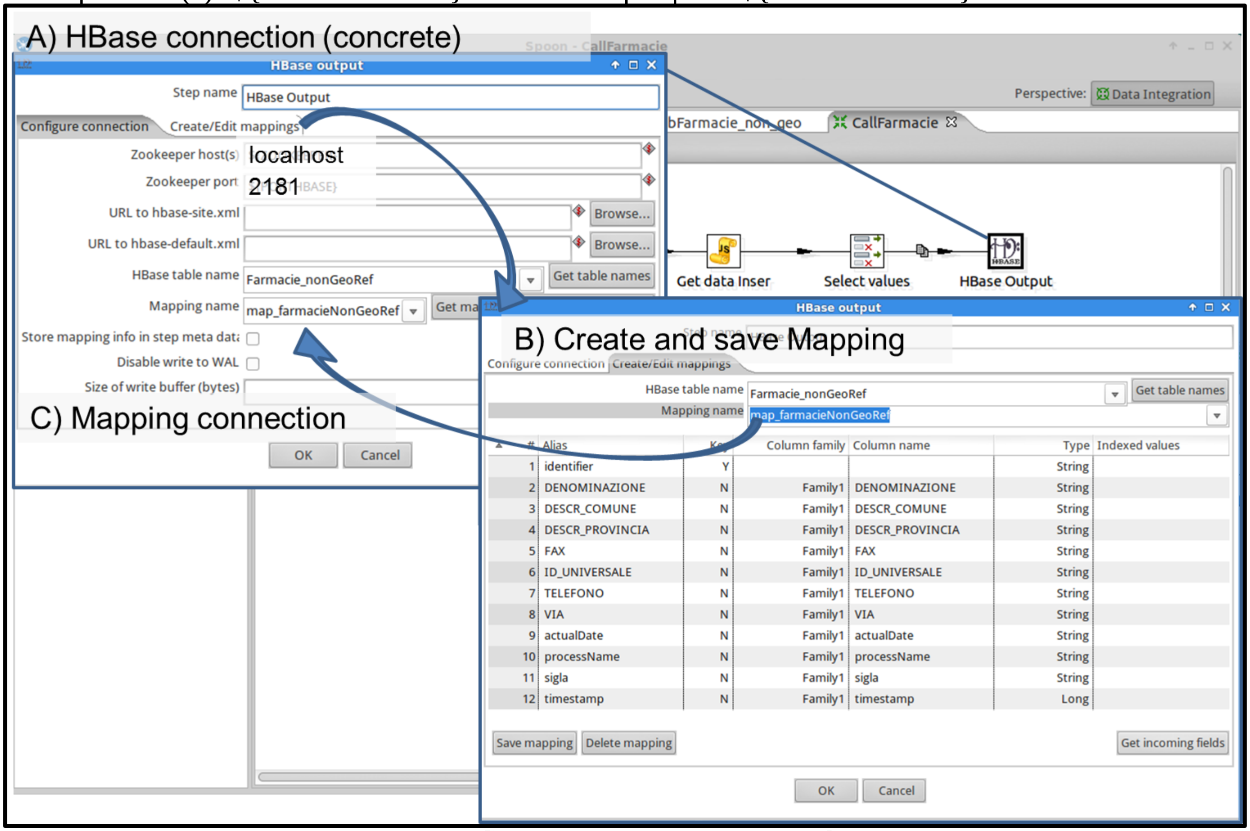
Include unspecified fields, ordered by n: ☐

Fig. 5.8C): Select final fields.

- HBase Output:
  - This block saves the selected field on HBase
  - NOTE that you MUST MANUALLY Create the table on HBase (with the convention: Family='Family1')
  - 1) Tab 'configure connection'
    - Parametrized values (Zookeeper host: ${ZOOKEEPER}, Zookeeper port: ${PORTHBASE})

        - Concrete values (e.g.: Zookeeper host: localhost … | Zookeeper port: 2181). As usual the concrete values are NECESSARY the first time.
- 2) Tab 'Create/edit mappings'
    - Click on the button 'Get table names' and select the table you have created manually (in 1))
    - Click on the button 'Get incoming fields' and create the mapping:
        - Identifier: {Key: Y | Column Family: void | Column name: void | Type: String}
        - timestamp: {Key: N | Column Family: Family1 | Column name: timestamp | Type: Long}
        - Other Fields: {Key: N | Column Family: Family1 | Column name: Field_name | Type: String}
    - Save the mapping (i) give a name, ii) click on 'save mapping')
- Tab 'Configure connection' (again)
    - Click on 'Get table names' and select the table created in 1) and on which you have done the mapping
    - Click on the 'Get mapping for the specified table' and select your mapping (that you have created at point 2))
    - Click on 'ok' button

<u>NOTE:</u> use of the variables in this block once you have created and tested the mapping, ZOOKEEPER and PORTHBASE have been defined in the configuration file (config_file.csv): Zookeeper host(s): ${ZOOKEEPER} and Zookeeper port: ${PORTHBASE}



A) HBase connection (concrete)

Step name: HBase Output

Configure connection | Create/Edit mappings

Zookeeper host(s): localhost

Zookeeper port: 2181

URL to hbase-site.xml:

URL to hbase-default.xml:

HBase table name: Farmacie_nonGeoRef

Mapping name: map_farmacieNonGeoRef

Store mapping info in step meta data ☐

Disable write to WAL ☐

Size of write buffer (bytes):

C) Mapping connection

B) Create and save Mapping

Configure connection | Create/Edit mappings

HBase table name: Farmacie_nonGeoRef

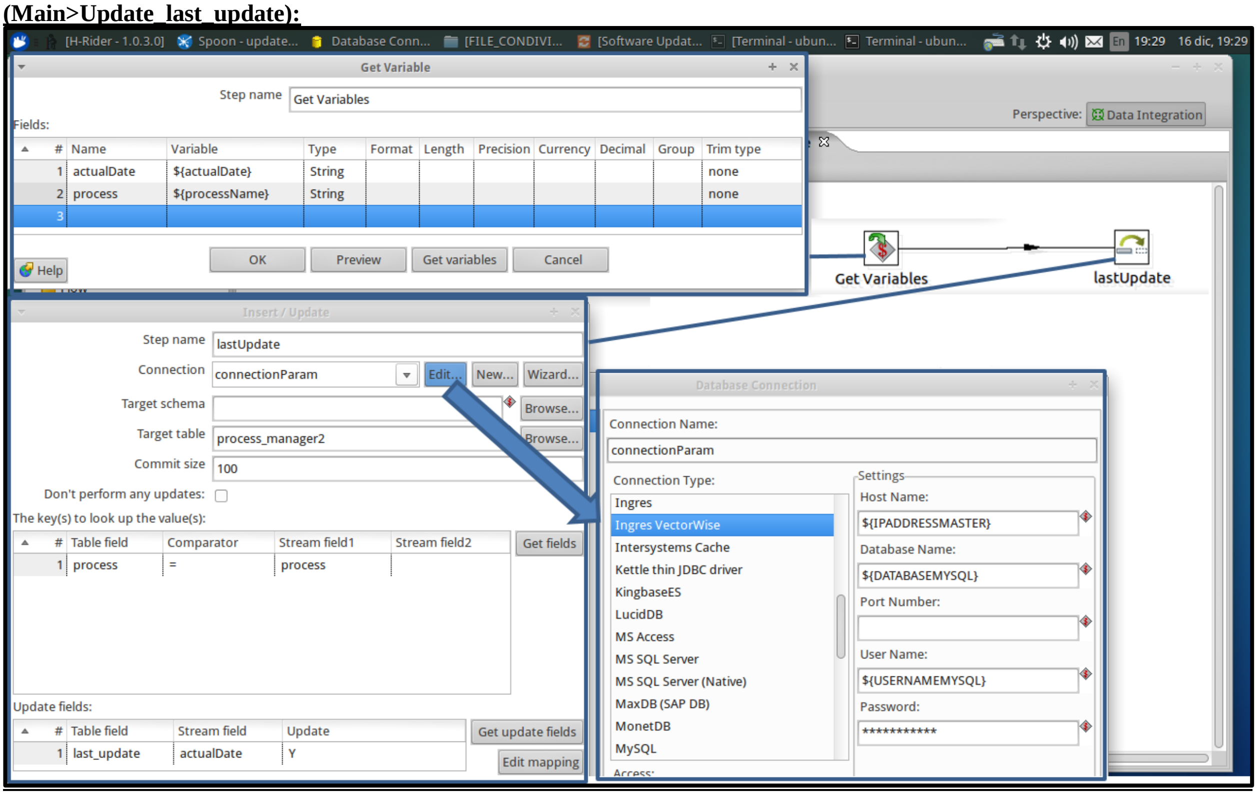
Mapping name: map_farmacieNonGeoRef

#	Alias	Key	Column family	Column name	Type	Indexed values
1	identifier	Y			String	
2	DENOMINAZIONE	N	Family1	DENOMINAZIONE	String	
3	DESCR_COMUNE	N	Family1	DESCR_COMUNE	String	
4	DESCR_PROVINCIA	N	Family1	DESCR_PROVINCIA	String	
5	FAX	N	Family1	FAX	String	
6	ID_UNIVERSALE	N	Family1	ID_UNIVERSALE	String	
7	TELEFONO	N	Family1	TELEFONO	String	
8	VIA	N	Family1	VIA	String	
9	actualDate	N	Family1	actualDate	String	
10	processName	N	Family1	processName	String	
11	sigla	N	Family1	sigla	String	
12	timestamp	N	Family1	timestamp	Long	

Fig. 8d) HBase Mapping - Details.

BLOCK TO SAVE THE JOB (MAIN PROCESS) STATUS ON THE MYSQL DATABASE

(Main>Update_last_update):



Get Variable

Step name: Get Variables

Fields:

#	Name	Variable	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	actualDate	${actualDate}	String							none
2	process	${processName}	String							none
3										

Insert / Update

Step name: lastUpdate

Connection: connectionParam

Target schema:

Target table: process_manager2

Commit size: 100

Don't perform any updates: □

The key(s) to look up the value(s):

#	Table field	Comparator	Stream field1	Stream field2
1	process	=	process	

Update fields:

#	Table field	Stream field	Update
1	last_update	actualDate	Y

Database Connection

Connection Name: connectionParam

Connection Type: Ingres, Ingres VectorWise, Intersystems Cache, Kettle thin JDBC driver, KingbaseES, LucidDB, MS Access, MS SQL Server, MS SQL Server (Native), MaxDB (SAP DB), MonetDB, MySQL

Access:

Settings

Host Name: ${IPADDRESSMASTER}

Database Name: ${DATABASEMYSQL}

Port Number:

User Name: ${USERNAMEMYSQL}

Password: ***********

Fig. 5.9 MySQL update (with conditional access) - Details.

- In this case the connection to the MySQL DB is established and it will be updated the row of the table ‘process_manager2’ with the column (field) ‘process’ equal to the processName (process = processName = ‘Farmacie_nonGeoRef_xml’). [Is a modality to Select a Row of the table]
    - The column (field) that will be updated is ‘last_update’. In this way everytiome the Ingestion process is launched, the data (‘last_update’) will be updated.

BLOCK TO EXIT WITH SUCCESS FROM THE JOB

- This block exit from the ETL in case of success and return an integer (1)

Final Result of Phase I, (Ingestion): in HBase is present a table with the uploaded data (nomenclature: ‘dataset_name’)

5.4.1.1.2 Phase II: Quality Improvement

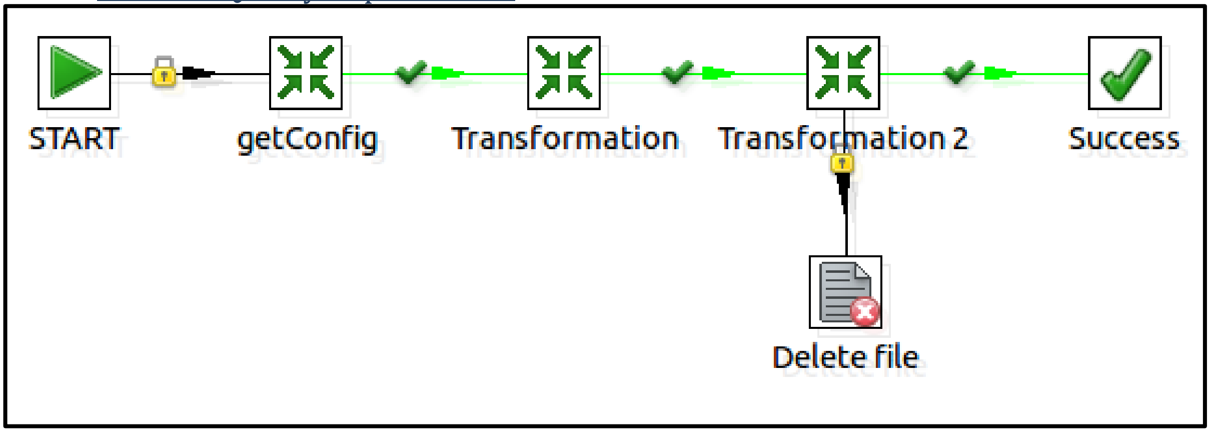


Fig. 5.10 Quality Improvement Phase – Main job.

The second phase (QI) includes the operations necessary to improve the quality of data retrieved previously (in the ingestion phase) and it was designed for those datasets that contain data with many mistakes and many inaccuracies. In this phase, each ETL transformation developed allows to improve the content of each field present in the specific dataset.

The Quality Improvement Phase is composed of the following actions (referring to Fig. 5.10):

- **Process parameters** (click on the green triangle to start the Process/Spoon Job):
  - In some processes (you find the processName parameter that must be specified at the execution time and saved in the 'process_manager_2 Mysql table')
- **Configuration blocks:** blocks containing the configuration of the variables that can be used in the processes:
  - 'getConfig':
    - Transformation composed by the blocks: 'Text file input' and 'Set Variables' (see the same Transformation explained in the ingestion phase) [for more details, see the Ingestion phase]
- **Transformation1:**
  - Considers the fields of the dataset and makes Quality improvement on each of them (in this a case this step has its focus on phone and fax numbers, street names, etc.), the results are written in a temporary file
- **Transformation2:**
  - This step upload the results, reading the text file written during the previous step ('Transformation1'), into HBase and then delete the temporary file
- **Success Block** to exit from the process:
  - 'Success' (the output is '1' if the ETL process ends with success)

Expanding the jobs and the transformations, it is possible to describe more details (only those not seen in the ingestion phase):

TRANSFORMATION1:

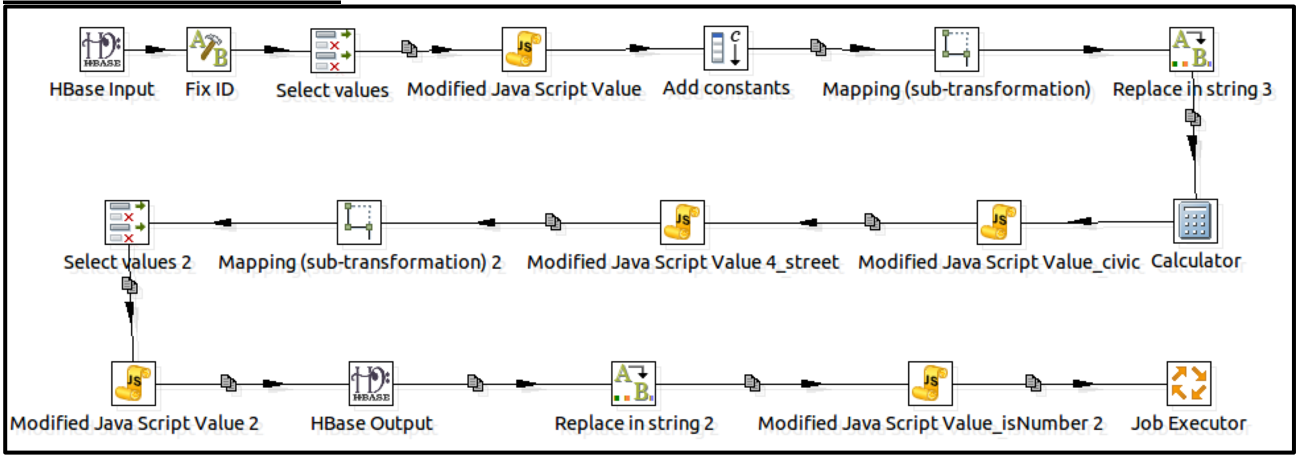
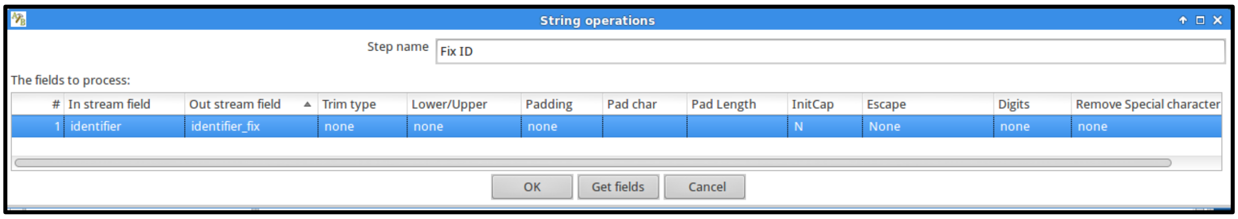

HBase Input
Fix ID
Select values
Modified Java Script Value
Add constants
Mapping (sub-transformation)
Replace in string 3
Select values 2
Mapping (sub-transformation) 2
Modified Java Script Value 4_street
Modified Java Script Value_civic
Calculator
Modified Java Script Value 2
HBase Output
Replace in string 2
Modified Java Script Value_isNumber 2
Job Executor

Fig. 5.10 QI Phase – Transformation1.

- HBase input:
  - Establishes the connection with HBase
  - Extracts from HBase, the data uploaded during the ingestion phase
- FIXID:



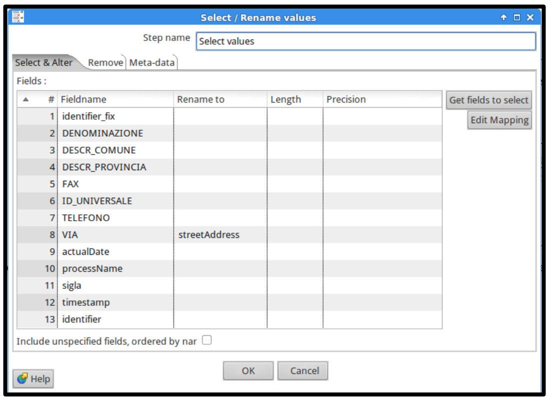
Step name: Fix ID

The fields to process:

#	In stream field	Out stream field	Trim type	Lower/Upper	Padding	Pad char	Pad Length	InitCap	Escape	Digits	Remove Special character
1	identifier	identifier_fix	none	none	none			N	None	none	none

Fig. 5.11 QI Phase – Transformation1, FIX ID

  - Executes string operations (checks) on the 'identifier' field
  - As output obtains the new checked field 'identifier_fix'
- Select values (Fig.5.12):
  - This step is useful to check the field names (column of the database tables) and eventually modify them.
  - In the example al the input fields maintain the same name except for 'VIA' that will be renamed to 'streetAddress'



Step name: Select values

Fields :

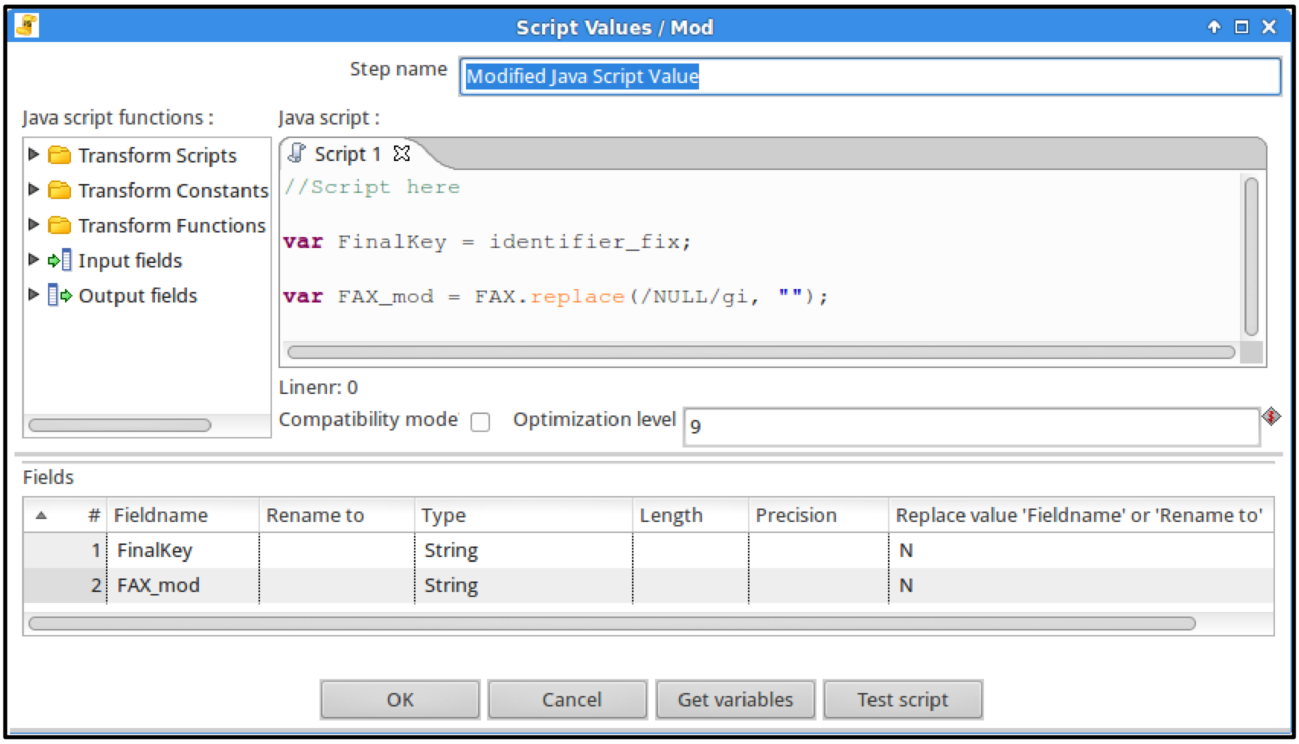
#	Fieldname	Rename to	Length	Precision
1	identifier_fix			
2	DENOMINAZIONE			
3	DESCR_COMUNE			
4	DESCR_PROVINCIA			
5	FAX			
6	ID_UNIVERSALE			
7	TELEFONO			
8	VIA	streetAddress		
9	actualDate			
10	processName			
11	sigla			
12	timestamp			
13	identifier			

Include unspecified fields, ordered by nar □

Fig. 5.12 QI Phase – Transformation1, selects values

- Modified Java Script Value (Fig.5.13):
  - Use the field checked field 'identifier_fix' and put it in the variable 'FinalKey'

- Check the input field ‘FAX’ and put the validated data in the variable ‘FAX_mod’



Step name: Modified Java Script Value

```
//Script here

var FinalKey = identifier_fix;

var FAX_mod = FAX.replace(/NULL/gi, "");
```

Linenr: 0

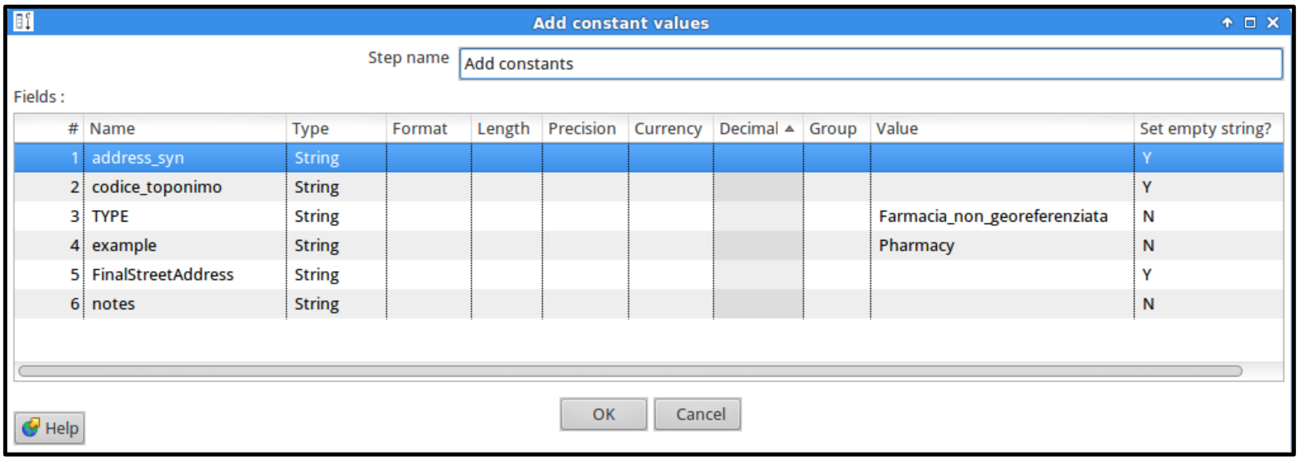
Compatibility mode □ Optimization level 9

Fields

#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fieldname' or 'Rename to'
1	FinalKey		String			N
2	FAX_mod		String			N

Fig. 5.13 QI Phase – Transformation1, JavaScript block.

- Add constants (Fig. 5.14):
  - Add some fields that will be used in this transformation, in some cases put a default value (ex: TYPE = ’Farmacia_non_georeferenziata’)
  - Note that all the fields and treated as **String**



Step name: Add constants

Fields :

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Value	Set empty string?
1	address_syn	String								Y
2	codice_toponimo	String								Y
3	TYPE	String							Farmacia_non_georeferenziata	N
4	example	String							Pharmacy	N
5	FinalStreetAddress	String								Y
6	notes	String								N

Fig. 5.14 QI Phase – Transformation1, Add constants block.

- Mapping1 (Fig.5.15):
  - This Mapping takes a file as input (left click on > block settings, Fig.5.15a))
  - It executes a set of string transformations (right click on) to ‘normalize’ the phone numbers (Fig. 5.15 b)), using:
    - JavaScript blocks - Fig. 5.15c
    - filters

- selections
- appending fields contents
- …

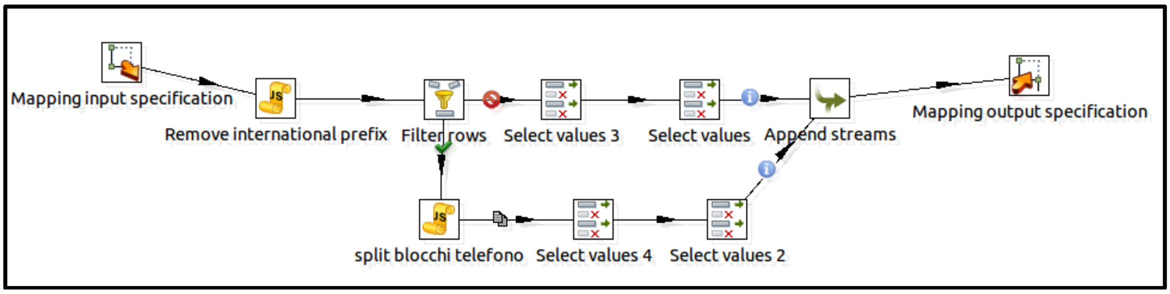
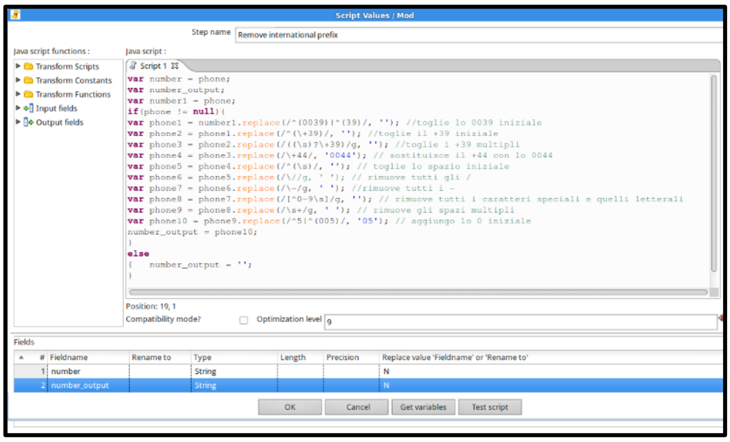


Fig. 5.15b QI Phase – Transformation1, Mapping.



Step name: Remove international prefix

```
var number = phone;
var number_output;
var number1 = phone;
if(phone != null){
var phone1 = number1.replace(/^(0039)|^(39)/, ''); //toglie lo 0039 iniziale
var phone2 = phone1.replace(/^(\+39)/, ''); //toglie il +39 iniziale
var phone3 = phone2.replace(/((\s)?\+39)/g, ''); //toglie i +39 multipli
var phone4 = phone3.replace(/\+44/, '0044'); // sostituisce il +44 con lo 0044
var phone5 = phone4.replace(/^(\s)/, ''); // toglie lo spazio iniziale
var phone6 = phone5.replace(/\//g, ' '); // rimuove tutti gli /
var phone7 = phone6.replace(/\-/g, ' '); //rimuove tutti i -
var phone8 = phone7.replace(/[^0-9\s]/g, ''); // rimuove tutti i caratteri speciali e quelli letterali
var phone9 = phone8.replace(/\s+/g, ' '); // rimuove gli spazi multipli
var phone10 = phone9.replace(/^5|^(005)/, '05'); // aggiungo lo 0 iniziale
number_output = phone10;
}
else
{    number_output = '';
}
```

Position: 19, 1

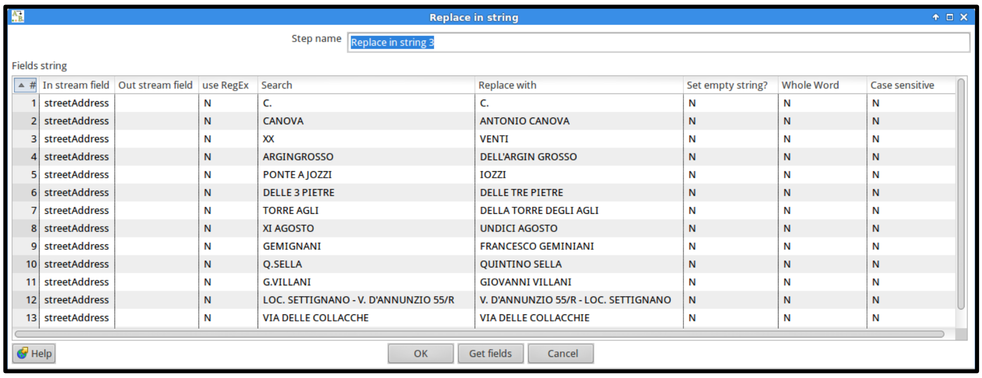
Compatibility mode? □ Optimization level 9

Fields

#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fieldname' or 'Rename to'
1	number		String			N
2	number_output		String			N

Fig. 5.15c QI Phase – Transformation1, Mapping: Removing international prefix.

- Replace in string 3 (Fig.5.16): make string operations on the field ‘streetAddress’. This modality is used to manage some errors or problems (it is not a general transformation, but specific).



Step name: Replace in string 3

Fields string

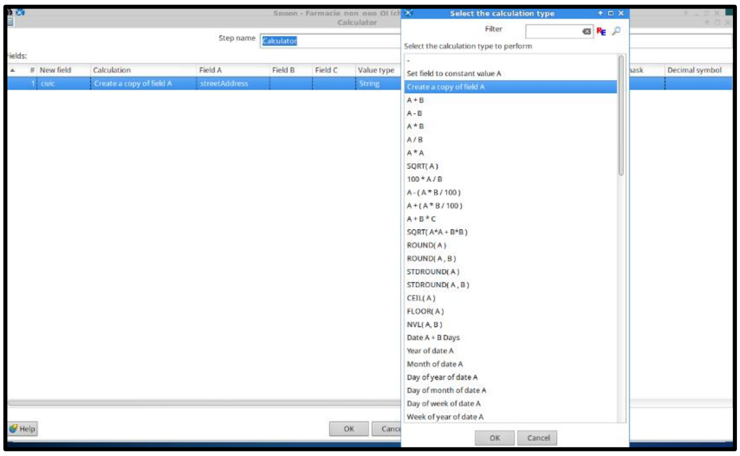
#	In stream field	Out stream field	use RegEx	Search	Replace with	Set empty string?	Whole Word	Case sensitive
1	streetAddress		N	C.	C.	N	N	N
2	streetAddress		N	CANOVA	ANTONIO CANOVA	N	N	N
3	streetAddress		N	XX	VENTI	N	N	N
4	streetAddress		N	ARGINGROSSO	DELL'ARGIN GROSSO	N	N	N
5	streetAddress		N	PONTE A JOZZI	IOZZI	N	N	N
6	streetAddress		N	DELLE 3 PIETRE	DELLE TRE PIETRE	N	N	N
7	streetAddress		N	TORRE AGLI	DELLA TORRE DEGLI AGLI	N	N	N
8	streetAddress		N	XI AGOSTO	UNDICI AGOSTO	N	N	N
9	streetAddress		N	GEMIGNANI	FRANCESCO GEMINIANI	N	N	N
10	streetAddress		N	Q.SELLA	QUINTINO SELLA	N	N	N
11	streetAddress		N	G.VILLANI	GIOVANNI VILLANI	N	N	N
12	streetAddress		N	LOC. SETTIGNANO - V. D'ANNUNZIO 55/R	V. D'ANNUNZIO 55/R - LOC. SETTIGNANO	N	N	N
13	streetAddress		N	VIA DELLE COLLACCHE	VIA DELLE COLLACCHIE	N	N	N

Fig. 5.16 QI Phase – Transformation1, Mapping: Removing international prefix.

- Calculator: (http://wiki.pentaho.com/display/EAI/Calculator)
  - This step is useful to execute operation on fields. It takes as input Field A, Field B, Field C (that can be selected from the input field available in the transformation) and make calculus on them. It possible to select the type of calculus that must be executed clicking on ‘Calculation’ column of the windows. Some of the possible calculus are the following: *Set field to constant value A, Creates a copy of Field,*

    A+B, A-B, …, SQRT(A), ROUND(A), ABS(A), …, Date A +B Days, Year of a date A, …, Byte to hex encode of string A, …, Checksum of a file A using MD5, …, Remove TAB from string A, Remove digits from string A, Return the length of string A, …, Escape/unescape html/xml SQL content, …, check if an xml file A is well formed, check if an xml string A is well formed, …

    - In this case, it *Creates a copy* of field 'civic'



Spoon - Farmacie_non_geo_QI (ch
Calculator

Step name: Calculator

Fields:

#	New field	Calculation	Field A	Field B	Field C	Value type	…ask	Decimal symbol
1	civic	Create a copy of field A	streetAddress			String		

Select the calculation type

Filter

Select the calculation type to perform

Fig. 5.17 QI Phase – Transformation1, Calculator step.

- Modified Java Script Value_civic & Modified Java Script Value 4_street:
    - JavaScript blocks to modify specific content of the fields: 'civic_number' and 'street'
- Mapping2:
    - Executes operations on the field 'FAX_mod'
- Select values2:
    - Select the correct fields that will be put into HBase
- Modified Java Script Value 2:
    - Make some specific corrections related to the 'Address' field
- **HBase Output**:
    - Establishes the connection with HBase
    - Upload all the values (remember that the original fields come from the HBase ingestion Table – named for this dataset 'Farmacie_nonGeoRef') of the fields selected in the block 'Select values2'. The output table is conventionally named as: 'name_of_the_ingetion_phase_table'_QI (in this dataset case: 'Farmacie_nonGeoRef_QI')
- Replace in string 2
- Modified Java Script Value_is_Number:
    - It takes in input the field 'Address'
    - It executes a set operations on the field 'Address' (for each row) to elaborate the most significant word (and put the results in the variable 'parola'):
        - General rule: It is supposed that the most significant word of an address could be with a large probability the most long
        - Specific rules: in some cases, the general rule it is not true, so a set of particular cases are also managed in this block
- Job Executor (right click on - Fig. 5.18):

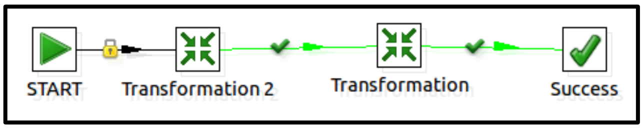
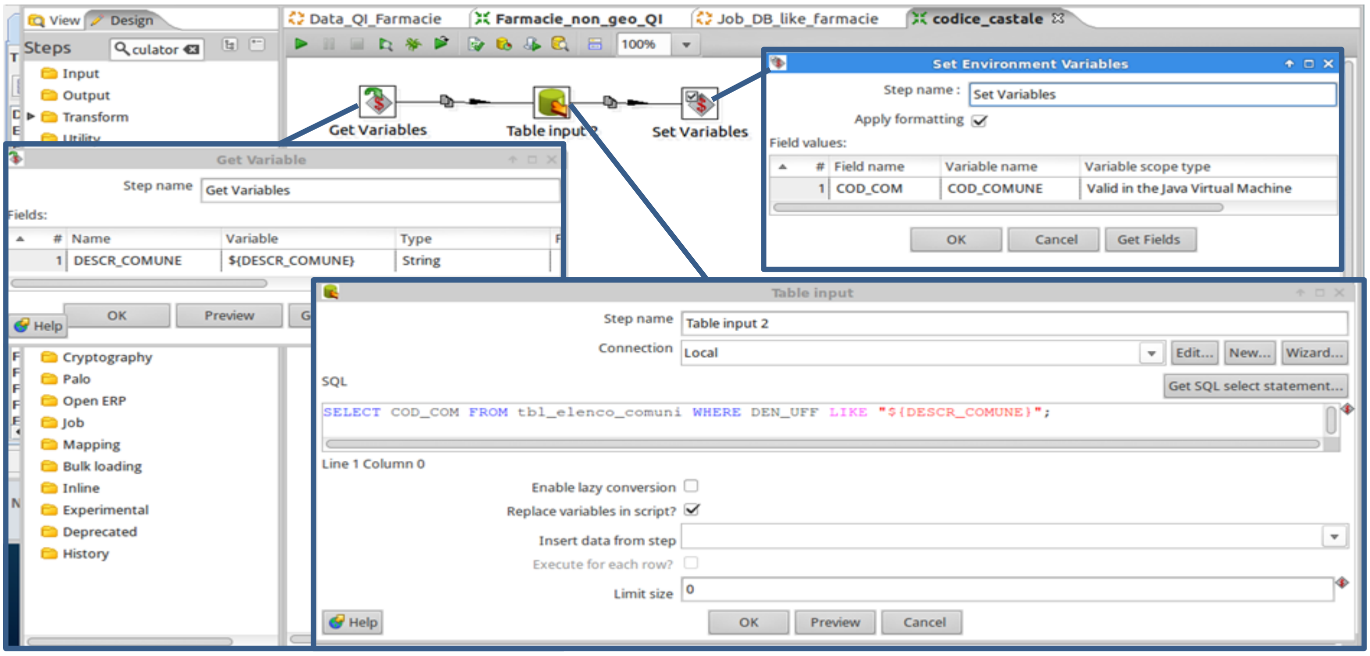


Fig. 5.18 QI Phase – Transformation1, Job Executor.

- **NOTE that** after the **HBase Output** previously described, the data coming from the ingestion phase has just been modified and improved. The data comes from the HBase ingestion table ‘‘Farmacie_nonGeoRef’ produced after the phase I (ingestion) execution, and then (during this phase II, the Quality improvement phase) it has been modified and put into the table ‘Farmacie_nonGeoRef_QI’
- The only information that must be added and associated to each row, at this point of this ETL, is the field ‘TOPONIMO’ (a Toponimo is a street unique identifier) that is relevant to connect at semantic level each data content to the others contained in the KM4city Knowledge Base.
    - For dataset in Tuscany, we have a relation among all the streets and their respective unique identifiers (TOPONIMO) and coordinates (Latitude, Longitude)
    - For other places, we will consider different modalities to obtain (thanks to this QI phase) all the data as geo-localized data

Expanding the transformations of the Job Executor, (starting from Fig. 5.18):



Get Variable

Step name: Get Variables

Fields:

#	Name	Variable	Type
1	DESCR_COMUNE	${DESCR_COMUNE}	String

Set Environment Variables

Step name : Set Variables

Apply formatting ☑

Field values:

#	Field name	Variable name	Variable scope type
1	COD_COM	COD_COMUNE	Valid in the Java Virtual Machine

Table input

Step name: Table input 2

Connection: Local

SQL

```
SELECT COD_COM FROM tbl_elenco_comuni WHERE DEN_UFF LIKE "${DESCR_COMUNE}";
```

Line 1 Column 0

Enable lazy conversion ☐

Replace variables in script? ☑

Insert data from step

Execute for each row? ☐

Limit size 0

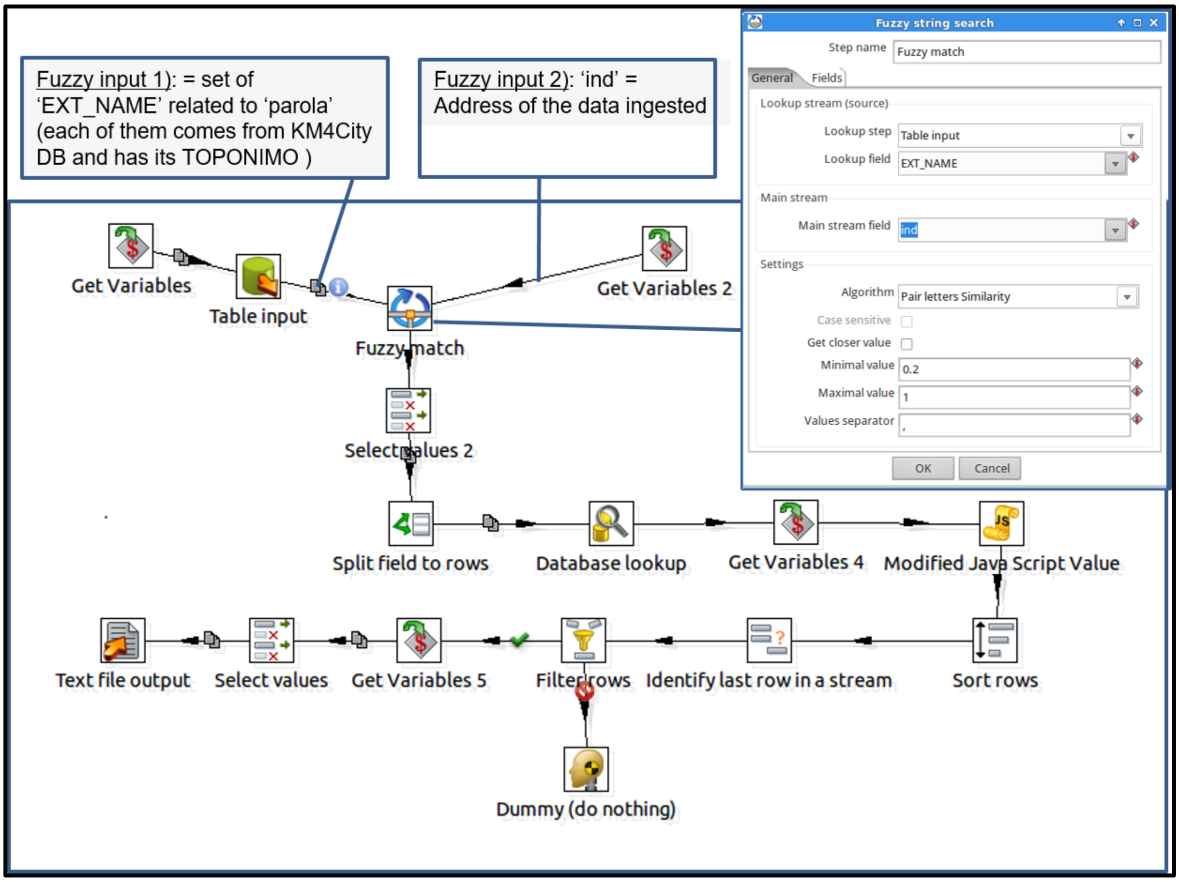
- Job Executor > TRANSFORMATION 2:

Fig. 5.19 QI Phase – Transformation1, Job Executor > Transformation 2.

    - This Transformation associates a code (COD_COM) to each row, necessary for the final association of the TOPONIMO, LATITUDE and LONGITUDE fields
    - This Transformation makes an association between each row and the code of the city ('comune') to which the data is related (ex, the query ‘SELECT

COD_COM FROM tbl_elenco_comuni WHERE DEN_UFF LIKE "${DESCR_COMUNE}’ in FIG. 19: associate to each row the 'COD_COM' contained in the table 'tbl_elenco_comuni' of the KM4City Database). This step is necessary for the final association of the field TOPONIMO.

- <u>Job Executor > TRANSFORMATION</u>
  - This Transformation comprehends:
    - i) a Fuzzy match block that makes a query on the KM4City MySQL DB (input 1, Lookup field = ‘EXT_NAME’ in Fig.19, basing on the most significant word of the field address, contained in the variable ‘parola’ and calculated in the previous ‘Modified Java Script Value_is_Number’ block, Fig.10) and compares the result of the query (a set of probably addresses, each of them having its TOPONIMO) with each Address (variable ‘ind’ in fig. 19) of the dataset. The Fuzzy match block return a value, a value that is between 0.2 and 1,for each match.



<u>Fuzzy input 1)</u>: = set of ‘EXT_NAME’ related to ‘parola’ (each of them comes from KM4City DB and has its TOPONIMO)

<u>Fuzzy input 2)</u>: ‘ind’ = Address of the data ingested

Fuzzy string search

Step name: Fuzzy match

General | Fields

Lookup stream (source)

Lookup step: Table input

Lookup field: EXT_NAME

Main stream

Main stream field: ind

Settings

Algorithm: Pair letters Similarity

Case sensitive: □

Get closer value: □

Minimal value: 0.2

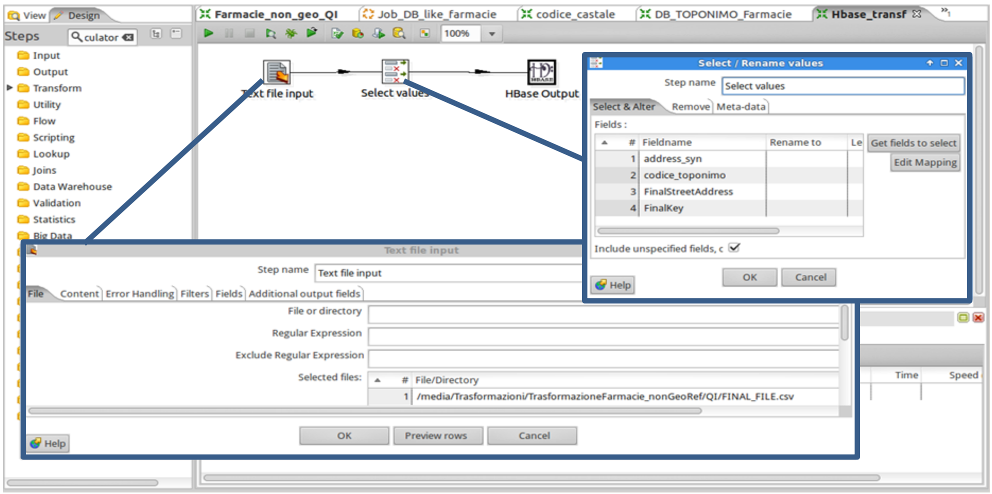
Maximal value: 1

Values separator: ,

Fig.5.19 QI Phase – Transformation1, Job Executor > Transformation 2.

- Then, the most likely value corresponding to each address, is selected through a series of steps (based also on Latitude and Longitude values, if these fields are present in the dataset, and in the KM4City DB). In this way, the association ‘Address’ (of the dataset ingested) <-> TOPONIMO can be done.

<u>TRANSFORMATION2</u> <u>(Fig. 20):</u>



Select / Rename values

Step name: Select values

Fields :

#	Fieldname	Rename to	Le
1	address_syn		
2	codice_toponimo		
3	FinalStreetAddress		
4	FinalKey		

Include unspecified fields, c ☑

Text file input

Step name: Text file input

File or directory

Regular Expression

Exclude Regular Expression

Selected files:

#	File/Directory
1	/media/Trasformazioni/TrasformazioneFarmacie_nonGeoRef/QI/FINAL_FILE.csv

Fig. 5.20 QI Phase – Transformation2, HBase final upload.

- This is the final Transformation
- It is visible in Fig.5.10, if you click on 'Transformation2'and expand the transformation, you see the details in Fig. 5.20.
- It uploads (update) the complete results, reading the text file input written during the previous step ('Transformation1'), into HBase

<u>Final Result of Phase II, (Quality improvement): in HBase is present a table with the improved data (nomenclature: 'dataset_name'_QI)</u>

5.4.1.1.3 <u>Phase III: Triplification</u>

Before this Phase, it is necessary to make the Mapping of the dataset ingested with the KM4City ontology. To do this it is possible to use Karma and realize the models ad hoc for each ETL (for more information see the karma for KM4City guidelines, [ETLGuide]).

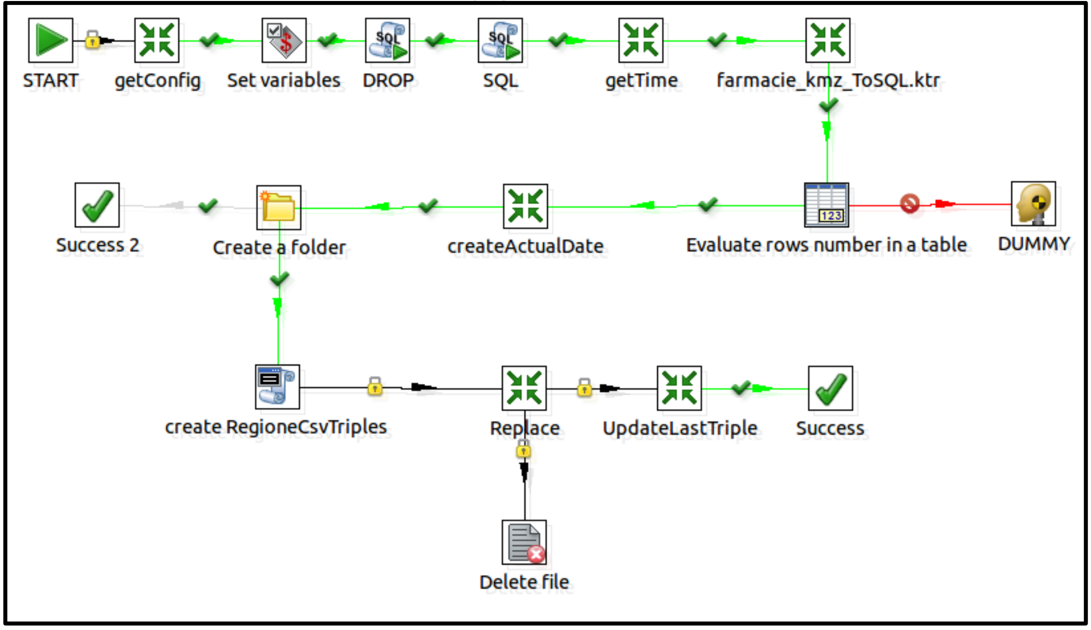
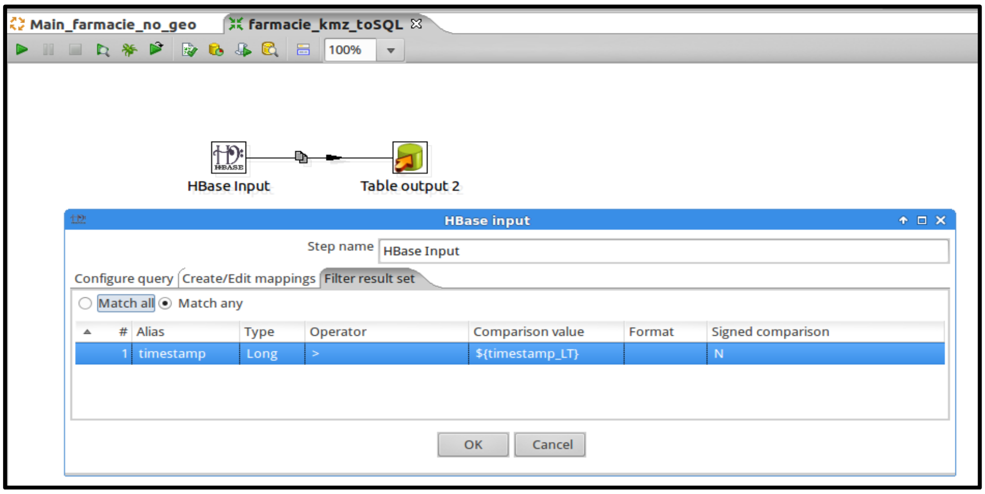


Fig. 5.21 Triplification Phase – Main job

.

The Triplification Phase is composed of the following main actions (referring to Fig. 5.21):

- **Process parameters** (click on the green triangle to start the Process/Spoon Job):
    - In some processes (you find the 'processName' parameter that must be specified at the execution time and saved in the 'process_manager_2 Mysql table')
- **Configuration blocks:** blocks containing the configuration of the variables that can be used in the processes:
    - 'getConfig' and 'Set Variables' (Spoon Transformations)
- **Drop:** drops (if exists) the MySQL table used every time the Triplification phase is launched (so it drops the old data) and that is create in the next step
- **SQL:** creates the new MySQL table used every time the Triplification
- **GetTime:** makes a connection to the MySQL database and takes the date in which the LAST triples have been done (variable: 'timestamp_LT', valid in the root job)
- **Farmacie_kmz_ToSQL.ktr:**
    - This transformation moves the data from HBase to MySQL (from the table created during the phase II, es: 'Farmacie_nonGeoRef_QI'). The copy of the data from HBase to MySQL is necessary to create the mapping with the KM4City ontology. The mapping can be realized with Karma ([Karma]).
    - Note that the HBase input step has a filter (Fig. 22): it copies from the QI HBase table, ONLY the rows having the 'timestamp' (time at which the data has been INGESTED) greater than 'timestamp_LT' (calculated in the 'GetTime' previuos step, is the time in which the last triples have been done). This filter is necessary to avoid the duplication of the triples.



Step name: HBase Input

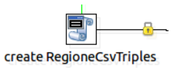
○ Match all ● Match any

#	Alias	Type	Operator	Comparison value	Format	Signed comparison
1	timestamp	Long	>	${timestamp_LT}		N

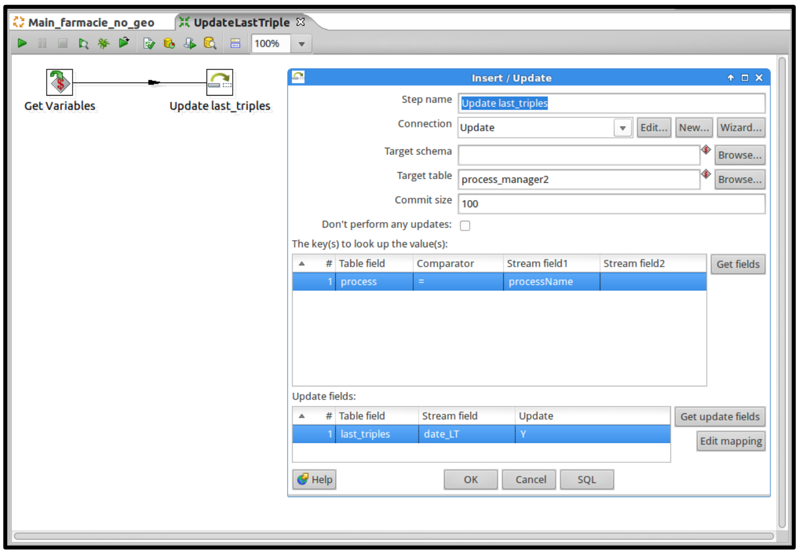
Fig. 5.21 Triplification Phase – Main job.

- **Evaluate rows number in a table:**
    - Input: the results coming from the previous step (the rows present in the QI HBase table with timestamp > timestamp_LT)
    - Output: it has two possible flows:
        - A) DUMMY Dummy step, if it has no INPUT (no rows having timestamp > timestamp_LT are present in the QI table), so no NEW triples have to be done). In this case the Phase III not ends with a success.
        - B) continuation of the work flow toward the next step ('Create Actual Date')
- **Create Actual Date**: set the actual date (in a variable valid in the root job)

- **Create a Folder**: creates the folder into which the dataset will be uploaded (according with the KM4City nomenclature previously described)
- **Create RegioneCsvTriples:**



  - this block executes a shell script and use the Karma model to create the Triples
- **Replace:** this block makes some string operation on the Triples
- **UpdateLastTriples:** (Fig. 5.22)
  - This Transformation update the 'last_triples' date column of the MYSQL table 'process_manager2', to update the status of the actions done on this dataset.
- **Success** Block to exit from the process:
  - 'Success…' (the output is '1' if the ETL process ends with success)



Insert / Update

Step name: Update last_triples

Connection: Update

Target schema:

Target table: process_manager2

Commit size: 100

Don't perform any updates: □

The key(s) to look up the value(s):

#	Table field	Comparator	Stream field1	Stream field2
1	process	=	processName	

Update fields:

#	Table field	Stream field	Update
1	last_triples	date_LT	Y

Fig. 5.22 Triplification Phase – UpdateLastTriples.

<u>Final Result of Phase III, (Triplification): a file (or a set of file) containing the RDF Triples</u>

6 Mapping to Km4City Ontoloyg

6.1 Prepare Karma

1. Karma data integration ver. 2.024 (for the Triplification Phase, [Karma])

   - Start from the folder "/programs/Web-Karma-master/karma-web" with the command ***mvn -Djetty.port=9999 jetty:run (shell)***
   - Access from web http://localhost:9999 (Fig.5.23)

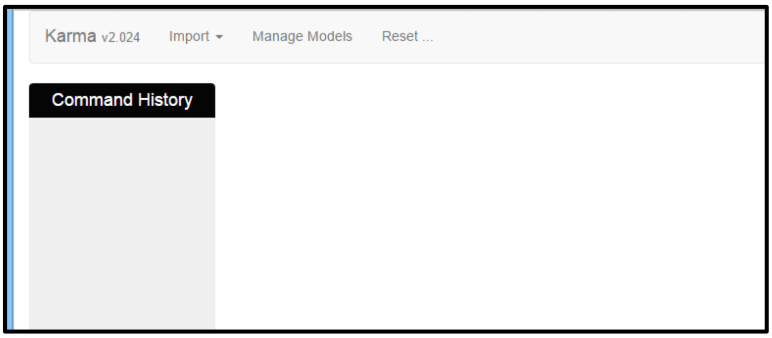


Fig. 5.23 Karma home page

2. How to import the right ontologies: select from the menu: '**Import/From File**' (Fig.5.24)

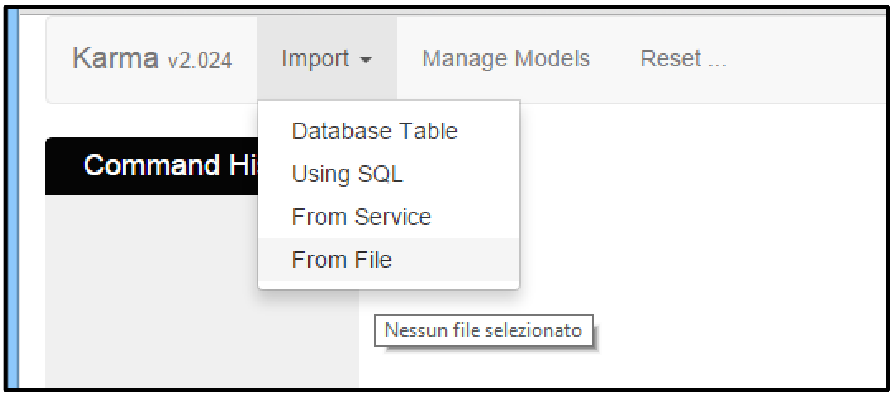
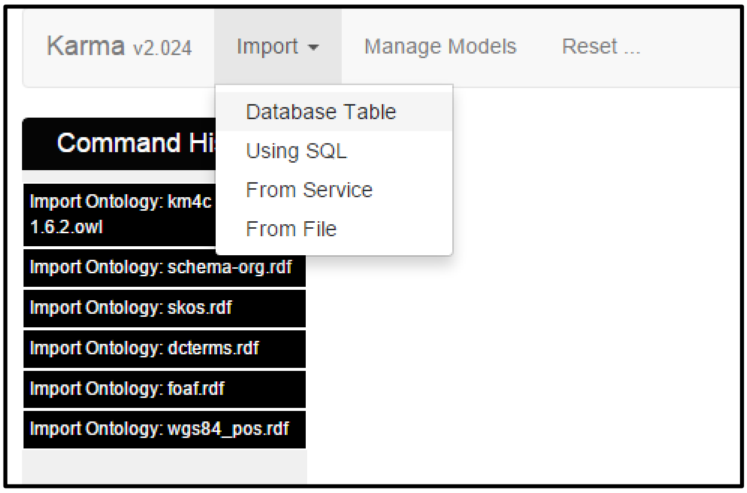


Fig. 5.24 Karma: Import/From File.

3. The most relevant ontologies that must be uploaded are:
   - Km4city
   - Schema.org
   - Dcterms
   - Skos
   - FOAF
   - Wgs84 (per long, lat)
   - Geosparql (per Geometry, asWKT, ...)
   - …

Other ontologies could be useful, depending on the specific type of dataset (e.g. time, cito, otn, etc.)

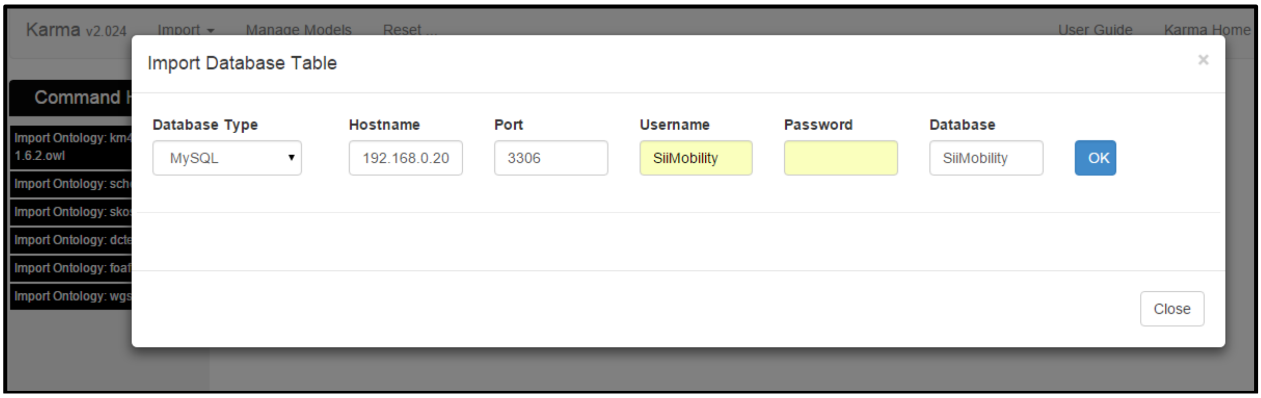
4. Upload the MySQL table on which it is necessary to create the mapping, select from the menu: **Import/Database Table** (Fig. 5.25):



Import Ontology: km4c 1.6.2.owl
Import Ontology: schema-org.rdf
Import Ontology: skos.rdf
Import Ontology: dcterms.rdf
Import Ontology: foaf.rdf
Import Ontology: wgs84_pos.rdf

Fig. 5.25 Karma: Import MySQL table.

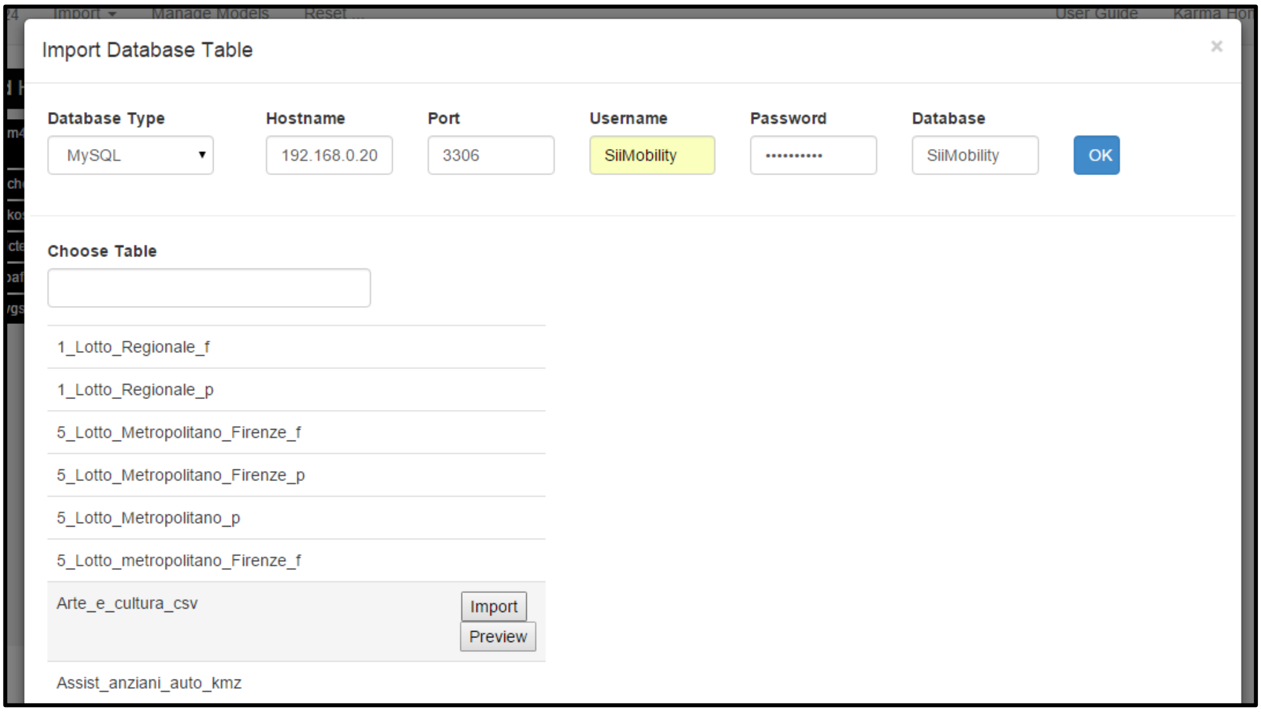
Select: {Database type= MySQL | hostname= your host | Port= 3306 | username =… | password=…| Database=…} then select the Table and click on Import (Fig. 5.26, 5.27, 5.28). Then close the dialog.



Import Ontology: km4 1.6.2.owl
Import Ontology: sch
Import Ontology: sko
Import Ontology: dcte
Import Ontology: foaf
Import Ontology: wgs

Database Type	Hostname	Port	Username	Password	Database
MySQL	192.168.0.20	3306	SiiMobility		SiiMobility

Fig. 5.26 Karma: Import MySQL table.

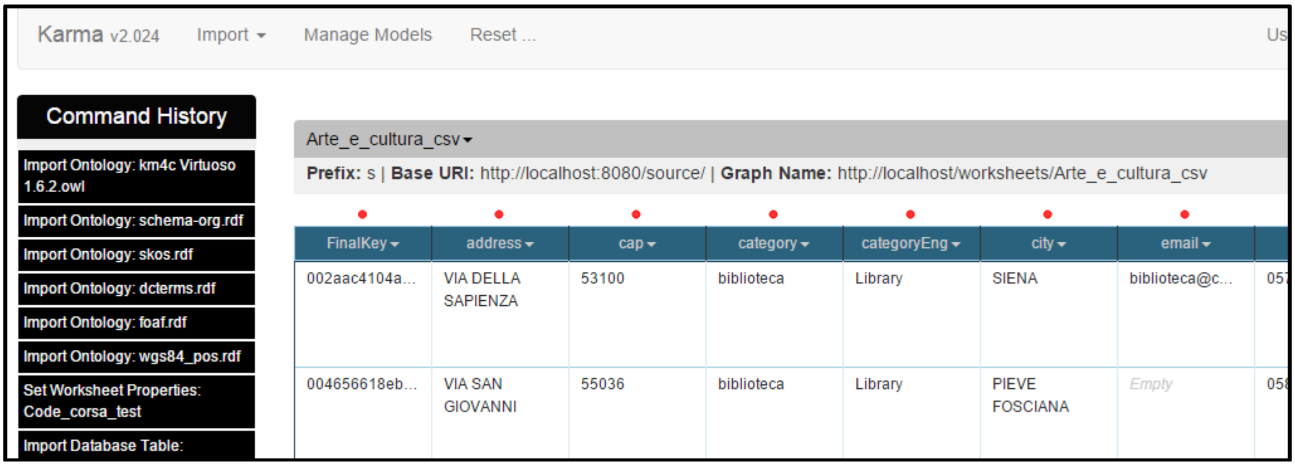


Import Database Table

Database Type	Hostname	Port	Username	Password	Database
MySQL	192.168.0.20	3306	SiiMobility	••••••••••	SiiMobility

1_Lotto_Regionale_f
1_Lotto_Regionale_p
5_Lotto_Metropolitano_Firenze_f
5_Lotto_Metropolitano_Firenze_p
5_Lotto_Metropolitano_p
5_Lotto_metropolitano_Firenze_f
Arte_e_cultura_csv
Assist_anziani_auto_kmz

Fig.5.27 Karma: Import MySQL table.



Import Ontology: km4c Virtuoso 1.6.2.owl
Import Ontology: schema-org.rdf
Import Ontology: skos.rdf
Import Ontology: dcterms.rdf
Import Ontology: foaf.rdf
Import Ontology: wgs84_pos.rdf
Set Worksheet Properties: Code_corsa_test
Import Database Table:

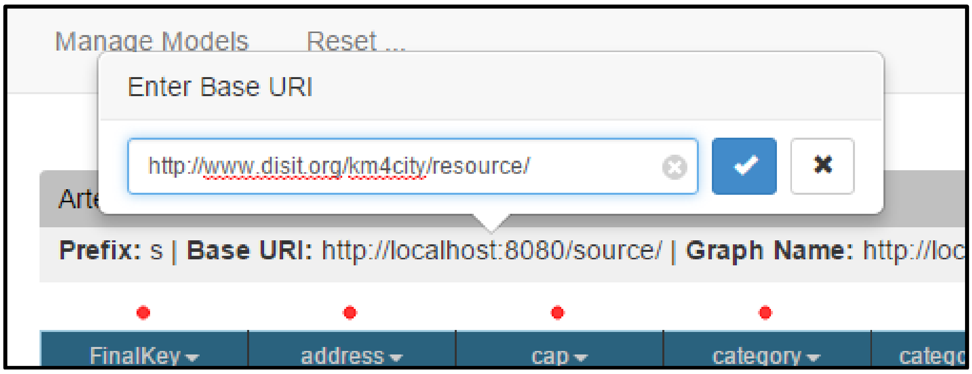
Arte_e_cultura_csv

Prefix: s | **Base URI:** http://localhost:8080/source/ | **Graph Name:** http://localhost/worksheets/Arte_e_cultura_csv

FinalKey	address	cap	category	categoryEng	city	email
002aac4104a...	VIA DELLA SAPIENZA	53100	biblioteca	Library	SIENA	biblioteca@c...
004656618eb...	VIA SAN GIOVANNI	55036	biblioteca	Library	PIEVE FOSCIANA	Empty

Fig. 5.28 Karma: MySQL table imported.

5. Click on ‘Base URI’ and set it to: http://www.disit.org/km4city/resource/(Fig. 5.29)



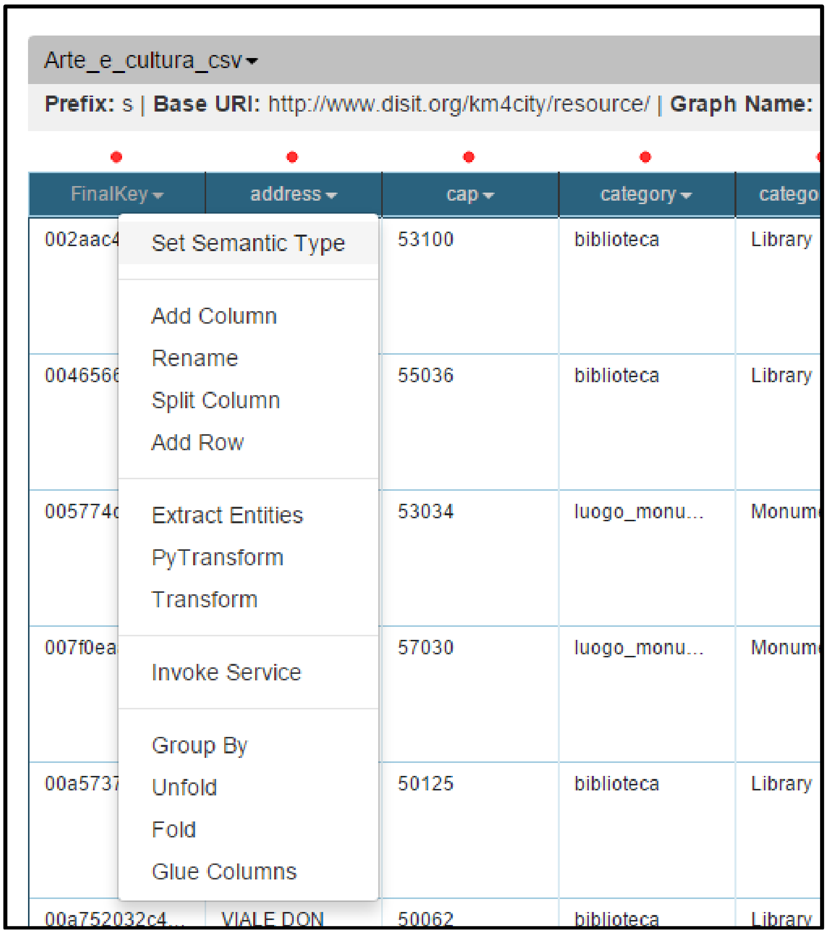
http://www.disit.org/km4city/resource/

Prefix: s | **Base URI:** http://localhost:8080/source/ | **Graph Name:** http://loc

Fig. 5.29 Karma: Base Uri setting.

6.2 Simple Mapping of the columns

At first, set the mapping for the key of the table (in this case the FinalKey) by choosing "Set Semantic Type" on the menu that is bound to the column (Figs. 5.30, 5.31)

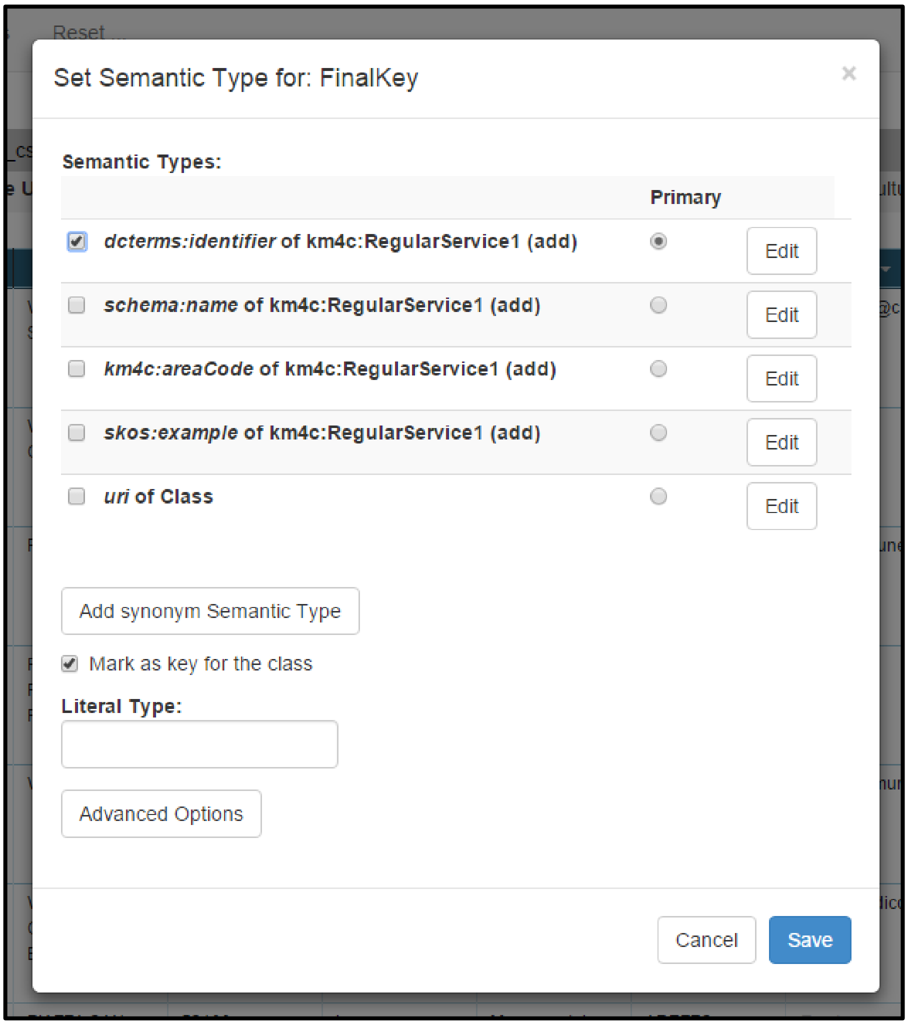


Arte_e_cultura_csv

Prefix: s | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name:**

FinalKey	address	cap	category	catego
002aac4		53100	biblioteca	Library
0046566		55036	biblioteca	Library
005774c		53034	luogo_monu...	Monum
007f0ea		57030	luogo_monu...	Monum
00a5737		50125	biblioteca	Library
00a752032c4...	VIALE DON	50062	biblioteca	Library

Fig. 5.30 Karma: Set Semantic Type.



Set Semantic Type for: FinalKey

Semantic Types:

	Primary	
☑ *dcterms:identifier* of km4c:RegularService1 (add)	●	Edit
☐ *schema:name* of km4c:RegularService1 (add)	○	Edit
☐ *km4c:areaCode* of km4c:RegularService1 (add)	○	Edit
☐ *skos:example* of km4c:RegularService1 (add)	○	Edit
☐ *uri* of Class	○	Edit

Add synonym Semantic Type

☑ Mark as key for the class

Literal Type:

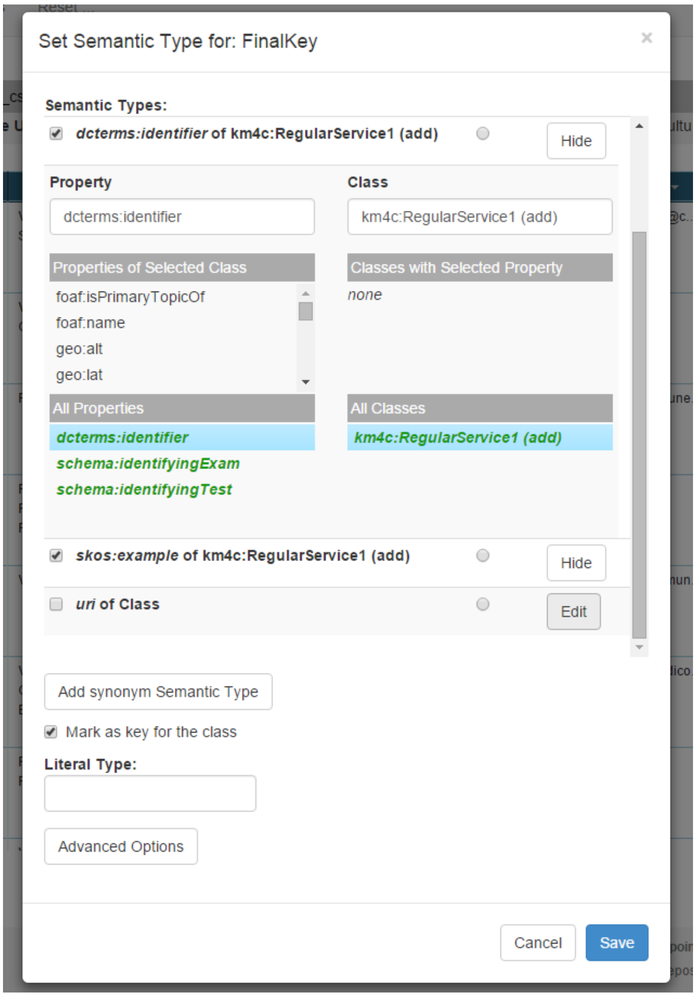
Advanced Options

Cancel Save

Fig. 5.31 Karma: Set Semantic Type, details.

Mappare la chiave come **dcterms:identifier** di una classe **km4c:RegularService** e selezionare "Mark as key of the class".

Se non fosse presente la proprietà *dcterms:identifier* fare click su Edit si una qualsiasi proprietà nell'elenco, quindi compare un elenco di properietà e classi da selezionare. Per facilitare la ricerca scrivere parte del nome della proprietà e della classe nei due campi e selezionare la proprietà e classe desiderati:



Set Semantic Type for: FinalKey

Semantic Types:

☑ ***dcterms:identifier*** **of km4c:RegularService1 (add)** ○ Hide

Property: dcterms:identifier

Class: km4c:RegularService1 (add)

Properties of Selected Class	Classes with Selected Property
foaf:isPrimaryTopicOf	*none*
foaf:name	
geo:alt	
geo:lat	

All Properties	All Classes
dcterms:identifier	***km4c:RegularService1 (add)***
schema:identifyingExam	
schema:identifyingTest	

☑ ***skos:example*** **of km4c:RegularService1 (add)** ○ Hide

□ ***uri*** **of Class** ○ Edit

Add synonym Semantic Type

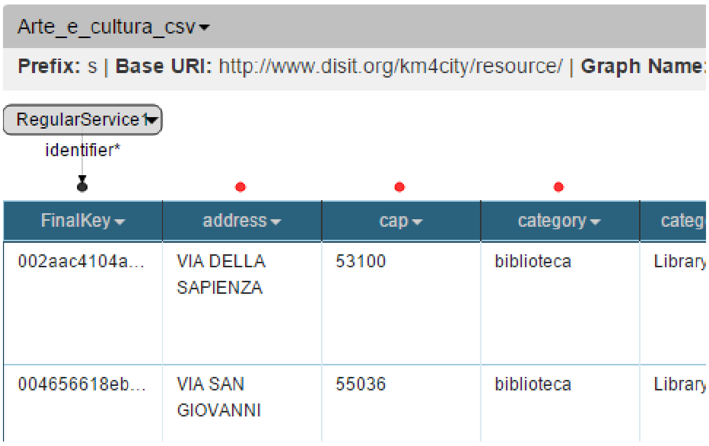
☑ Mark as key for the class

Literal Type:

Advanced Options

Cancel Save

Premedo Save si ottiene (dopo qualche secondo di attesa):



Arte_e_cultura_csv

Prefix: s | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name**:

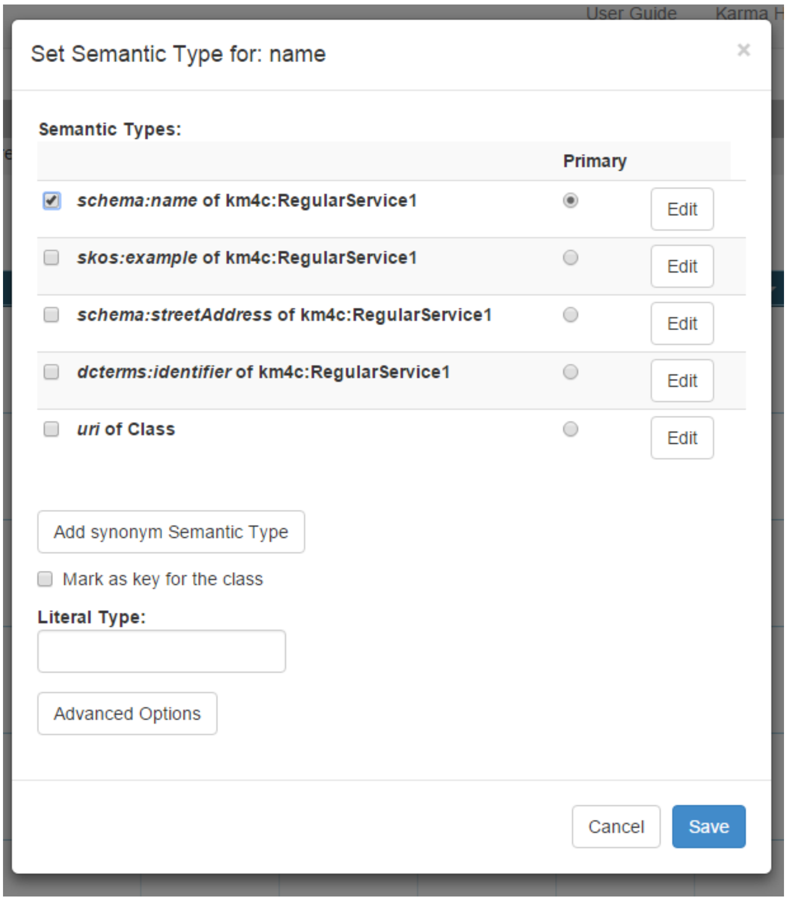
RegularService1

identifier*

FinalKey	address	cap	category	categ
002aac4104a...	VIA DELLA SAPIENZA	53100	biblioteca	Library
004656618eb...	VIA SAN GIOVANNI	55036	biblioteca	Library

L’asterisco dopo *identifier* indica che è il campo chiave e che viene usato per identificare la riga insieme alla Base URI e genera quindi la URL che identifica il record (nel primo caso sarà http://www.disit.org/km4city/resource/002aac41...). Se ci si dimentica di indicare il campo come chiave, si deve eliminare il mapping ed inserirlo nuovamente.

In modo analogo si fa il mapping del nome usando la proprietà **schema:name**:



Set Semantic Type for: name

Semantic Types:

		Primary	
☑	*schema:name* of km4c:RegularService1	●	Edit
□	*skos:example* of km4c:RegularService1	○	Edit
□	*schema:streetAddress* of km4c:RegularService1	○	Edit
□	*dcterms:identifier* of km4c:RegularService1	○	Edit
□	*uri* of Class	○	Edit

Add synonym Semantic Type

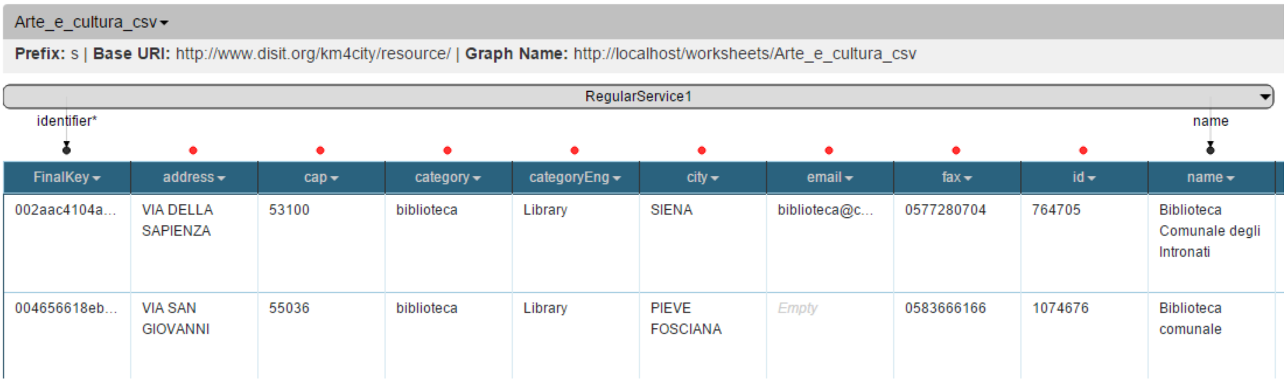
□ Mark as key for the class

Literal Type:

Advanced Options

Cancel Save

Si noti che in questo caso non c'è scritto (add) dopo il nome della classe in quanto si vuole associare il nome alla stessa entità definita per il mapping dell'identificatore. Premedo Save si ottiene:

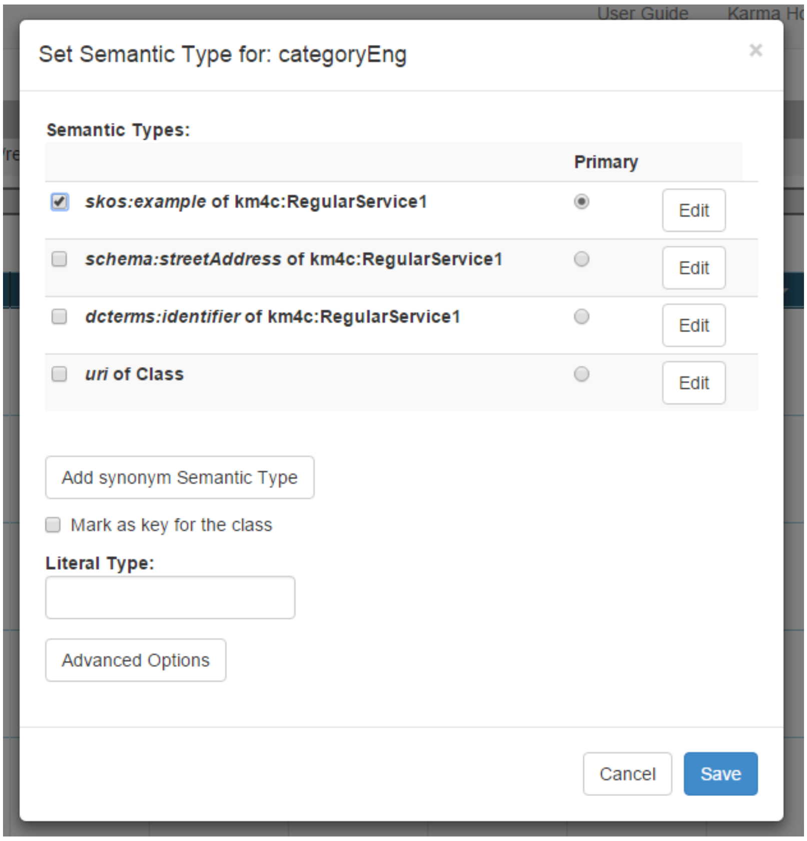

Arte_e_cultura_csv▾

Prefix: s | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name:** http://localhost/worksheets/Arte_e_cultura_csv

FinalKey▾	address▾	cap▾	category▾	categoryEng▾	city▾	email▾	fax▾	id▾	name▾
002aac4104a...	VIA DELLA SAPIENZA	53100	biblioteca	Library	SIENA	biblioteca@c...	0577280704	764705	Biblioteca Comunale degli Intronati
004656618eb...	VIA SAN GIOVANNI	55036	biblioteca	Library	PIEVE FOSCIANA	Empty	0583666166	1074676	Biblioteca comunale

Dato che Karma non permette di associare il tipo della classe sulla base dei dati presenti nella tabella (es. Library) ma solo staticamente al mapping (RegularService) per convenzione è stata usata la proprietà skos:example per associare il tipo, successivamente in fase di elaborazione delle triple generate verrà sostituito skos:example con rdf:type.

Quindi per associare il tipo di servizio si mappa la colonna **categoryEng** come skos:example:



Set Semantic Type for: categoryEng

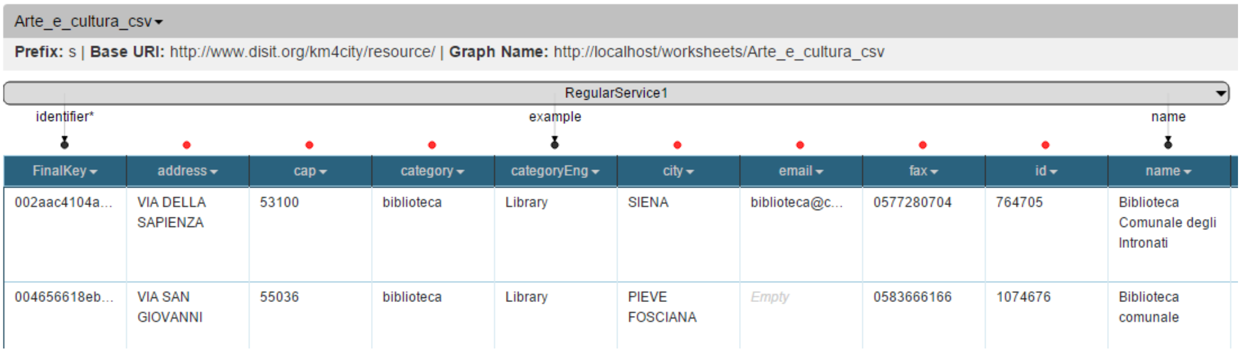
Semantic Types:

		Primary	
☑	***skos:example*** **of km4c:RegularService1**	●	Edit
□	***schema:streetAddress*** **of km4c:RegularService1**	○	Edit
□	***dcterms:identifier*** **of km4c:RegularService1**	○	Edit
□	***uri*** **of Class**	○	Edit

□ Mark as key for the class

Literal Type:

Ottenendo



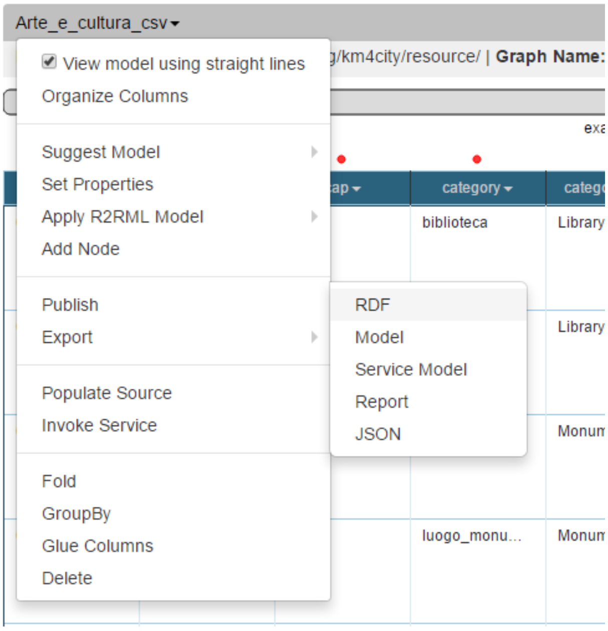
Arte_e_cultura_csv

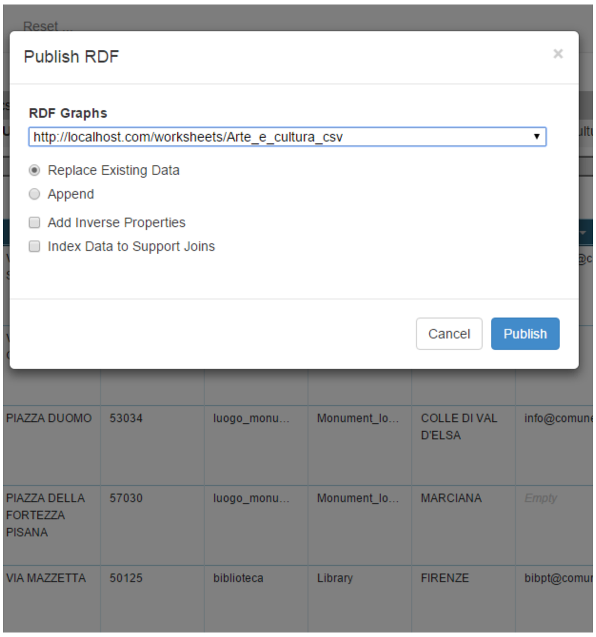
Prefix: s | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name:** http://localhost/worksheets/Arte_e_cultura_csv

FinalKey	address	cap	category	categoryEng	city	email	fax	id	name
002aac4104a...	VIA DELLA SAPIENZA	53100	biblioteca	Library	SIENA	biblioteca@c...	0577280704	764705	Biblioteca Comunale degli Intronati
004656618eb...	VIA SAN GIOVANNI	55036	biblioteca	Library	PIEVE FOSCIANA	Empty	0583666166	1074676	Biblioteca comunale

6.3 Verifica del mapping

Per verificare il mapping effettuato si può pubblicarlo, selezionando Publish/RDF sul menu associato alla tabella e indicando come RDF Graph il grafo relativo al dataset e indicando “Replace Existing Data” (se il grafo non fosse presente scegliere Create New Context).

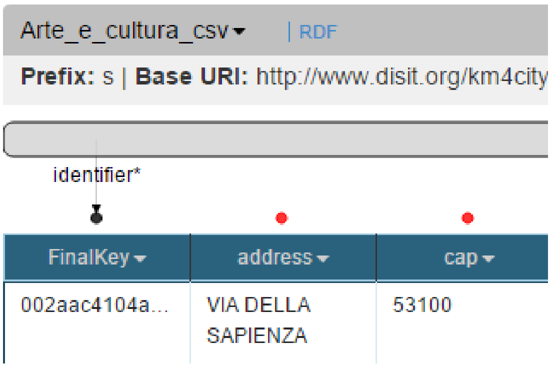



Publish RDF

RDF Graphs: http://localhost.com/worksheets/Arte_e_cultura_csv

● Replace Existing Data
○ Append
□ Add Inverse Properties
□ Index Data to Support Joins

Dopo aver premuto Publish compare accanto al nome della tabella il link RDF che mostra l'applicazione del mapping


Arte_e_cultura_csv | RDF

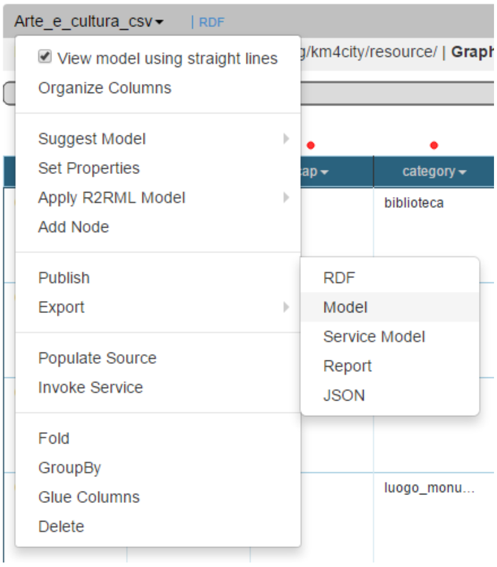
Prefix: s | Base URI: http://www.disit.org/km4city

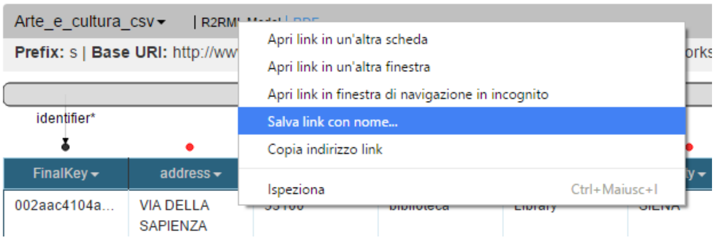
FinalKey	address	cap
002aac4104a...	VIA DELLA SAPIENZA	53100

6.4 Salvataggio del mapping

Per salvare il mapping si deve pubblicare il mapping e salvarlo in un file per poter essere applicato successivamente. Lo stesso mapping può essere usato anche per tabelle diverse con stesso nome delle colonne da mappare.

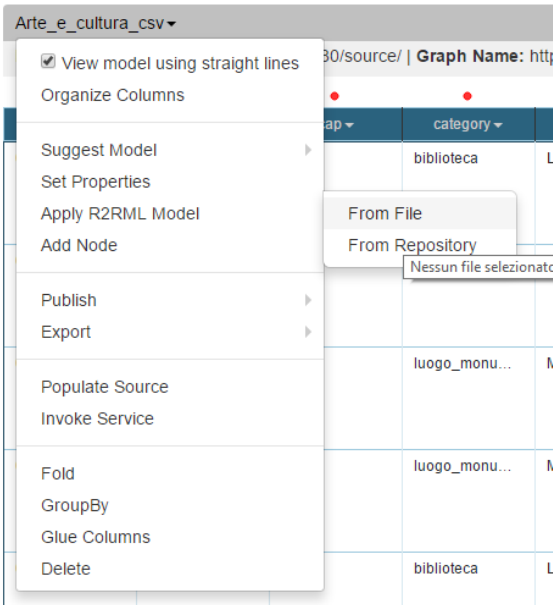
Sul menu associato alla tabella selezionare Publish/Model e dopo un poco compare link R2RML Model accanto al nome della tabella, salvando il link con nome si salva in locale il modello di mapping.



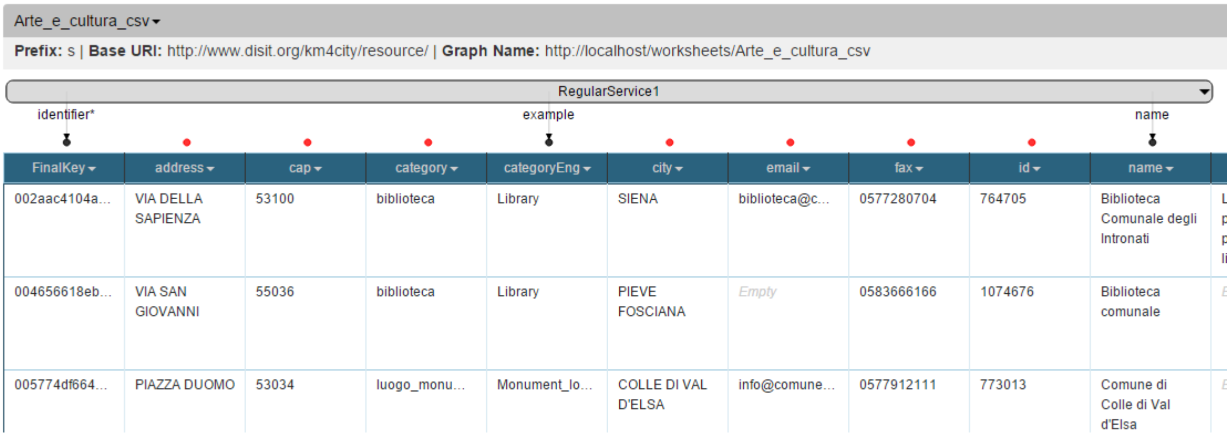


6.5 Caricamento mapping

Per caricare il mapping si devono prima caricare le ontologie usate e la tabella da mysql con i dati da mappare quindi nel menu della tabella selezionare **Apply R2RML Model/From File** quindi specificare il file contenente il modello.



E quindi il mapping viene ricaricato:



Prefix: s | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name:** http://localhost/worksheets/Arte_e_cultura_csv

FinalKey	address	cap	category	categoryEng	city	email	fax	id	name
002aac4104a...	VIA DELLA SAPIENZA	53100	biblioteca	Library	SIENA	biblioteca@c...	0577280704	764705	Biblioteca Comunale degli Intronati
004656618eb...	VIA SAN GIOVANNI	55036	biblioteca	Library	PIEVE FOSCIANA	Empty	0583666166	1074676	Biblioteca comunale
005774df664...	PIAZZA DUOMO	53034	luogo_monu...	Monument_lo...	COLLE DI VAL D'ELSA	info@comune...	0577912111	773013	Comune di Colle di Val d'Elsa

6.6 Guida Mapping

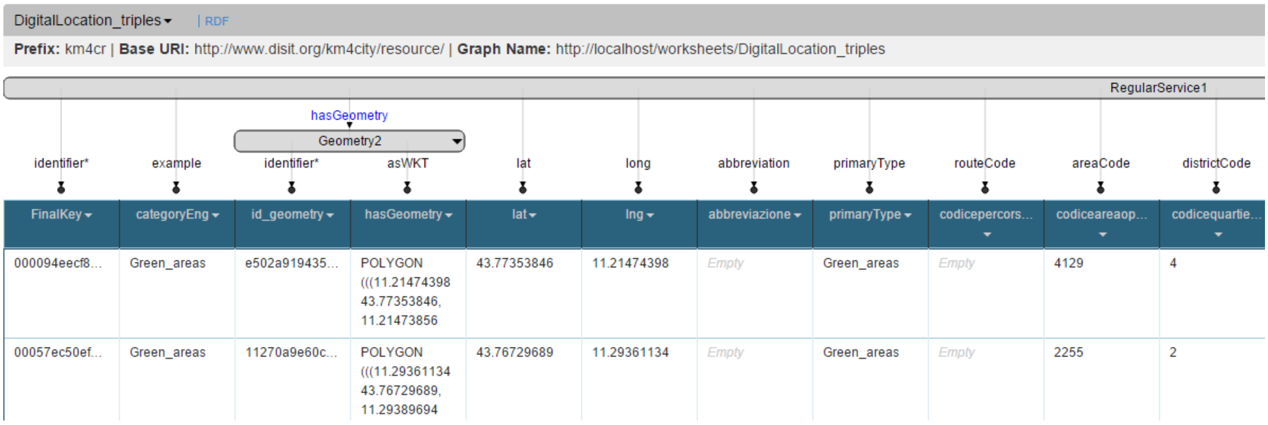
6.6.1 Mapping di campi semplici

Nella tabella seguente sono riportati alcuni casi di mapping di campi che sono mappati direttamente tramite una proprietà:

Nome Campo Tabella	**Esempio**	**Proprietà**
FinalKey	002aac4104a0304cd6c303a9aef4fa12	dcterms:identifier
name	Biblioteca Comunale degli Intronati	schema:name
address	VIA DELLA SAPIENZA	schema:streetAddress
cap	53100	schema:postalCode
city	SIENA	schema:addressLocality
province	SI	schema:addressRegion
civic	3	km4c:houseNumber
phone	0577280704	schema:telephone
fax	0577280704	schema:faxNumber
email	biblioteca@comune.siena.it	schema:email
notes	La biblioteca può vantare un patrimonio librario ricco ed eterogeneo: monografie e opuscoli, periodici, manoscritti, lettere autografe, disegni, stampe, incunaboli, fotografie, pellicole, microfilm e ...	skos:note
description		dct:description
website	http://www.biblioteca.comune.siena.it	schema:url
lat	43.760799672945794	geo:lat (literal xsd:float)
long	11.270243834110481	geo:long (literal xsd:float)
cod_toponimo	RT04801702206TO	km4c:isInRoad (come class uri)
codicequartiereopencms	5	km4c:districtCode
codiceareaopencms	817	km4c:areaCode

6.6.2 Mapping Geometry

Per il mapping delle geometrie si deve avere un mapping simile al seguente:



DigitalLocation_triples | RDF

Prefix: km4cr | **Base URI:** http://www.disit.org/km4city/resource/ | **Graph Name:** http://localhost/worksheets/DigitalLocation_triples

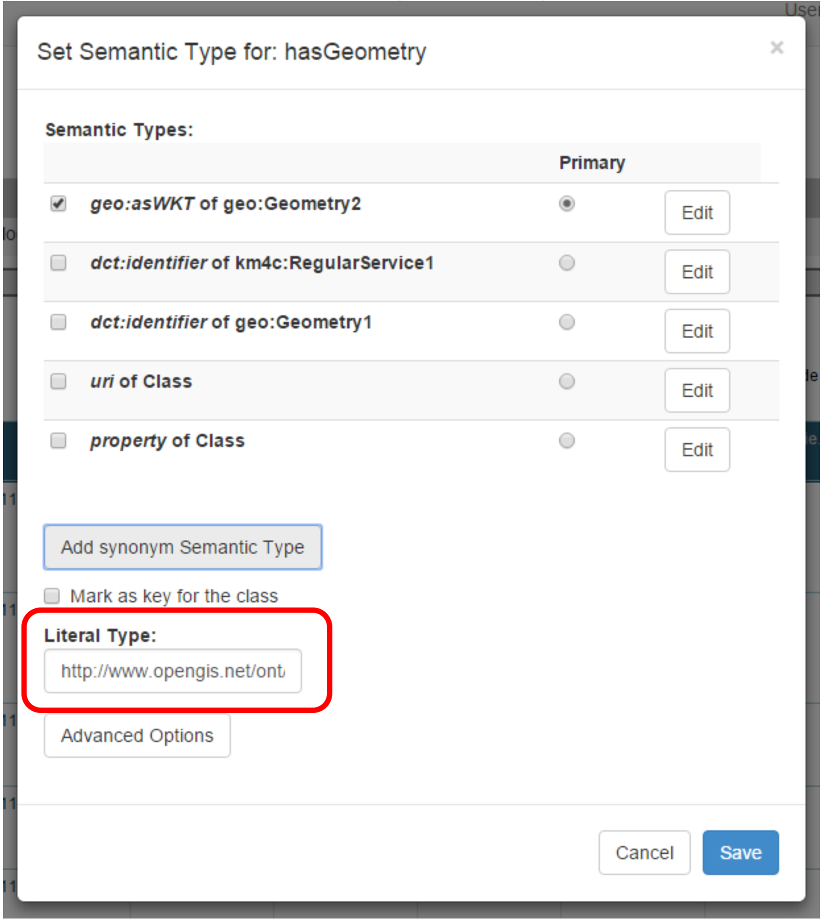
RegularService1

hasGeometry

Geometry2

identifier*	example	identifier*	asWKT	lat	long	abbreviation	primaryType	routeCode	areaCode	districtCode
FinalKey	categoryEng	id_geometry	hasGeometry	lat	lng	abbreviazione	primaryType	codicepercors...	codiceareaop...	codicequartie...
000094eecf8...	Green_areas	e502a919435...	POLYGON (((11.21474398 43.77353846, 11.21473856	43.77353846	11.21474398	Empty	Green_areas	Empty	4129	4
00057ec50ef...	Green_areas	11270a9e60c...	POLYGON (((11.29361134 43.76729689, 11.29389694	43.76729689	11.29361134	Empty	Green_areas	Empty	2255	2

Dove viene introdotta una nuova entità geo:Geometry che ha un identificatore (chiave) preso dal campo id_geometry ed il testo WKT associato tramite proprietà geo:asWKT che deve avere valore literal di tipo http://www.opengis.net/ont/geosparql#wktLiteral.



Set Semantic Type for: hasGeometry ×

Semantic Types:

		Primary	
☑	*geo:asWKT* of geo:Geometry2	●	Edit
□	*dct:identifier* of km4c:RegularService1	○	Edit
□	*dct:identifier* of geo:Geometry1	○	Edit
□	*uri* of Class	○	Edit
□	*property* of Class	○	Edit

Add synonym Semantic Type

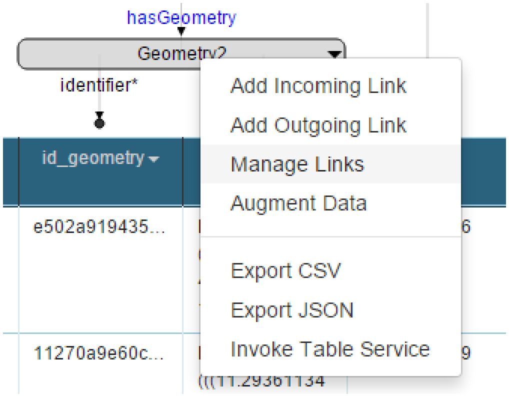
□ Mark as key for the class

Literal Type: http://www.opengis.net/ont/

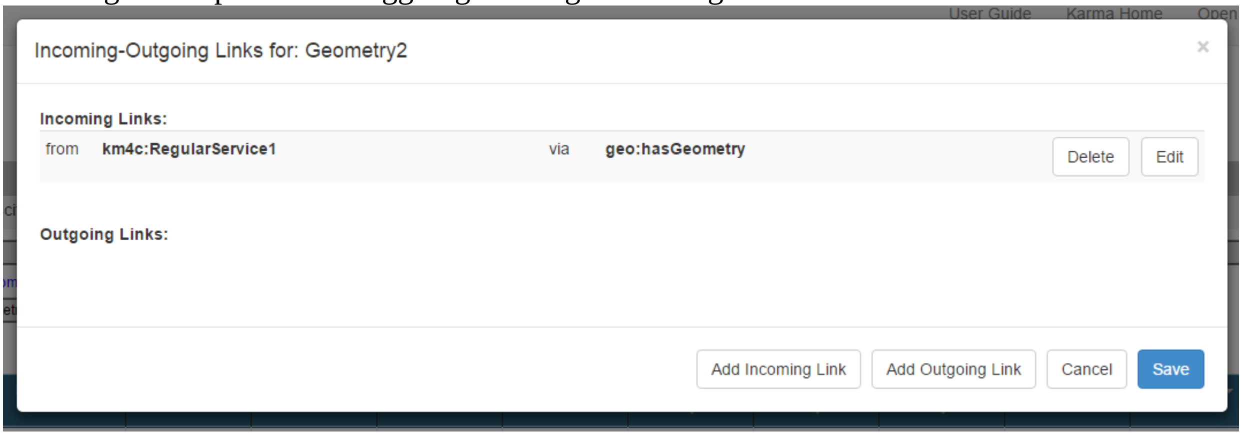
Advanced Options

Cancel Save

Il collegamento tra RegularService1 e Geometry2 tramite la proprietà geo:hasGeometry si ottiene tramite ManageLinks su Geometry2 (o RegularService1)



Il ManageLinks permette di aggiungere e togliere collegamenti con altre entità.



Incoming-Outgoing Links for: Geometry2

Incoming Links:

from **km4c:RegularService1** via **geo:hasGeometry**

Outgoing Links:

Il mapping di una riga produce:

```
<http://www.disit.org/km4city/resource/001c9cb6a1b0401c73481005de7ea0df> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.disit.org/km4city/schema#RegularService> .

<http://www.disit.org/km4city/resource/001c9cb6a1b0401c73481005de7ea0df>
<http://www.disit.org/km4city/schema#primaryType> "Wifi" .

<http://www.disit.org/km4city/resource/001c9cb6a1b0401c73481005de7ea0df> <http://purl.org/dc/terms/identifier>
"001c9cb6a1b0401c73481005de7ea0df" .

<http://www.disit.org/km4city/resource/001c9cb6a1b0401c73481005de7ea0df>
<http://www.w3.org/2003/01/geo/wgs84_pos#long> "11.2594299316406"^^<http://www.w3.org/2001/XMLSchema#float> .

<http://www.disit.org/km4city/resource/001c9cb6a1b0401c73481005de7ea0df>
<http://www.w3.org/2003/01/geo/wgs84_pos#lat> "43.7753715515137"^^<http://www.w3.org/2001/XMLSchema#float> .

<http://www.disit.org/km4city/resource/001c9cb6a1b0401c73481005de7ea0df>
<http://www.opengis.net/ont/geosparql#hasGeometry>
<http://www.disit.org/km4city/resource/4b80bf9f39794757b14f871cf32d0ca8> .

<http://www.disit.org/km4city/resource/4b80bf9f39794757b14f871cf32d0ca8> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.opengis.net/ont/geosparql#Geometry> .

<http://www.disit.org/km4city/resource/4b80bf9f39794757b14f871cf32d0ca8> <http://purl.org/dc/terms/identifier>
"4b80bf9f39794757b14f871cf32d0ca8" .

<http://www.disit.org/km4city/resource/4b80bf9f39794757b14f871cf32d0ca8>
<http://www.opengis.net/ont/geosparql#asWKT> "POINT (11.2594299316406
43.7753715515137)"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
```

7 DIM - Data Ingestion Manager

Data Ingestion Manager allows the creation of Open Data records, setup and management of the ingestion process. The ingestion process starts by collecting raw Open Data and ends with the generation of RDF Triples according to the domain ontology model adopted. The creation of Open Data record and properties allows the insertion and editing of an Open Data in the repository. Open Data are described by a set of properties to be set like: Name, Category, Resource, Source, Format,

Type (real-time or static) and more (see section 2.2 for a full list).
The setup and management of ingestion process allows selecting tasks to execute both in the creation step and during the life of data for update purposes. The following tasks are available and could be executed singularly or joined ("concatenate"):

- **Ingestion (I)**of the data instances performs the raw data retrieval from the source where the Open Data is stored.
- **Quality Improvement (QI)** task is focused on enriching the Open Data by adding for instance links to external Linked Open Data (LOD) or refining possible inconsistences.
- **Triples Generation (T)** performs thegeneration of RDF Triples by mapping static, dynamic data on the basis of the domain ontology model.
- **Validation(V)** of the Open Data detects possible inconsistencies, incompleteness, correctness of relationships, etc…
- **Reconciliation (R)** tasktries to solve the lack of coherence among indexed entities referring to the same concept but coming from different data sets.

8 DISCES - Distributed Smart City Engine Scheduler

A typical major requirement in a Smart City/Cloud environment consists of an engine for distributed task scheduling. In this context, DISIT lab developed an efficient solutions for Smart management and scheduling, Distributed SCE Scheduler, DISCES. DISCES consists of a set of distributed instances of running agents performing concurrent tasks. DISCES engine with cluster functionality allows adding distributed nodes and defining jobs, without service downtime.

www.disit.org/drupal/?q=node/6746

9 Bibliografia

- [Amdocs] Amdocs Billing, http://www.amdocs.com
- [Aoki et al., 2009] Aoki, P. M., Honicky, R. J., Mainwaring, A., Myers, C. (2009). A vehicle for research: using street sweepers to explore the landscape of environmental community action, ACM HFCS'09.
- [Armoni, 2002] Armoni A., " Data Security Management in Distributed Computer Systems ", Informing Science. Data Security. 2002, Vol. 5, N. 1, Paper Editor: Lech Janczewski
- [Bellini et al., 2013] Bellini P., Di Claudio M., Nesi P., Rauch N., "Tassonomy and Review of Big Data Solutions Navigation", Big Data Computing To Be Published 26th July 2013 by Chapman and Hall/CRC
- [Karma] http://usc-isi-i2.github.io/karma/ https://github.com/usc-isi-i2/Web-Karma/wiki
- [HBase] https://archive.apache.org/dist/hbase/hbase-0.90.5
- [Penthao] http://community.pentaho.com/projects/data-integration
- [ETLGuide] http://www.disit.org/drupal/?q=node/6975
- [Guide_VM] http://www.disit.org/drupal/?q=node/6690
- [Oracle] http://www.oracle.com/technetwork/java/javase/downloads/index.html
- [Ubuntu] https://help.ubuntu.com/community/Java
- [PDI] http://sourceforge.net/projects/pentaho/files/Data%20Integration
- [XAMPP] http://wiki.ubuntu-it.org/Server/Xampp
- [HBase] https://archive.apache.org/dist/hbase/hbase-0.90.5
- [HRider] https://github.com/NiceSystems/hrider/wiki
- [Karma] https://github.com/usc-isi-i2/Web-Karma/wiki
- [UNIFI] http://www.unifi.it
- [UNIFISchools] http://www.unifi.it/vp-10306-schools.html

- [UNIFIDep] http://www.unifi.it/cmpro-v-p-10305.html
- [QGIS] https://www.qgis.org/it/site
- [MMQGIS] http://michaelminn.com/linux/mmqgis
- [GOOGLEAPI] https://developers.google.com/maps/documentation/geocoding/intro
- [km4CitySchema] http://www.disit.org/km4city/schema

10 Acronimi

- API: Application Program Interface
- AVL: Automatic vehicle location
- AVM: Automatic Vehicle Monitoring
- BDaaS: Big Data as a Service
- CAP principle: Consistency Availability Partition Tolerance principle
- CBB: Content Based Billing
- CBB: Content Based Billing
- CEN: European Committee for Standardization
- DBMS: database management system
- ETL: Extract – Transform - Load
- FCD: Floating Cellular Data
- GPRS: General packet radio service
- GPS: Global positioning System
- GSM: Global System for Mobile
- ICT: Information and Communication Technologies
- ITS: Intelligent Transport Systems
- LCD: liquid-crystal display
- LOD: linked open data
- MC: Mobile Collector
- MMS: Multimedia Messaging Service
- NLP: Natural Language Processing
- NoSQL: no SQL database
- OD: open data
- OD: Open Data
- OGC: Open Geospatial Consortium
- OWL: Web Ontology Language
- PA: Pubblica Amministrazione
- PMI: Piccola e Media Impresa
- PMS: Private Mobile Systems
- POS: part-of-speech
- RDF: Resource Description Framework
- RFID: Radio Frequency IDentification o Identificazione a radio frequenza
- RTTI: Real-time Travel & Traffic Information
- SDI: Spatial Data Infrastructures
- SII: sistema di interoperabilità integrato
- SIMONE: progetto Simone
- SMS: Short Message Service
- SN: social networking, oppure sensor network
- SOA: Service Oriente Architecture
- SOAP: Simple Object Access Protocol
- SSAMM: Agenzia per la Mobilità Metropolitana strumenti di supporto, TOSCANA
- TPEG: Transport Protocol Experts Group

- TPL: gestore trasporto pubblico locale
- UML: Unified Modeling Language
- UMTS: Universal Mobile Telecommunications System
- UTC: Urban Traffic Control
- UUDI: Universal Description Discovery and Integration
- V2I: Vehicle-to-Infrastructure
- V2V: vehicle-to-vehicle
- VMS: Variable Message Sign
- VWSN: Vehicular Wireless Sensor Networks
- W3C: World Wide Web Consortium
- WSD: Word Sense Disambiguation
- WSDL: Web Services Description Language
- WSN: Wireless Sensor Networks
- XMI: XML Metadata Interchange standard di OMG
- XML: Extensible Markup Language
- ZTL: Zona a Traffico Limitato