

# Smart City: data ingestion and mining

Parte 12 (2015-2016) of  
Course on KNOWLEDGE MANAGEMENT AND PROTECTION  
SYSTEMS

*Giacomo Martelli, Mariano di Claudio*

**DISIT Lab**, Dipartimento di Ingegneria dell'Informazione, DINFO

Università degli Studi di Firenze

Via S. Marta 3, 50139, Firenze, Italy

Tel: +39-055-2758515, fax: +39-055-2758570

<http://www.disit.dinfo.unifi.it> *alias* <http://www.disit.org>  
[giacomo.martelli@unifi.it](mailto:giacomo.martelli@unifi.it)

Prof. Paolo Nesi, [paolo.nesi@unifi.it](mailto:paolo.nesi@unifi.it)



# Index

## 1. Big Data: from Open Data to Triples

## 2. ETL process

## 3. ETL tool: Pentaho Data Integration (PDI)

- Features
- Key concepts
- Examples



# Index

## 4. Developing ETL processes with PDI

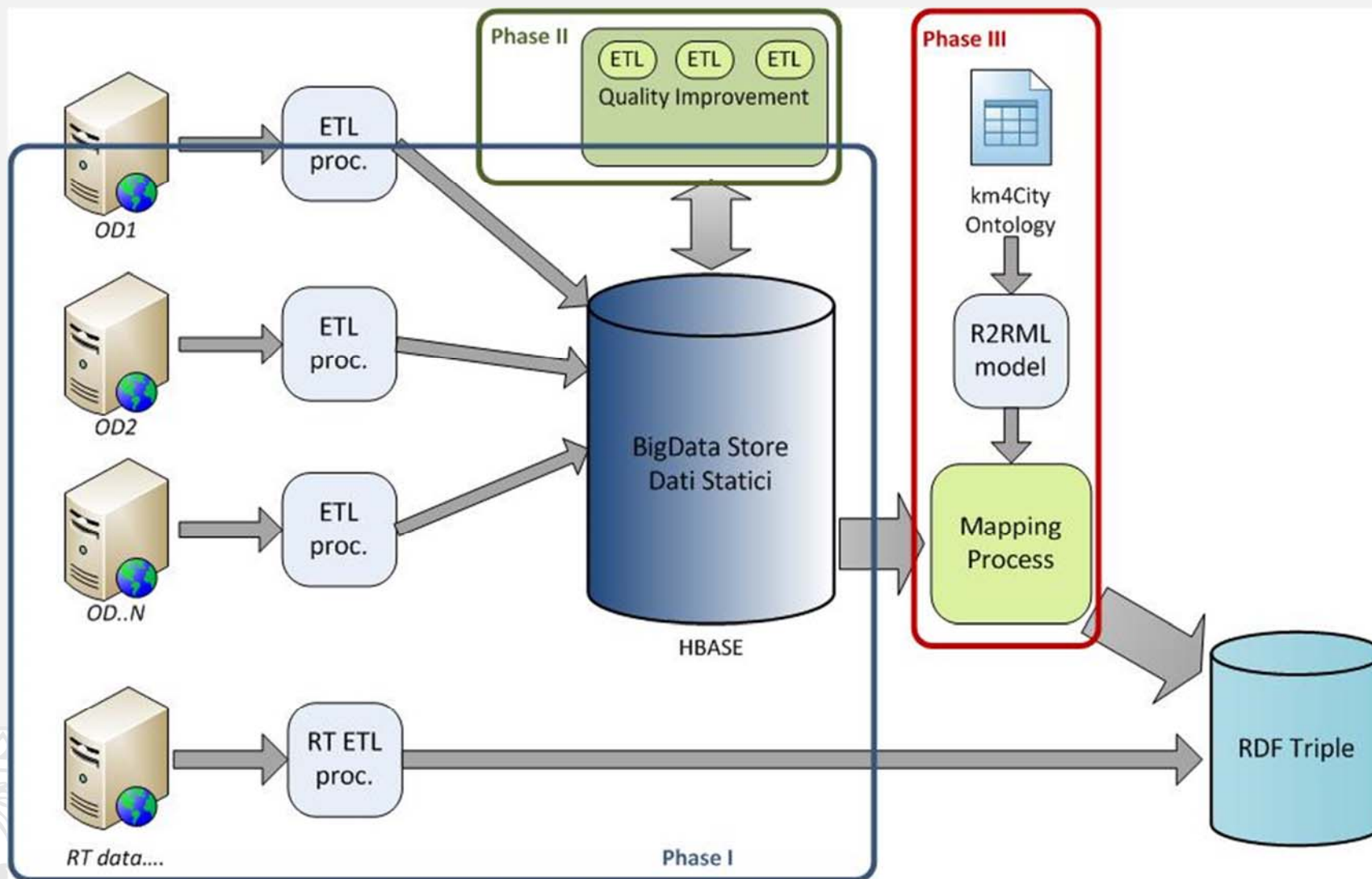
- Tool installation & configuration
- SiiMobility Project
- ETL processes implementation (within the SiiMobility Project)



1

# Big Data: from Open Data to Triples

# Data Engineering Architecture



# Phase I: Data Ingestion

- **Ingesting a wide range of OD/PD:** open and private data, static, quasi static and/or dynamic real time data.
- **Static and semi-static data** include: points of interests, geo-referenced services, maps, accidents statistics, etc.
  - files in several formats (SHP, KML, CVS, ZIP, XML, etc.)
- **Dynamic data** mainly data coming from sensors
  - parking, weather conditions, pollution measures, bus position, etc..
  - using Web Services.
- Using **Pentaho - Kettle** for data integration (Open source tool)
  - using specific **ETL** Kettle transformation processes (one or more for each data source)
  - data are stored in HBase (Bigdata NoSQL database)

# Phase II: Data Quality Improvement

- **Problems kinds:**
  - Inconsistencies, incompleteness, typos, lack of standards, multiple standards, ..
- **Problems on:**
  - CAPs vs Locations
  - Street names (e.g., dividing names from numbers, normalize when possible)
  - Dates and Time: normalizing
  - Telephone numbers: normalizing
  - Web links and emails: normalizing
- **Partial Usage of**
  - Certified and accepted tables and additional knowledge

# Phase III: Data mapping

- Transforms the data from HBase to RDF triples
- Using **Karma Data Integration tool**, a mapping model from SQL to RDF on the basis of the ontology was created
  - Data to be mapped first temporary passed from Hbase to MySQL and then mapped using Karma (in batch mode)
- The mapped data in triples have to be uploaded (and indexed) to the **RDF Store** (Sesame with OWLIM-SE / Virtuoso)





2

# ETL Process



# Useful tools

## Pre-processing data to RDF triples generation: ETL (Extract, Transform and Load)

- Process used in database and data warehousing that involves three phases.
- Useful tools to prepare data for the following data analysis phase and eventual translation into RDF.
- Translation in **RDF Triples** is based on use of a specific **ontology**.
  - Definition of mapping models from SQL to RDF
  - The triples generated are loaded on RDF store.

# ETL Process

The three phases are:

- **Extracting** data from outside sources (**Ingestion** phase).
- **Transforming** it to fit operational needs, which can include quality levels (**Data Quality Improvement** phase).
- **Loading** it into the end target (database, operational data store, data warehouse, data mart, etc.....). So the data can be translated in **RDF triples using a specific ontology.**

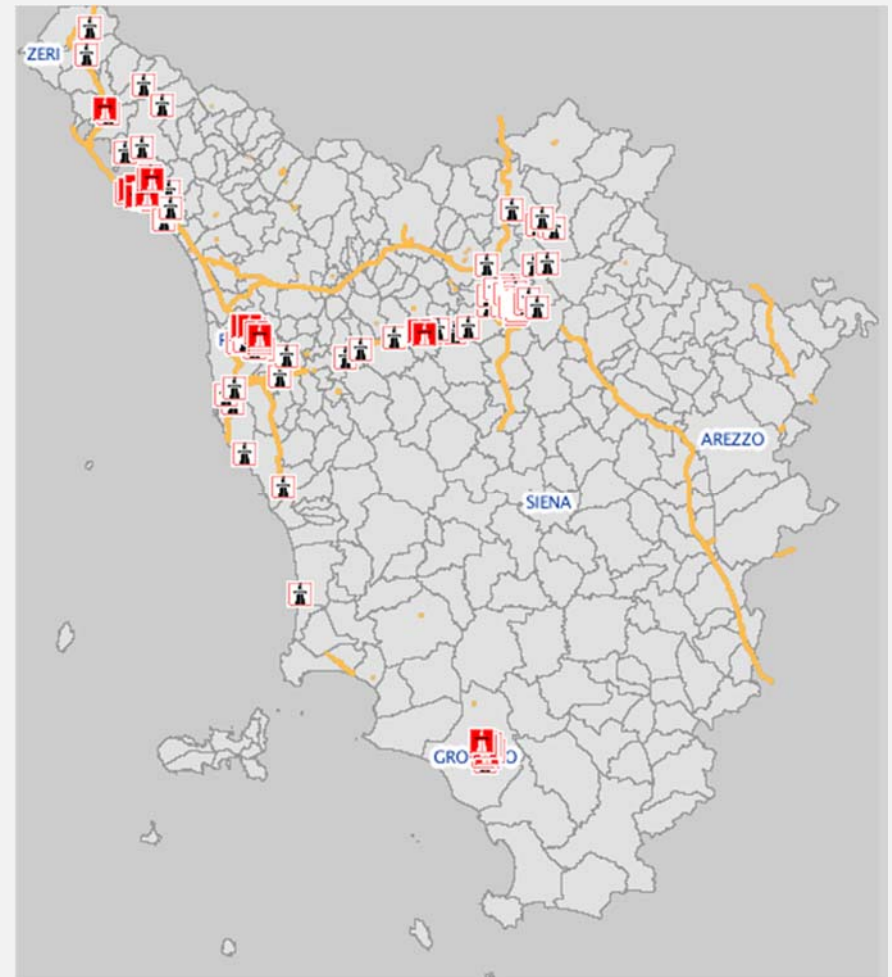
# Dataset

- **Mobility data** are made available by MIIC (Mobility Information Integration Center)
  - MIIC is a project of the Tuscany regional authority that deals with the collection of infomobility data (from federal authorities) and their distribution via web services.
  - Web services expose data about: traffic, parking, AVM (Automatic Vehicle Monitoring), emergencies and weather information.



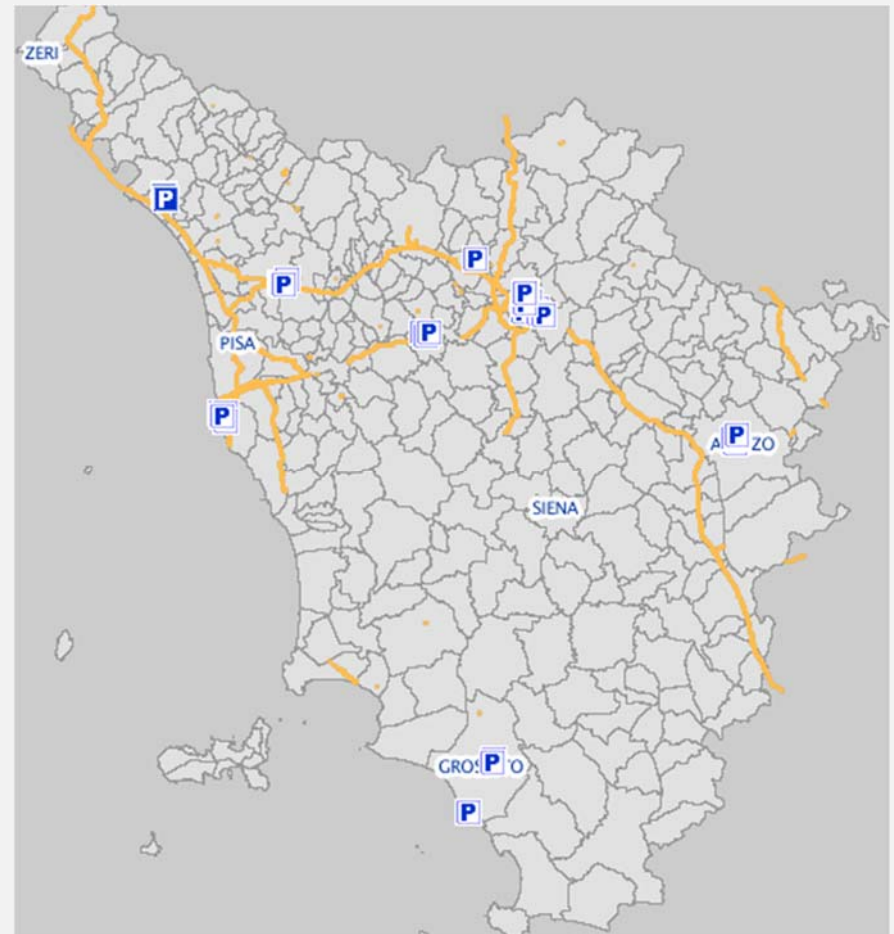
# Dataset: MIIC

- **Traffic sensors:** data about the situation of the traffic report from sensor detection systems operators
  - The measurements include data such as average distance between vehicles, average speed of transit, percentage of occupancy of the road, transit schedules, etc.
  - The sensors are divided into groups identified by a catalog code that is used when invoking the web service
  - A group is a set of sensors that monitors a road section
  - The groups produce a measurement every 5 or 10 minutes



# Dataset: MIIC

- **Parking:** data on the status of occupancy of the parking from parking areas operators.
  - The status of a parking is described by data such as the number of places occupied, the total number of vehicles in and out, etc.
  - The parking are divided into groups identified by a catalog code that is used when invoking the web service
  - A group corresponds to the collection of parking owned by a municipality
  - The situation of each parking is published approximately every minutes



# Dataset: MIIC

- **AVM:** real-time data about local public transport of Florence metropolitan area equipped with AVM devices
  - Monitors the status of active rides in the territory, where a ride is the path that a vehicle runs from a start point to the end
  - The data provided are related to delay or advance state of a vehicle, location vehicle in GPS coordinates, information about the last stop made and programmed, etc.
  - The web service is invoked passing the identification code of the race as a parameter.
  - AVM devices send two types of messages, one at a programmed time, usually every minute, and one at a major event like the arrival at a stop, departure from a stop or interruption of service.

# Dataset: MIIC

- **Static data:** data updated infrequently that can enrich those in real time.
  - Are provided by the regional observatory for mobility and transport through a portal with graphical user interface.
  - Positional information about parking and sensors surveyed by MIIC .
  - Additional details on public transport network: description and details of lines, routes and stops, Geolocation with Gauss-Boaga coordinates of stops





# Dataset

- **Weather forecasts** provided by LaMMA
  - XML Format
  - Information about current day
  - Weather on the current day and the next 4 days
  - Forecast on five times of the day: morning, afternoon ...
- **Services of the Tuscany region**
  - CSV Format
  - Various services: banks, schools, food, hospitals, shops, theatres, museums and. ..
  - Geolocalited by address (Street, house number) and municipality of belonging
  - Contains the service name, address, city, State, type of service, phone number, email ...

# Dataset

- **Statistics on the Florence municipality**
  - CSV Format
  - Contain information on the town and on the streets of Florence: crashes, tourist arrivals, circulating vehicles, etc..
  - The statistics shall cover the last five years

- **Tram line**

- KMZ Format
- Contains the KML file format used for geospatial data managing in Google earth and Google maps
- Contains the coordinates of the path covered by tram line



# Dataset

- **Events** of Florence municipality
  - JSON Format
  - Contain information about exhibitions, theater performances, sporting and cultural events ....
  - Dataset updated daily
- **Digital Location** of Florence municipality
  - CSV Format
  - Regroups 39 different categories of services such as WiFi hot spots, museums, green areas, gardens, cycle paths or tourist trails

3

# ETL tool: Pentaho Data Integration (PDI)

## FEATURES

# Pentaho Data Integration (PDI)

- **Pentaho** is a framework that contains several packages integrated to allow complete management:
  - *Business Intelligence problems;*
  - *Data Warehouse problems;*
  - *Big Data problems.*
- **Kettle** is the ETL component Pentaho for data transfer and processing.



# Pentaho Data Integration (Kettle)

- Free, **open source** (LGPL) ETL (Extraction, Transformation and Loading) tool.
  - It is available also in **enterprise version**.
- **Developed in Java**, therefore is guaranteed the compatibility and portability with the major operating systems (Windows, Linux, OS X..).
- **Powerful** Extraction, Transformation and Loading (ETL) capabilities.

# Pentaho Data Integration (Kettle)

- **Scalable**, standards-based architecture.
- Opportunity to interfacing with the main NoSQL Databases (HBase, Cassandra, MongoDB, CouchDB...).
- It uses an innovative, **metadata-driven** approach.
- Graphical, **drag and drop** design environment.

# Pentaho Data Integration (Kettle)

Main strengths:

- Collect data from a **variety of sources** (extraction);
- Move and modify data (transport and transform) while cleansing, denormalizing, aggregating and enriching it in the process;
- Frequently (daily) store data (loading) in the final target destination, usually a **large dimensionally modeled database (or data warehouse)**.



# Pentaho Data Integration (Kettle)

Main weakness:

- Kettle is not able to transform data into RDF triples, therefore it is necessary use other tools at a later stage (Karma).



# Kettle's 4 main programs

- **Spoon:** graphically oriented end-user tool to model the **flow of data** from input through transformation to output (**transformation**).
- **Pan** is a **command line tool** that executes transformations modeled with Spoon.
- **Chef:** a graphically oriented **end-user tool** used to model **jobs** (transformations, FTP downloads etc. placed in a flow of control).
- **Kitchen** is a **command line tool** to execute jobs created with Chef.

# Kettle's 4 main programs

- Interesting feature: Kettle is **model-driven**.
- **Spoon** and **Chef** have a graphical user interface to define the ETL processes on a **high level**.
- **Pan** and **Kitchen** can read and interpret the models created by Spoon and Chef respectively.
- Models can be saved to a particular **XML format**, or they can be stored into a relational database (**repository**).
- Handling many models with repository: models are stored in a structured manner, arbitrary queries can be written against the repository.

3

# ETL tool: Pentaho Data Integration (PDI)

## KEY CONCEPTS

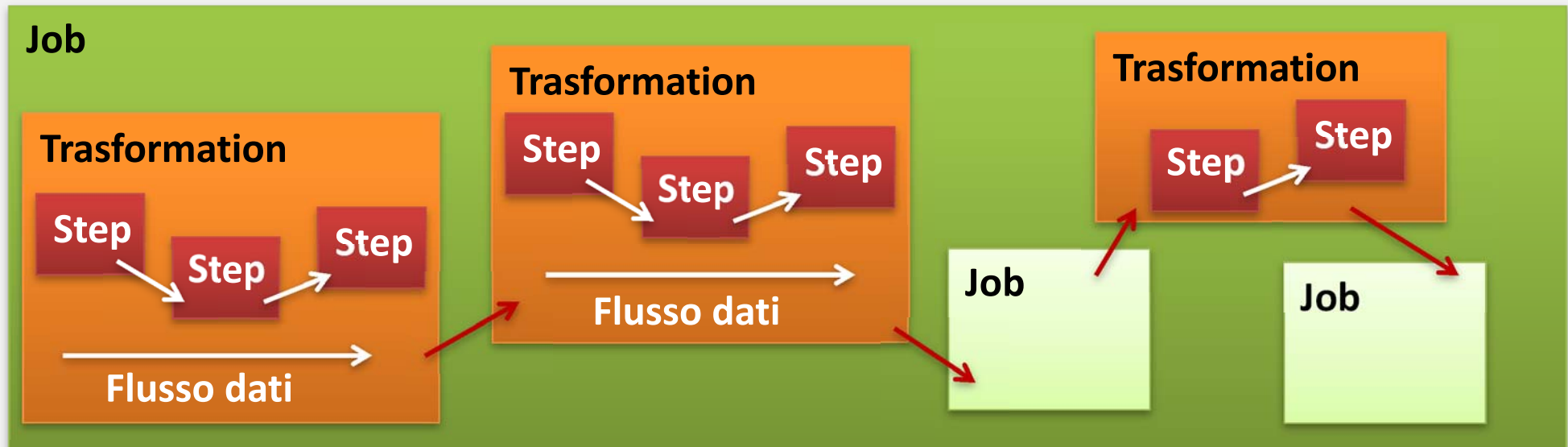
# Kettle: Concepts

- Kettle is based on two key concepts (from operating point of view):
  - **Job** (with extension “.kjb”);
  - **Transformation** (with extension “.ktr”), composed of several **steps**.
- Kettle’s key components are:
  - **Spoon** for ETL process modeling;
  - **Pan** to execute the transformations from command line;
  - **Kitchen** to execute the Job from command line.



# Kettle: Operational structure

Kettle operating components are organized as follows:



- The data are seen as rows flow from one step to another one.
- The steps are parallel executed on separated threads and there is no necessarily a beginning or end point of transformation.
- A job manages the sequential execution of lower-level entities: transformations or other jobs.

# Spoon Concepts: Steps and hoops

- One **step** denotes a particular kind of **action** that is performed **on data**.
- **Hops** are links to connect steps together and allow data to pass from one step to another.
- Steps are easily created by **dragging** the icon from the treeview **and dropping** them on the graphical model view.
- Kettle provides a lot of different step types, and can be **extended with plugin**.

# Type of Steps in Spoon (1/2)

Three different kinds of steps: **input**, **transform**, **output**.

- **Input steps** process some kind of 'raw' resource (file, database query or system variables) and create an output stream of records from it.
- **Output steps** (the reverse of input steps): accept records, and store them in some external resource (file, database table, etc.).



# Type of Steps in Spoon (2/2)

- **Transforming steps** process input streams and perform particular action on it (adding new fields/new records); This produce one or more output streams. Kettle offers many transformation steps out of the box, very simple tasks (renaming fields) and complex tasks (normalizing data, maintaining a slowly changing dimension in a data warehouse).
- **Main.kjb** is usually the primary job.

# Kettle: Spoon

To run Spoon, just launch the instruction `./spoon.sh` from command line.

The screenshot displays the Kettle Spoon interface for a job named "Modify\_website". The main workspace shows a flowchart with the following steps:

- Get rows from result
- Regex Evaluation
- Microsoft Excel Output
- Filter rows
- Microsoft Excel Output 2 2
- Microsoft Excel Output 2 2 2
- Modified Java Script Value 2
- Select values 3
- Check double web site
- Regex Evaluation 2
- Filter rows 2
- Microsoft Excel Output 2 4
- Modified Java Script Value
- Select values 4

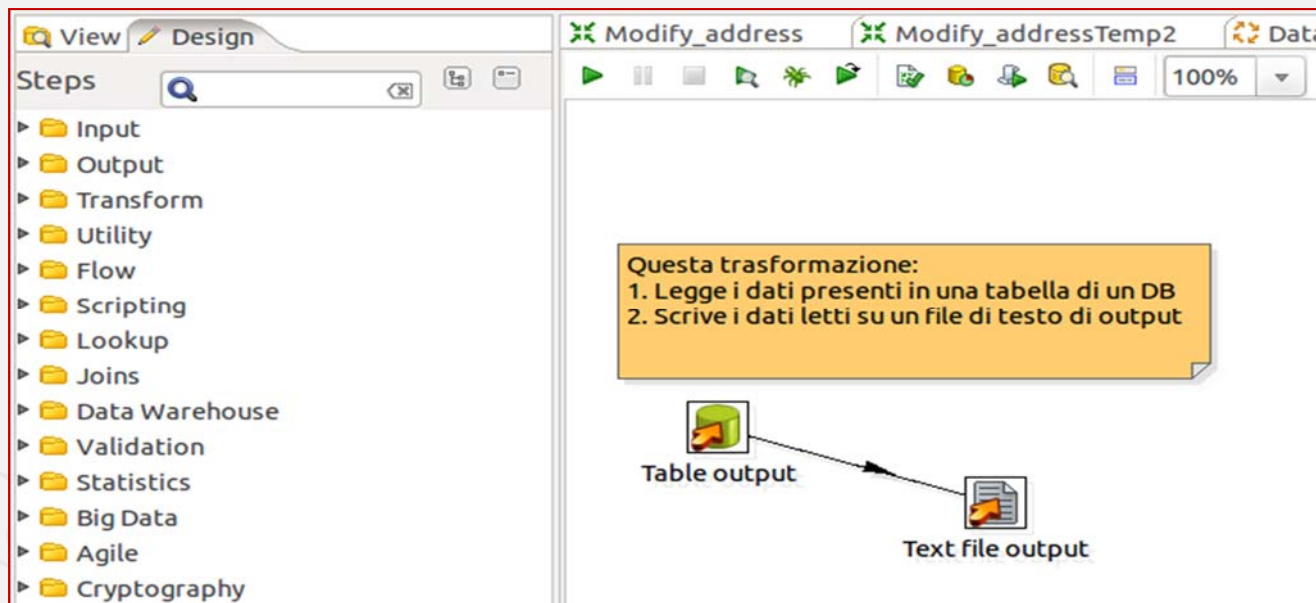
The "Execution Results" panel at the bottom shows the following log entries:

```

2014/09/11 13:11:10 - Spoon - starting job...
2014/09/11 13:11:26 - Spoon - Job has ended.
2014/09/11 16:56:16 - Spoon - Transformation opened.
2014/09/11 16:56:16 - Spoon - Launching transformation [Modify_website]...
2014/09/11 16:56:16 - Spoon - Started the transformation execution.
2014/09/11 16:56:17 - Modify_website - Dispatching started for transformation [Modify_website]
2014/09/11 16:56:17 - Data Grid.0 - Finished processing (I=0, O=0, R=0, W=6, U=0, E=0)
2014/09/11 16:56:18 - Spoon - The transformation has finished!!
  
```

# Kettle: Transformations

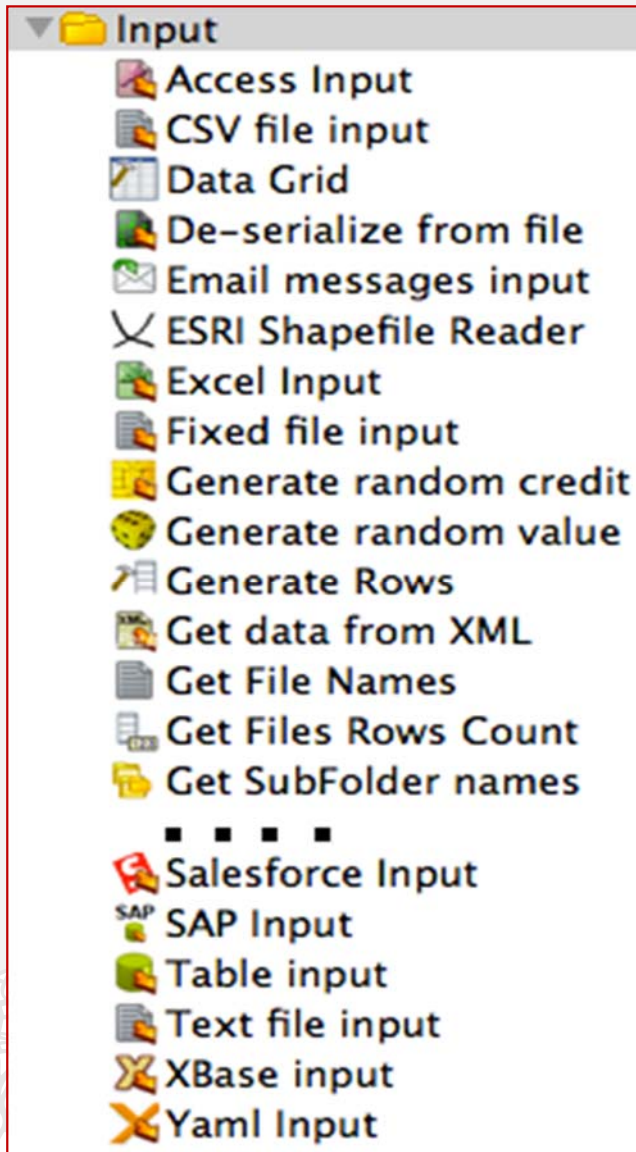
- Transformations define how the data must be collected, processed and reloaded.
- Consist of a series of step connected by links called Hop.
- Typically a transformation has one **input step**, one or multiple **transformation steps** and one or more **output step**.



# Kettle: Transformations

- There are several possible steps organized by type: ***Input, Output, Utility, Scripting, Flow, Validation, Lookup, Statistics...etc.***
- Each type of step, in turn, offers several of possibilities.
- For example, an input step can take data from different sources:
  - ***From a table of a relational database;***
  - ***From CSV file;***
  - ***From MS-Excel sheet.***
- The Hops between two steps don't define the execution sequence but represent the data flow and allow passing the content of a Field from one step to the next one.

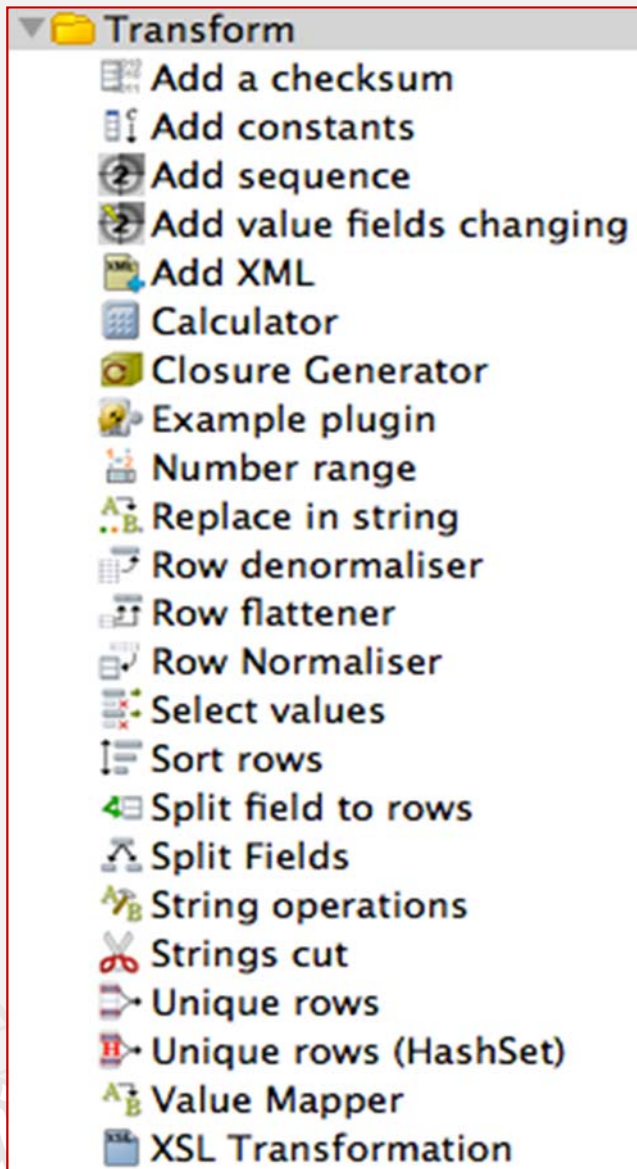
# Kettle: Transformations



## Input

Collection of step dealing with input data management. They are present in various types.

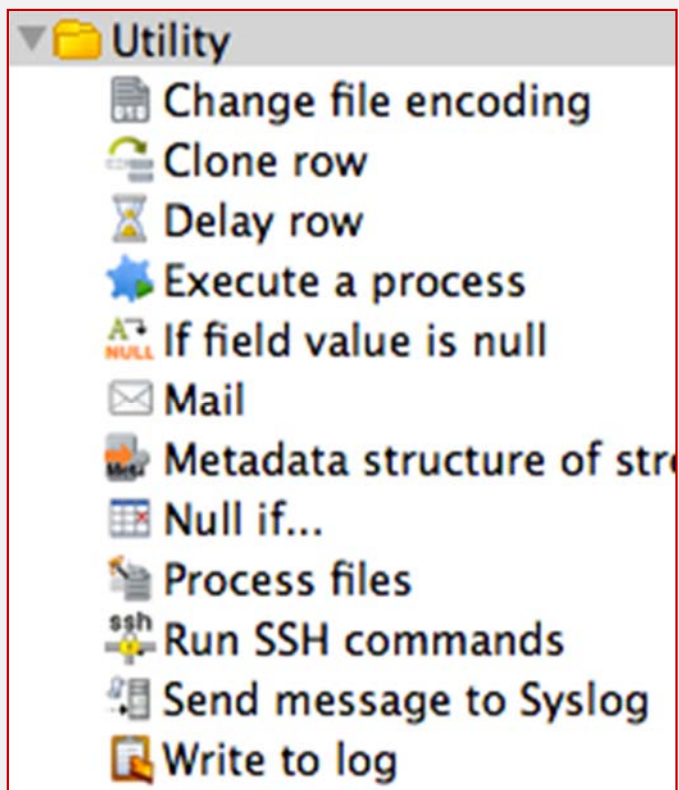
# Kettle: Transformations



## Transform

collection of step dealing with realize data transformation: i.e. trim, fields separation, strings truncation, rows sorting, etc.....

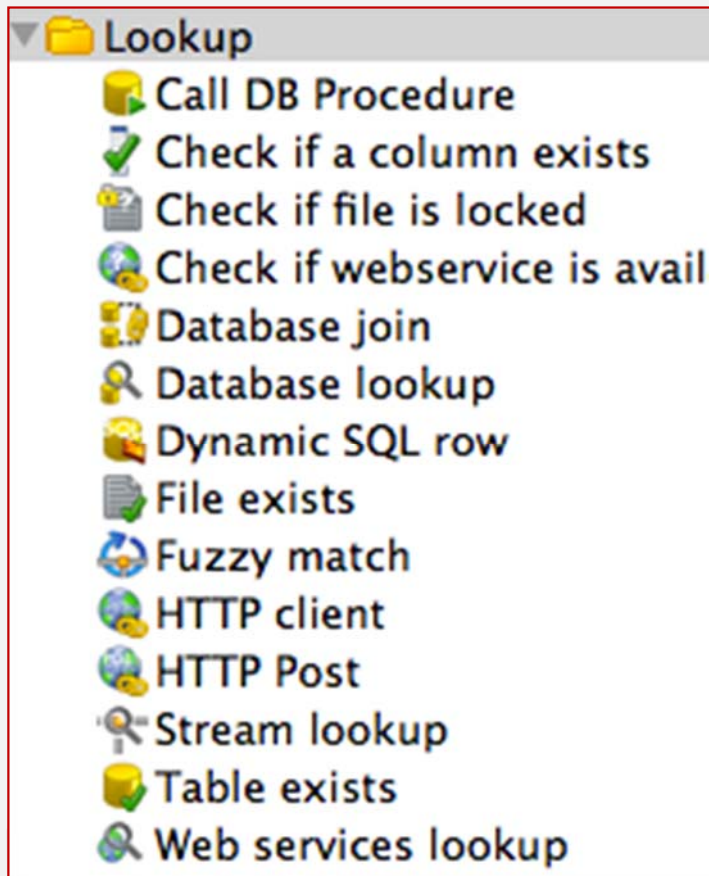
# Kettle: Transformations



## Utility

collection of step with advanced or supporting features: i.e. log writing, check if a field is null, rows deletion, etc....

# Kettle: Transformations

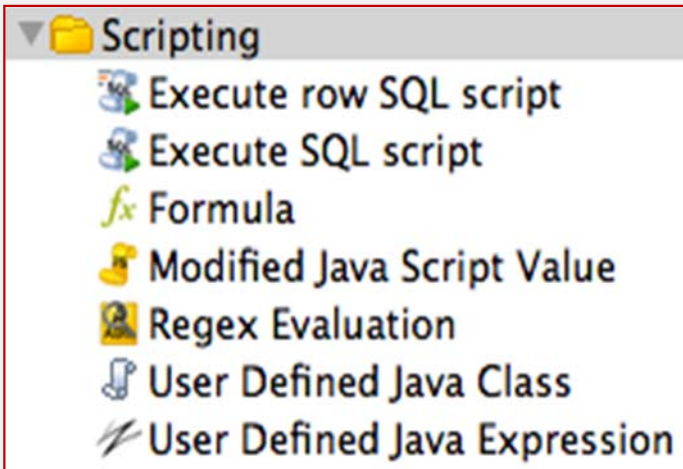


## Lookup

collection of step allowing data consultation operations on specific solutions or on data already extrapolated and kept in temporary structures (to increase speed and reactivity).



# Kettle: Transformations

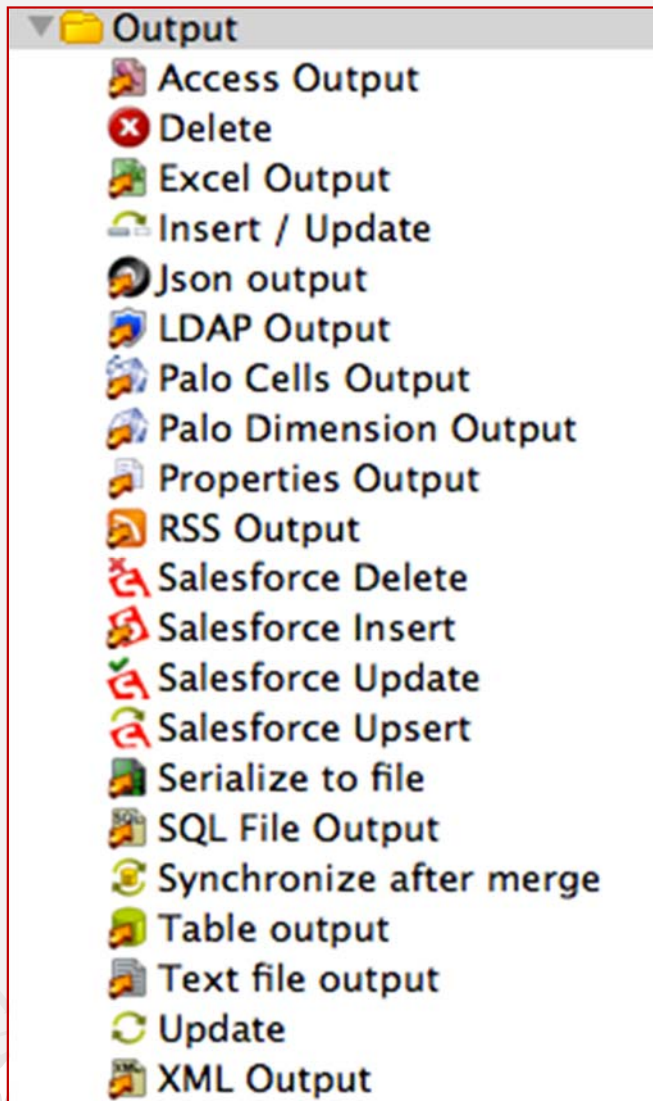


## Scripting

set of step in which you can define scripts in different languages (SQL, JavaScript, etc...).



# Kettle: Transformations



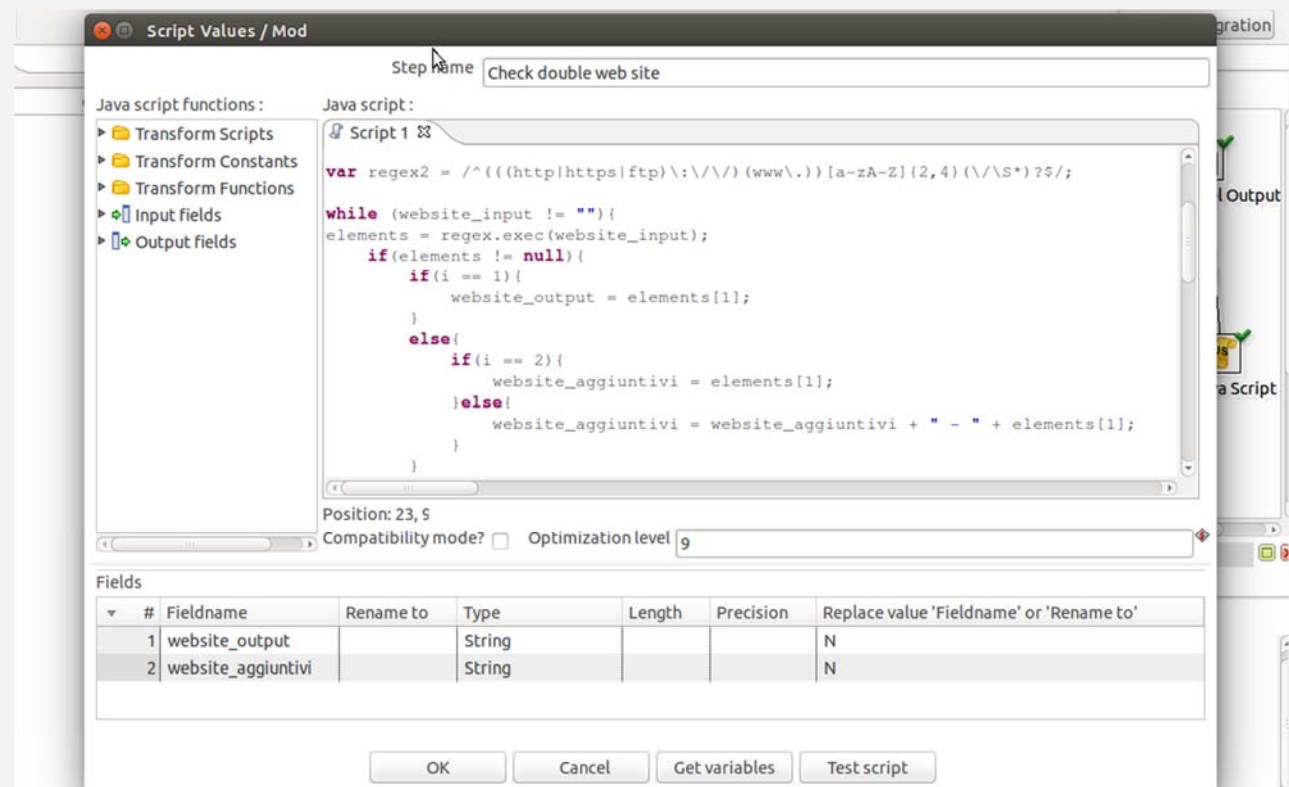
## Output

Collection of step dealing with output data management. They are present in various types.

# Kettle: Transformations

Kettle offers many types of steps to execute various data operations, also it offers:

- *possibility of use and add some JavaScript code.*
- *possibility of use regular expressions.*



The screenshot shows the 'Script Values / Mod' dialog box in Kettle. The 'Step name' is 'Check double web site'. The 'Java script functions' list includes Transform Scripts, Transform Constants, Transform Functions, Input fields, and Output fields. The 'Java script' field contains the following code:

```
var regex2 = /^(http|https|ftp)\:\/\/(www\.)?[a-zA-Z]{2,4}\/\S*?$/;
while (website_input != ""){
elements = regex.exec(website_input);
if(elements != null){
if(i == 1){
website_output = elements[1];
}
else{
if(i == 2){
website_aggiuntivi = elements[1];
}
else{
website_aggiuntivi = website_aggiuntivi + " - " + elements[1];
}
}
}
}
```

The 'Position' is 23, 5. The 'Compatibility mode?' checkbox is unchecked, and the 'Optimization level' is set to 9. The 'Fields' table is as follows:

#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fieldname' or 'Rename to'
1	website_output		String			N
2	website_aggiuntivi		String			N

Buttons at the bottom include OK, Cancel, Get variables, and Test script.

# Kettle: Pan e Kitchen

- The Transformations made with Spoon can be executed with Pan from command line ( similarly Kitchen for the Job).

```
/usr/local/pdi/pan.sh -file /home/pentaho/repos/LetturaDati.ktr
```

```
# Lancia il job ogni sabato alle sei di mattina...  
6 6 * * 6 /usr/local/pdi/kitchen.sh -file /home/pentaho/repo/Aggiornal.kjb >> /tmp/cron1.log 2>&1
```

- The output is typically recorded on a log in order to analyze it in case of problems.

```
INFO 07-06 06:10:02,486 - Using "/tmp/vfs_cache" as temporary files store.  
INFO 07-06 06:10:02,712 - Pan - Start of run.  
INFO 07-06 06:10:02,902 - Lettura dati per DWH - Dispatching started for transformation [Lettura dati per DWH]  
INFO 07-06 06:10:02,929 - Lettura dati per DWH - This transformation can be replayed with replay date: 2011/06/07 06:10:02  
INFO 07-06 06:10:03,233 - DB DWH - Connected to database [Self DB] (commit=100)  
INFO 07-06 06:10:03,599 - DB AS_UTIL - Finished reading query, closing connection.  
INFO 07-06 06:10:03,614 - DB AS_UTIL - Finished processing (I=27, O=0, R=0, W=27, U=0, E=0)  
INFO 07-06 06:10:03,625 - DB DWH - Finished processing (I=0, O=27, R=27, W=27, U=0, E=0)  
INFO 07-06 06:10:03,626 - Pan - Finished!  
INFO 07-06 06:10:03,627 - Pan - Start=2011/06/07 06:10:02.713, Stop=2011/06/07 06:10:03.126  
INFO 07-06 06:10:03,627 - Pan - Processing ended after 0 seconds.  
INFO 07-06 06:10:03,627 - Lettura dati per DWH -  
INFO 07-06 06:10:03,627 - Lettura dati per DWH - Step DB AS_UTIL.0 ended successfully, processed 27 lines. ( - lines/s)  
INFO 07-06 06:10:03,628 - Lettura dati per DWH - Step DB DWH.0 ended successfully, processed 27 lines. ( - lines/s)
```

# Sequential Execution



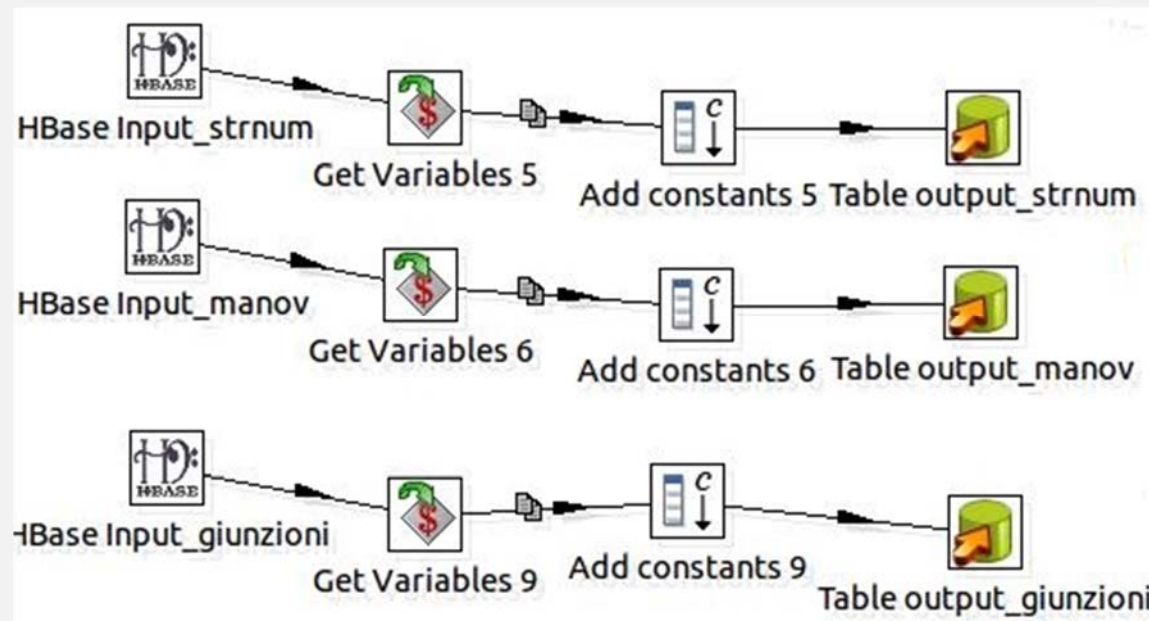
These steps (transformations) are executed sequentially (there is a single flow execution).

```

void main(){
    int a;
    f1(a);
    f2(a+2);
}
  
```

The statements are executed sequentially.

# Parallel Execution



- Unlike before there are multiple streams of execution that are executed in parallel (simultaneously).
- Like in a multi-threading programming, multiple thread (portions of the running program) can virtually run independently and in parallel.

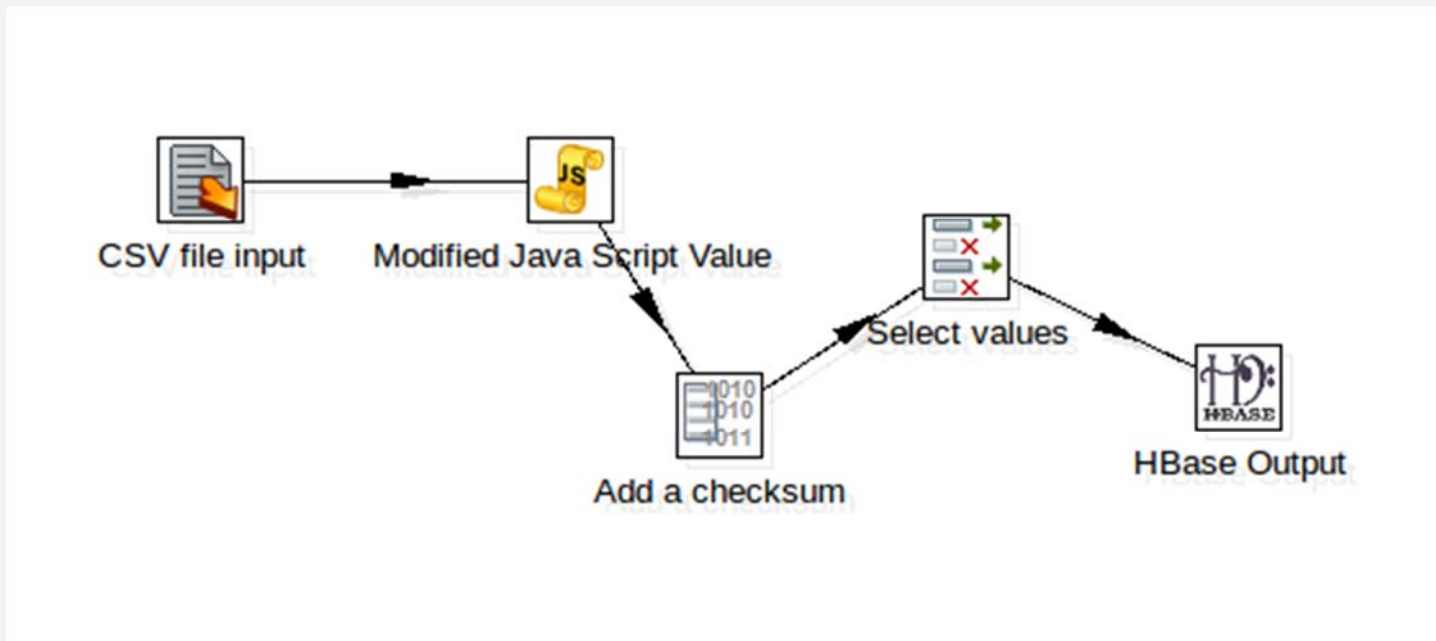
3

# ETL tool: Pentaho Data Integration (PDI)

## EXAMPLES

# Transformation Hbase Output

- This transformation takes the museums file in CSV format and defines a key to load data into the HBase table “monuments”.



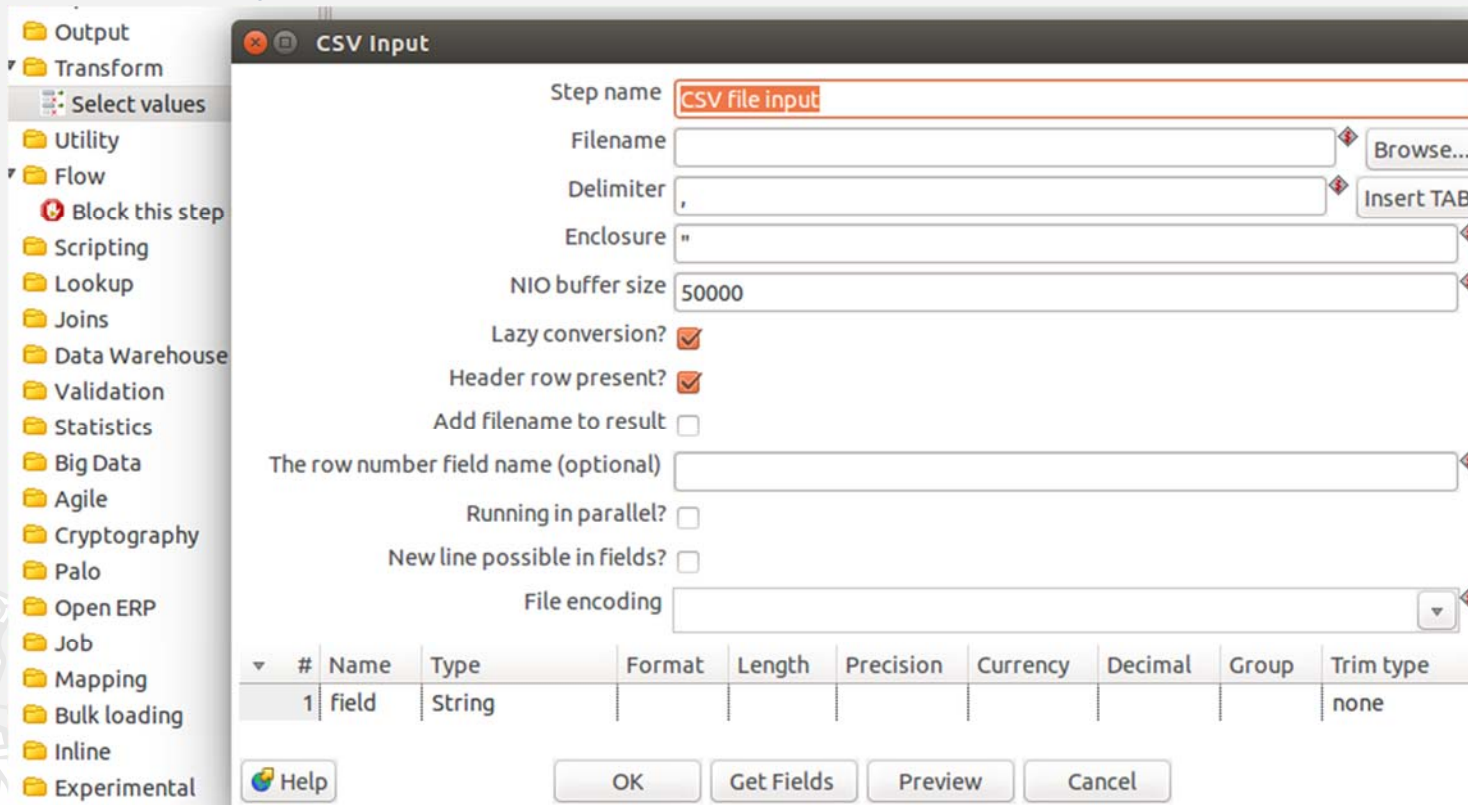
- It is composed of five steps.
- The sequence is given by data flow.



# Transformation Hbase Output

## CSV file input

- In this step you select the CSV file, you can choose the separator type used and select the fields to be imported using the Get field button (also you can determine the type and other parameters).



Step name:

Filename:

Delimiter:

Enclosure:

NIO buffer size:

Lazy conversion?

Header row present?

Add filename to result

The row number field name (optional)

Running in parallel?

New line possible in fields?

File encoding:

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	field	String							none

# Transformation Hbase Output

## Modified Java Script value

- In this step you can add JavaScript code. It is defined a variable by concatenating two input fields and at the end the same variable is used to define an output field.

Step name: Modified Java Script Value

Java script functions:

- Transform Scripts
- Transform Constants
- Transform Functions
- Input fields
- Output fields

Java script:

```
//Script here
var key = id+denominazione;
```

Position: 3, 27

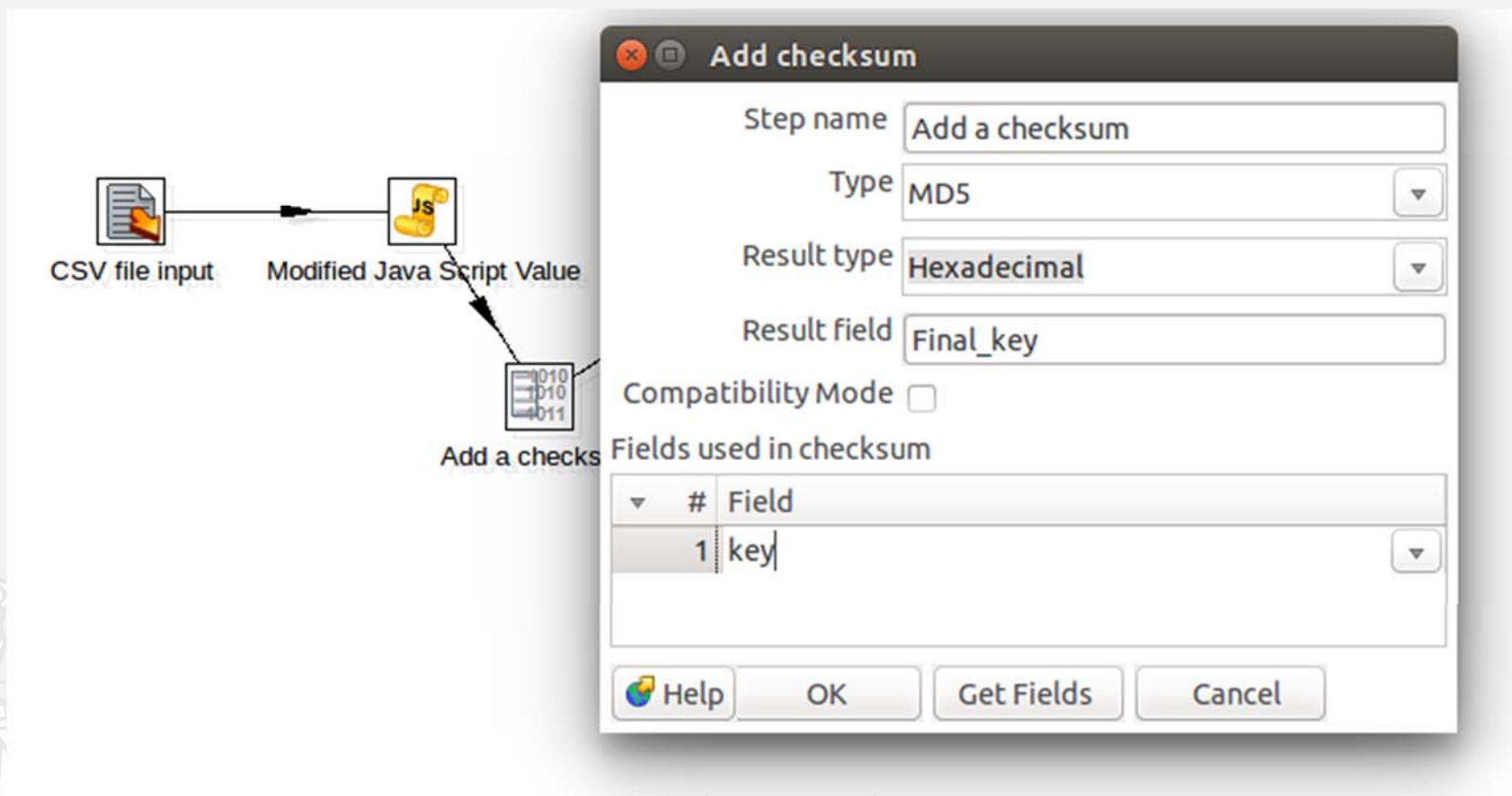
Compatibility mode?  Optimization level: 9

#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fieldname' or 'Rename to'
1	key		String			

# Transformation Hbase Output

## Add a Checksum

- This step allows you to choose which algorithm (MD5, CRC32) to use to encode a field (usually the key) and define the new name in output.



The diagram illustrates a data transformation process. It starts with a 'CSV file input' icon, which leads to a 'Modified Java Script Value' icon. From there, an arrow points to an 'Add a checksum' icon. A dialog box titled 'Add checksum' is overlaid on the 'Add a checksum' icon, showing the configuration for this step.

**Add checksum dialog configuration:**

- Step name: Add a checksum
- Type: MD5
- Result type: Hexadecimal
- Result field: Final\_key
- Compatibility Mode:
- Fields used in checksum:
 

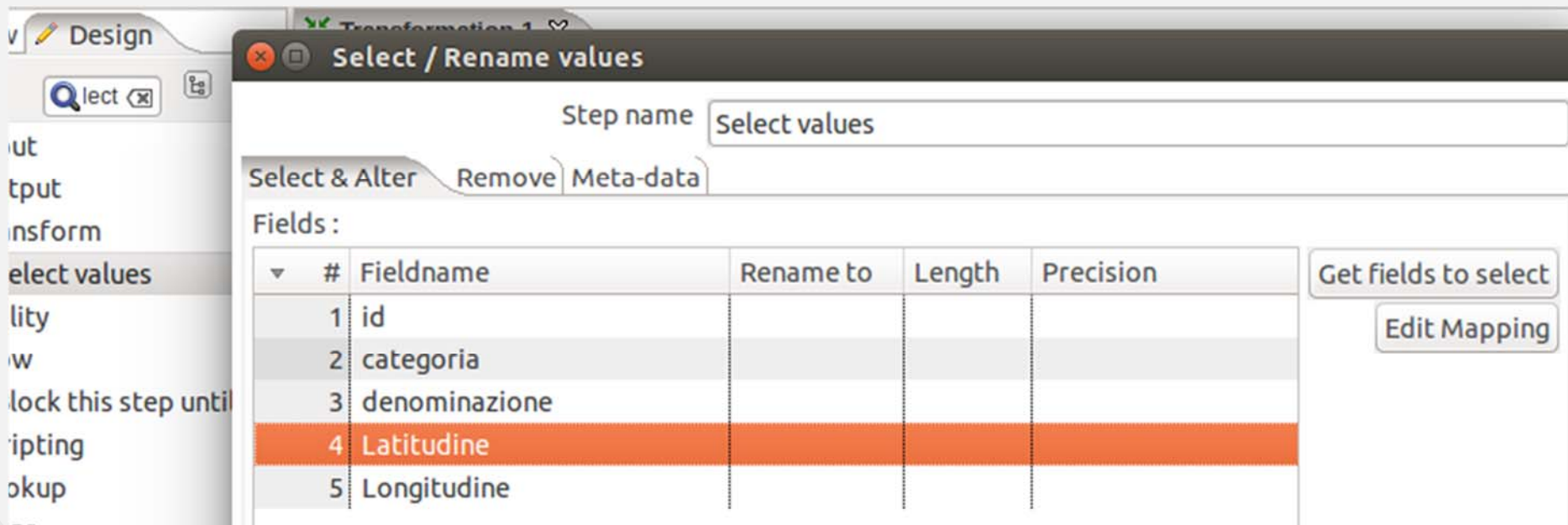
#	Field
1	key

Buttons at the bottom of the dialog: Help, OK, Get Fields, Cancel.

# Transformation Hbase Output

## Select Values

- In this step you can select the fields (one or more) that you want to pass to next step. Furthermore, you can also use the Remove option to select the fields you want to block.



Design

Select / Rename values

Step name: Select values

Select & Alter Remove Meta-data

Fields:

#	Fieldname	Rename to	Length	Precision
1	id			
2	categoria			
3	denominazione			
4	Latitudine			
5	Longitudine			

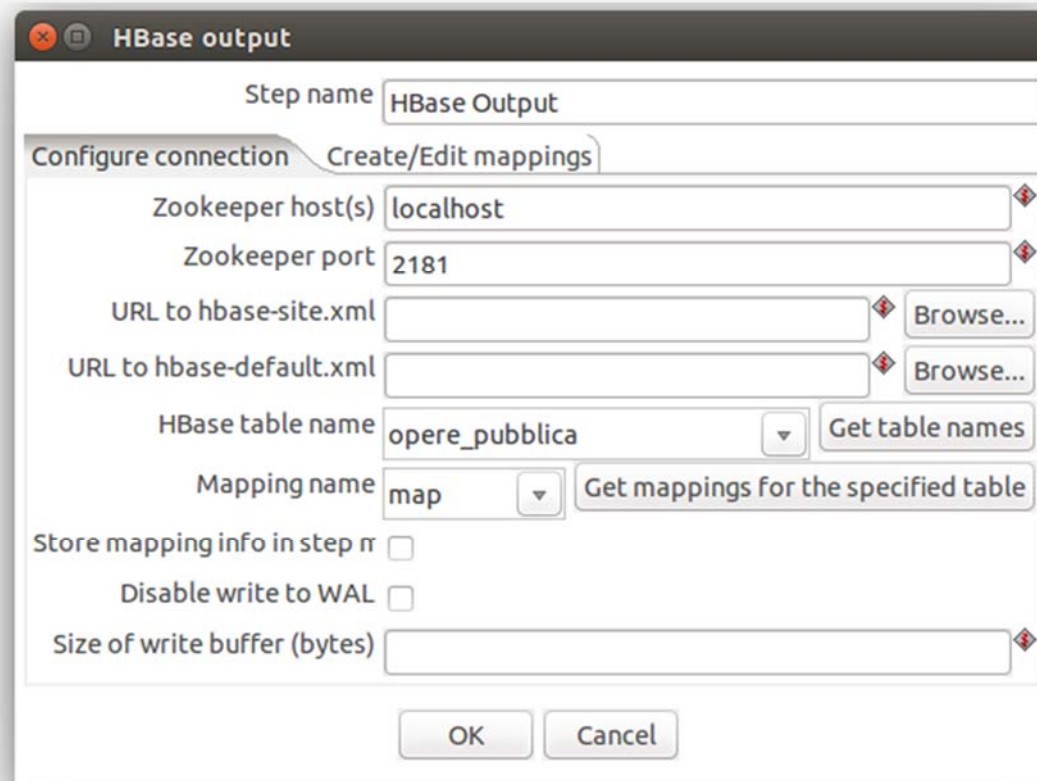
Get fields to select

Edit Mapping

# Transformation Hbase Output

## Hbase Output

- In this step you set the parameters to load data into a table HBase.
- In the first tab you define the port (2181) and the IP address of the machine that hosts the database.



The screenshot shows a dialog box titled "HBase output" with two tabs: "Configure connection" (selected) and "Create/Edit mappings". The "Configure connection" tab contains the following fields and controls:

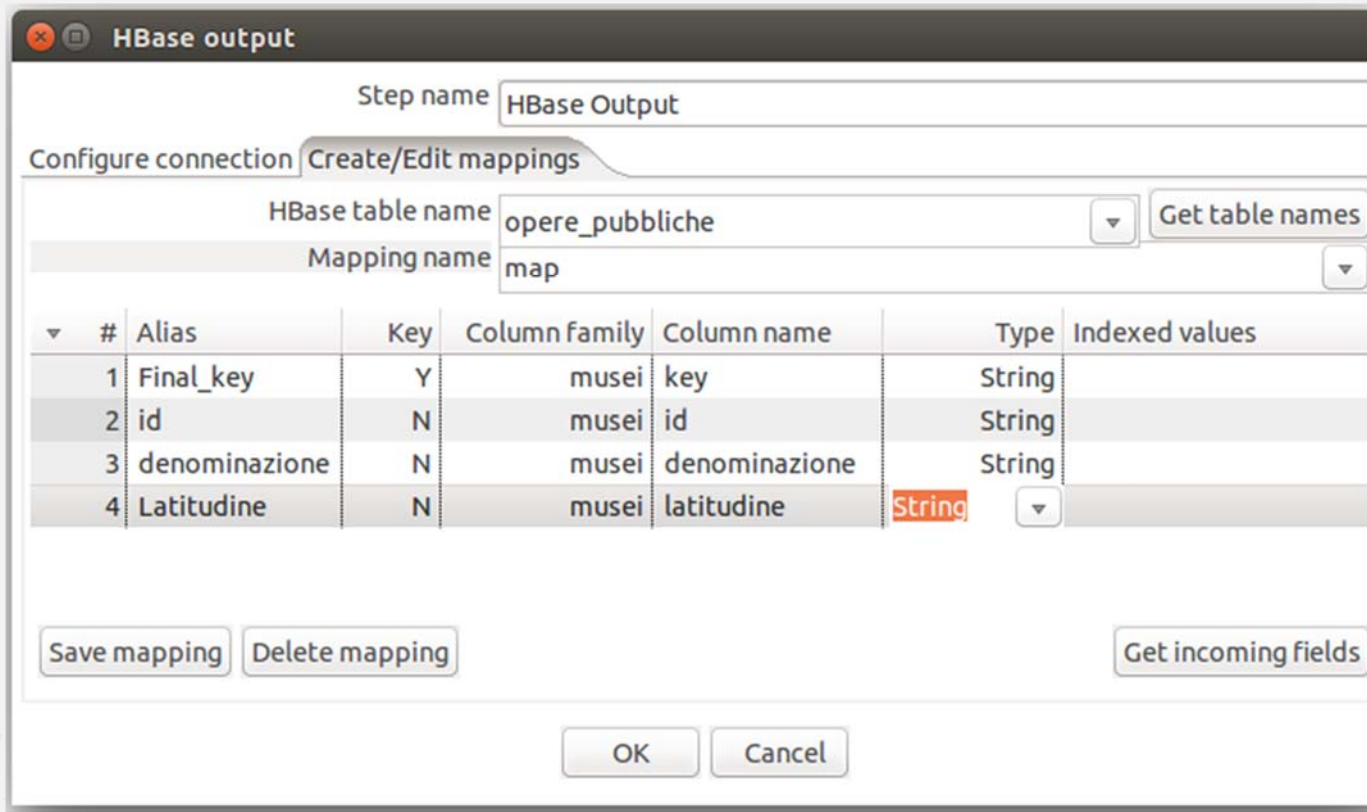
- Step name: HBase Output
- Zookeeper host(s): localhost
- Zookeeper port: 2181
- URL to hbase-site.xml: [empty] Browse...
- URL to hbase-default.xml: [empty] Browse...
- HBase table name: opere\_pubblica Get table names
- Mapping name: map Get mappings for the specified table
- Store mapping info in step:
- Disable write to WAL:
- Size of write buffer (bytes): [empty]

At the bottom of the dialog are "OK" and "Cancel" buttons.

# Transformation Hbase Output

## Hbase Output

- In the second tab you select the table, the mapping and the fields to be loaded.



Step name: HBase Output

Configure connection: Create/Edit mappings

HBase table name: opere\_pubbliche (Get table names)

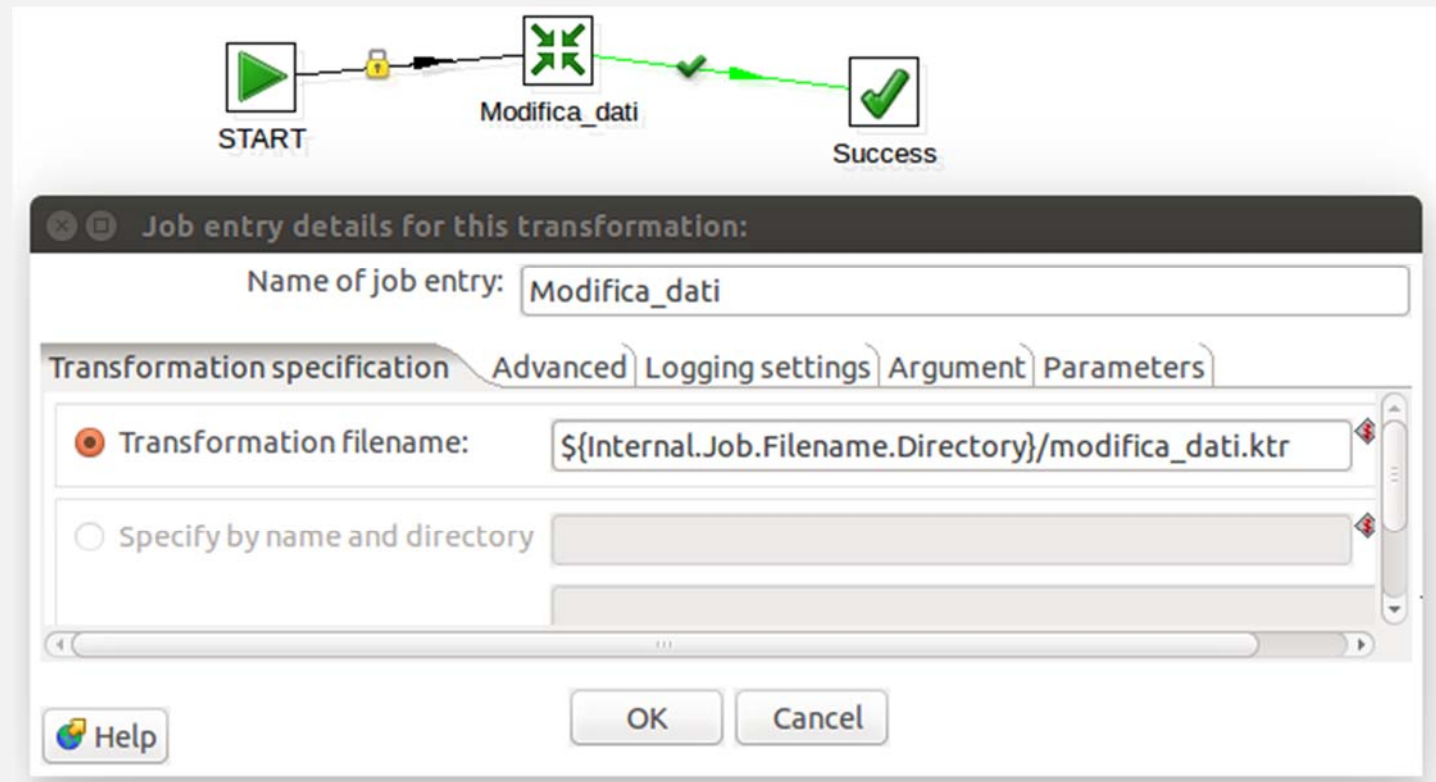
Mapping name: map

#	Alias	Key	Column family	Column name	Type	Indexed values
1	Final_key	Y	musei	key	String	
2	id	N	musei	id	String	
3	denominazione	N	musei	denominazione	String	
4	Latitudine	N	musei	latitudine	String	

Buttons: Save mapping, Delete mapping, Get incoming fields, OK, Cancel

# Job

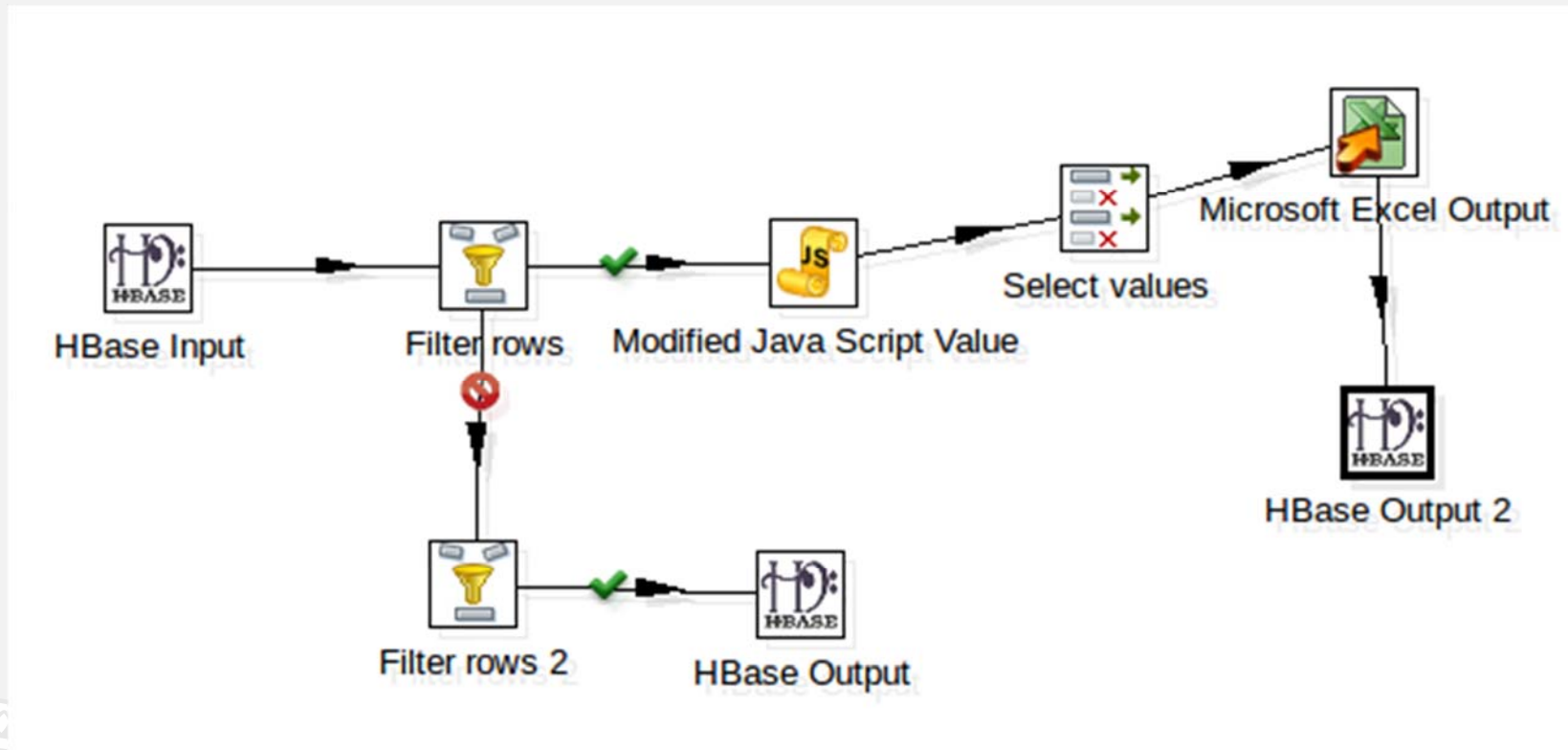
- Create a Job that contains the start step, the “Modifica\_dati” step (selected transformation) and the final step (Success).



- The Hop between the different steps of a Job represent the control flow and define the sequence of steps to perform.

# Transformation Modifica\_dati

- This Transform loads data from HBase through the HBase Input Step. Then, it applies the Filter rows and Modified Java Script Value Step to perform some data cleaning before storing again on HBase.



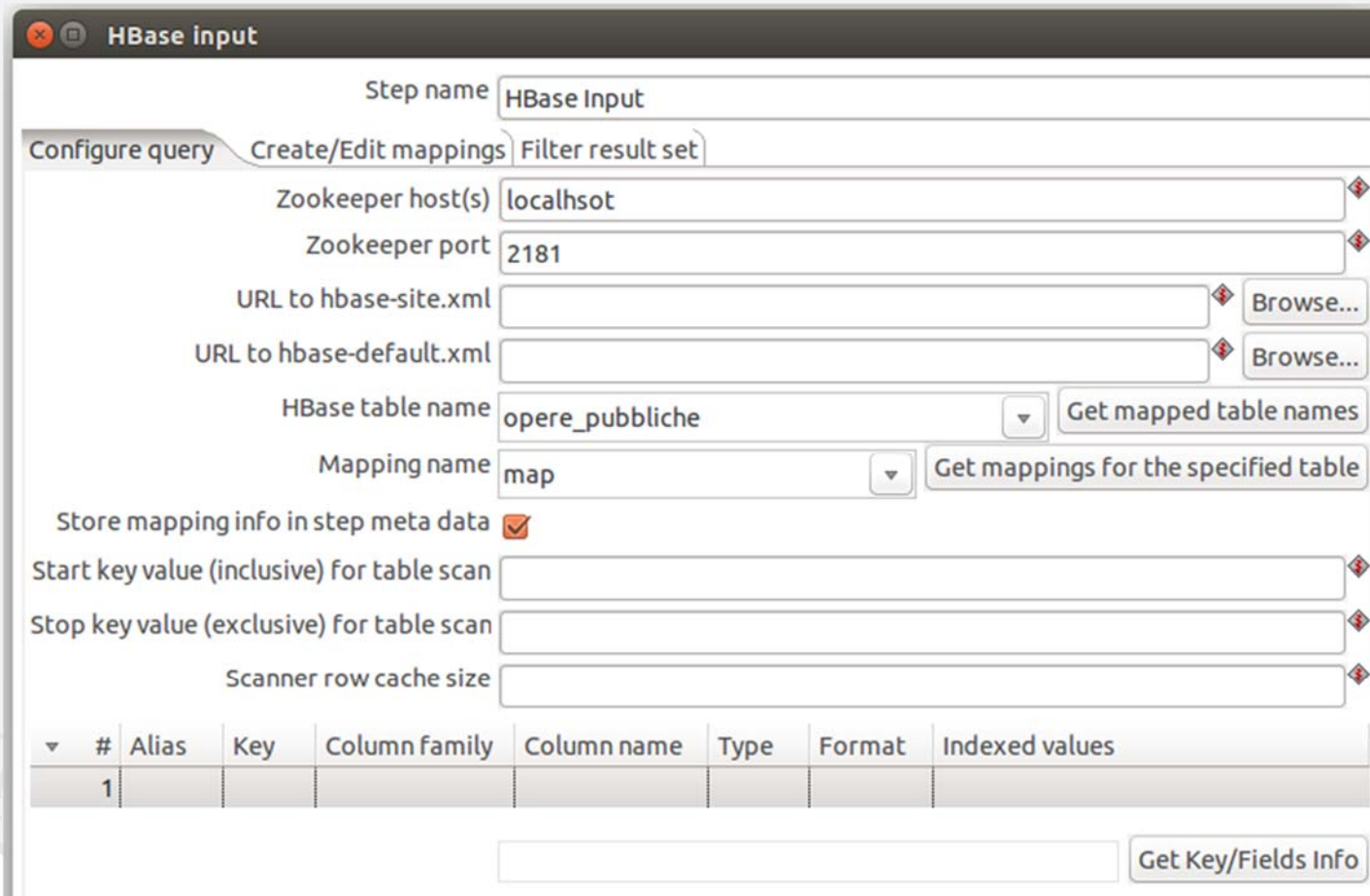
- In this case the data flow is splitted into two different smaller data streams based on specific condition.



# Transformation Modifica\_dati

## Hbase Input

- In this step you set the parameters to retrieve the data from Hbase.
- You can perform a filtering by setting some conditions in the tab Filter result set.



The screenshot shows the 'HBase input' configuration window with the following fields and options:

- Step name: HBase Input
- Configure query | Create/Edit mappings | Filter result set (selected)
- Zookeeper host(s): localhsot
- Zookeeper port: 2181
- URL to hbase-site.xml: [empty] Browse...
- URL to hbase-default.xml: [empty] Browse...
- HBase table name: opere\_pubbliche Get mapped table names
- Mapping name: map Get mappings for the specified table
- Store mapping info in step meta data:
- Start key value (inclusive) for table scan: [empty]
- Stop key value (exclusive) for table scan: [empty]
- Scanner row cache size: [empty]

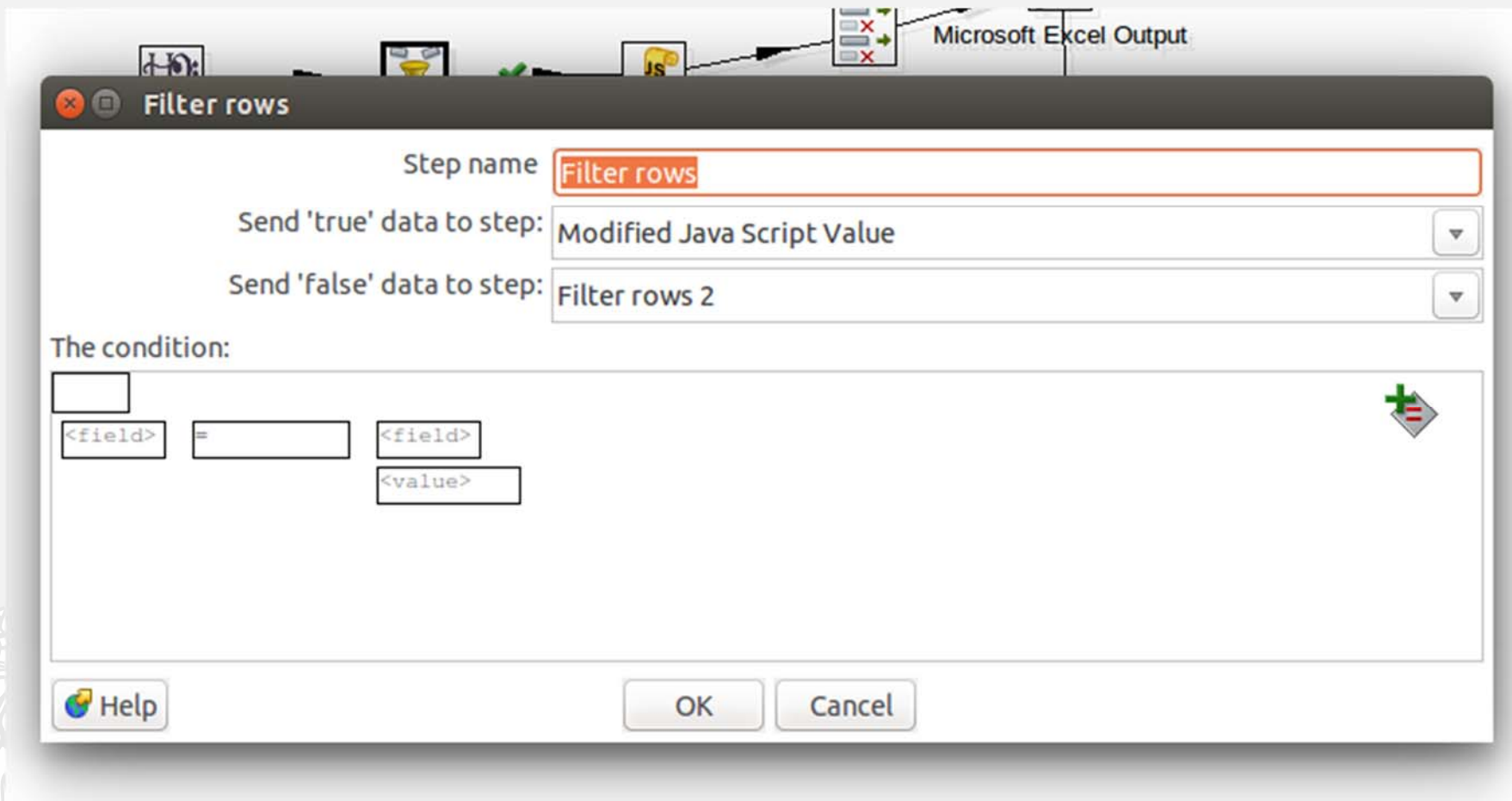
#	Alias	Key	Column family	Column name	Type	Format	Indexed values
1							

Get Key/Fields Info

# Transformation Modifica\_dati

## Filter rows

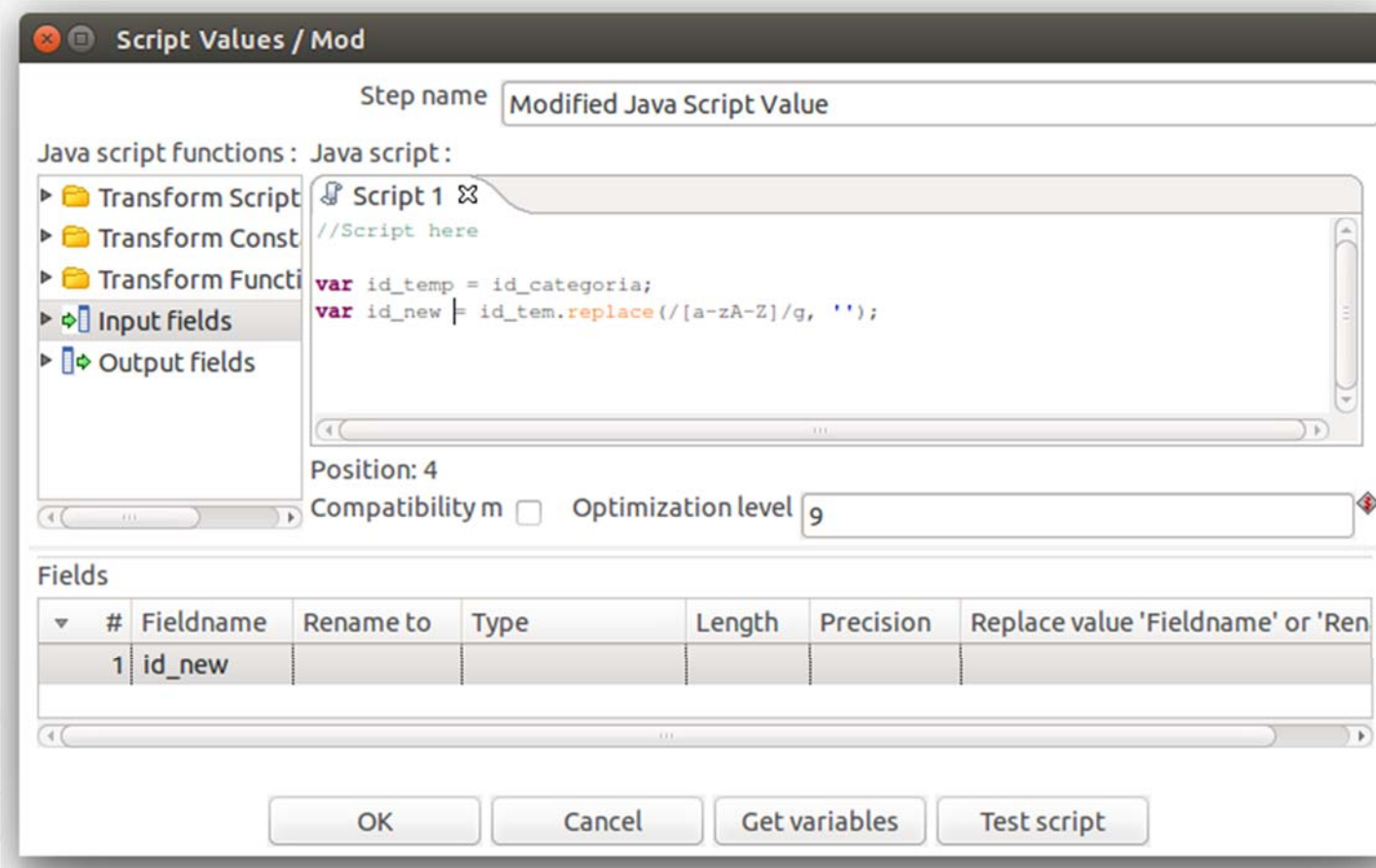
- It's another way to filter the data flow specifying a condition. In output this step creates a fork of the data flow.



# Transformation Modifica\_dati

## Modified Java Script Value

- In this step you will use a regular expression inside the Javascript code. The goal is to replace all literals characters within a given field, leaving only numeric ones.



Script Values / Mod

Step name: Modified Java Script Value

Java script functions: Java script:

- Transform Script
- Transform Const
- Transform Functi
- Input fields**
- Output fields

Script 1

```
//Script here
var id_temp = id_categoria;
var id_new = id_tem.replace(/[a-zA-Z]/g, '');
```

Position: 4

Compatibility  Optimization level 9

Fields

#	Fieldname	Rename to	Type	Length	Precision	Replace value 'Fieldname' or 'Ren
1	id_new					

OK Cancel Get variables Test script

# 4

## Developing ETL processes with PDI



# 4

## Tool installation & configuration

Developing  
ETL  
processes  
with PDI



# ON LINUX UBUNTU 14.04

# Step 1: Downloading PDI

- Prerequisite: Oracle Java 7 JDK already installed on your system.
  - <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (link Oracle)
  - <https://help.ubuntu.com/community/Java> (link Ubuntu)
- Pentaho Data Integration (PDI) will run on Mac, Linux and Windows.



# Step 1: Downloading PDI

- You can obtain Pentaho Data Integration in this way:
  1. Go to Sourceforge page of Pentaho Data Integration project:  
<http://sourceforge.net/projects/pentaho/files/Data%20Integration/> .
  2. Choose the version 5.0.1 stable and download the zip / tar.gz file.



## Step 2: Installation PDI

- PDI only requires you to unpack the downloaded zip / tar.gz file into a specific folder.
  - On Linux operating systems you will need to make the shell scripts executable by using the command `chmod`:

```
cd Kettle  
chmod +x *.sh
```





## Step 3: Running PDI

- After installation, your PDI folder contains the graphical user interface called Spoon, command line utilities to execute transformations, jobs and other tools.
- Spoon is the graphical tool with which you design and test every PDI process. The other PDI components (Pan and Kitchen) execute the processes designed with Spoon, and are executed from a terminal window.

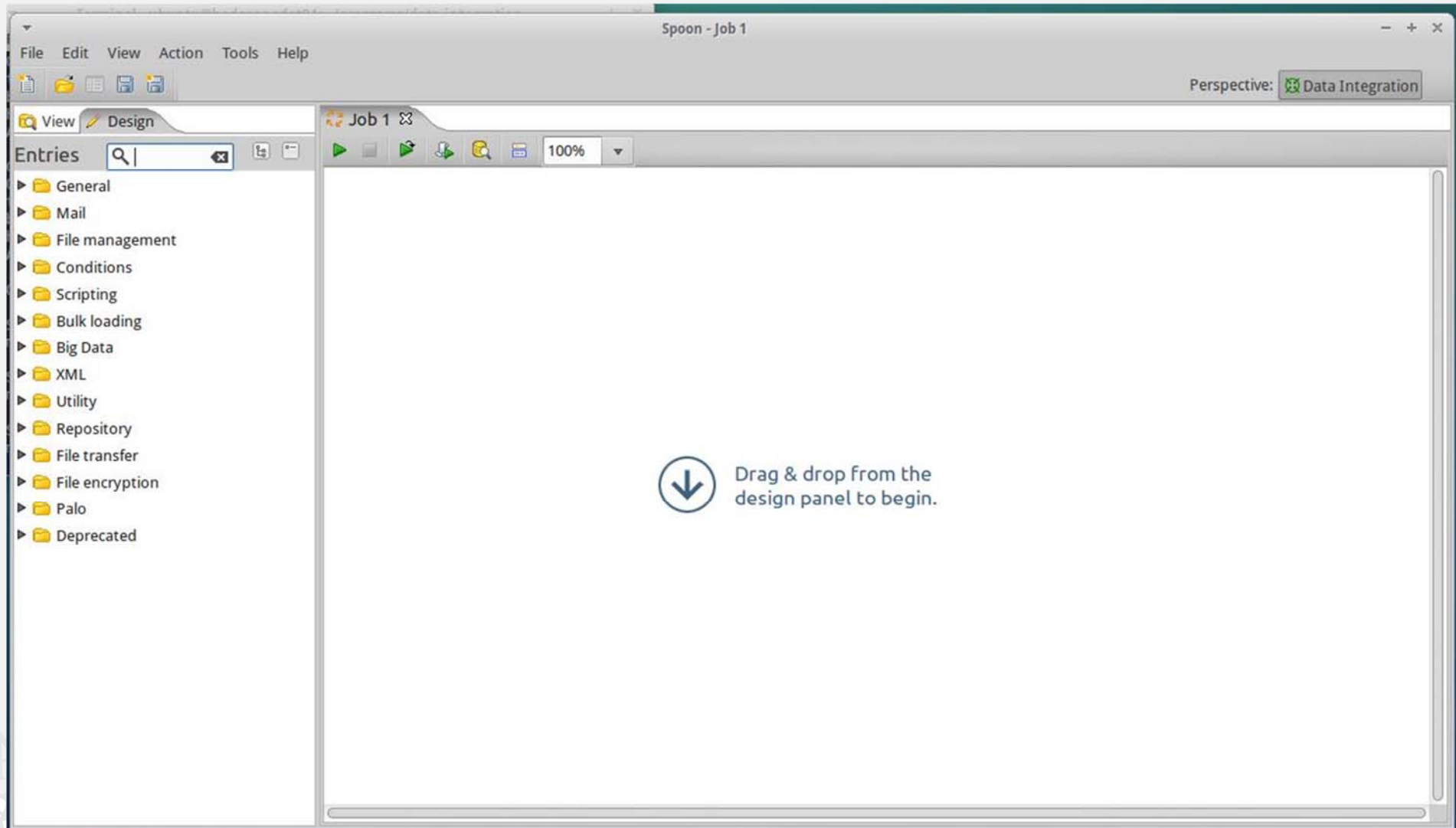


## Step 3: Running PDI

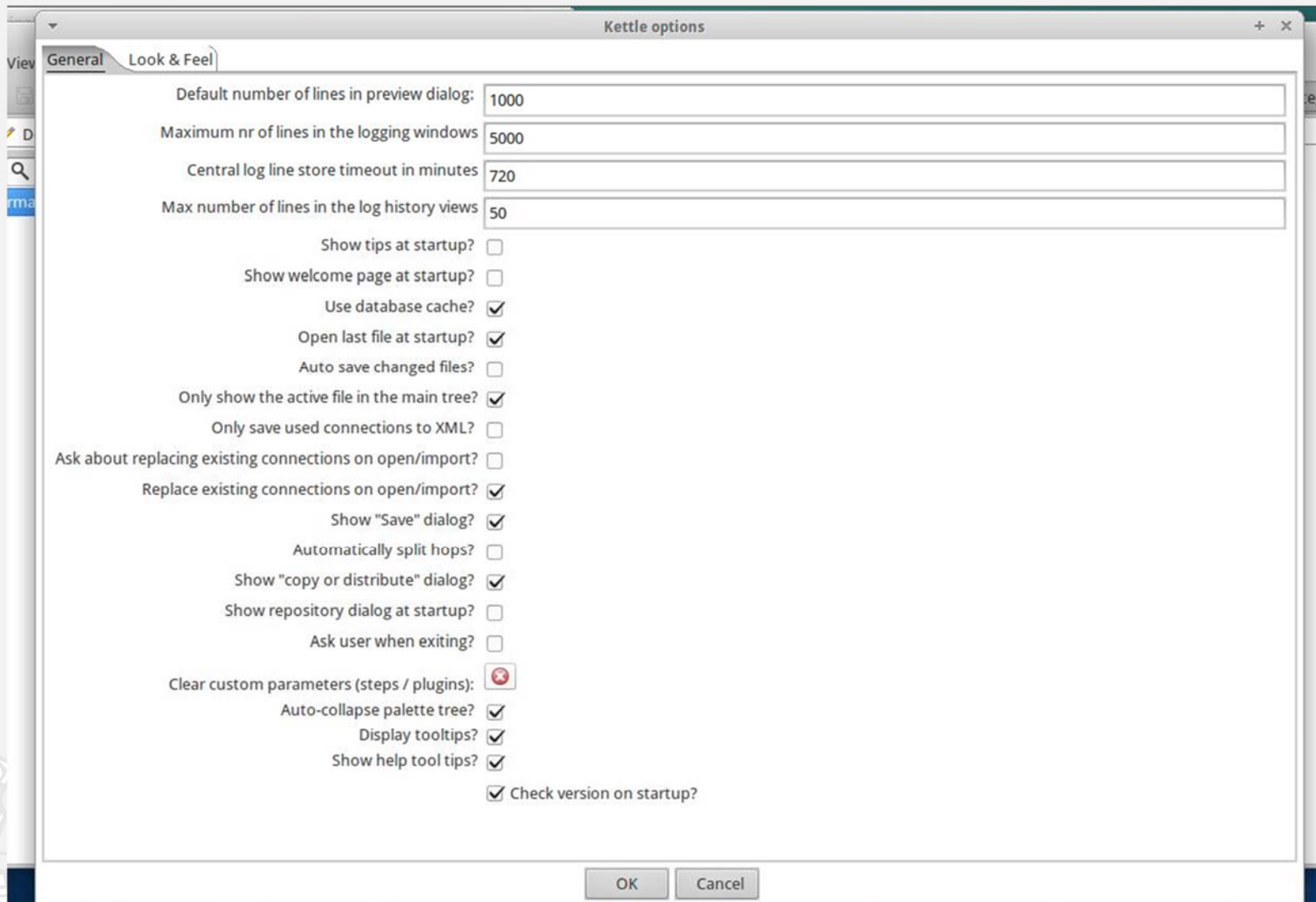
FILE/FOLDER	DESCRIPTION
<code>\PDI folder\data-integration</code>	Contains the Spoon designer and command line utilities
<code>\PDI folder\data-integration\spoon.sh</code>	Script file for starting the Spoon Designer on Linux and Macintosh
<code>\PDI folder\data-integration\Spoon.bat</code>	Script file for starting the Spoon Designer on Windows

To start Spoon you execute  
Spoon.bat on Windows or  
spoon.sh on Linux/Mac

# Step 3: Running PDI



# Step 4: Customizing Spoon



# Step 5: Configure Database

- Load / Write data from / to disparate data source: relational database (MySQL), not relational database (Hbase, Cassandra, MongoDB, CouchDB...), etc... .
- The database interaction is managed through specific database connector provided by different vendors.



# Step 5: Configure MySQL

- MySQL is one of the most widely used database in the world and is a relational database management system (RDBMS).
- SQL (Structured Query Language) is a standard language for relational database management and is used to query, insert, update and modify data... .
- MySQL Workbench is the free official integrated tool that enables users to graphically administer MySQL databases and visually to design database structures.

# Step 5: Configure MySQL

- You can use XAMPP to install MySQL DBMS in your machine.
  - XAMPP package contains also Apache web server and PHP as scripting language
  - There are currently distributions for Windows, Linux, and OS X.
  - Download from:
    - <https://www.apachefriends.org/it/index.html>
    - <http://wiki.ubuntu-it.org/Server/Xampp> (Ubuntu Linux)
  - Start XAMPP with the command **sudo /opt/lampp/lampp start**.

# Step 5: Configure MySQL

- To connect PDI to MySQL, the JDBC connector must be copied in Pentaho data integration folder.
- JDBC connector can be obtained from MySQL vendor:  
<http://dev.mysql.com/downloads/connector/j/3.1.html>
- Unzip the downloaded file and copy the file **mysql-connector-java-5.1.31-bin.jar** in .../penthao data integration folder/lib .



# Step 6: Configure HBase

- NoSQL Database.
- Column-oriented datastore.
- It is designed for random, real time read/write access to your Big Data.
- Run modes:
  - Standalone
  - Distributed (require an instance of Hadoop Distributed File System (HDFS)).

# Step 6: Configure HBase

- Prerequisite: Oracle Java JDK already installed (minimum version 1.6).
- Download HBase ver. **0.90.5** from this site (select the file that ends in *.tar.gz*):  
<http://archive.eu.apache.org/dist/hbase/hbase-0.90.5/>
- HBase only requires you decompress and unzip/untar your downloaded file.

## Step 6: Configure HBase

- To set HBase in Standalone mode you write your site-specific configurations in a certain file: simply edit the file *conf/hbase-site.xml*.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///DIRECTORY/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/DIRECTORY/zookeeper</value>
  </property>
</configuration>
```

- Set these properties:
  - the directory where HBase writes data (*hbase.rootdir*);
  - the directory where ZooKeeper writes its data (*hbase.zookeeper.property.dataDir*).

## Step 6: Configure HBase

- Finally, start HBase with this command:

```
./bin/start-hbase.sh
```

A P A C H E  
**HBASE**



4

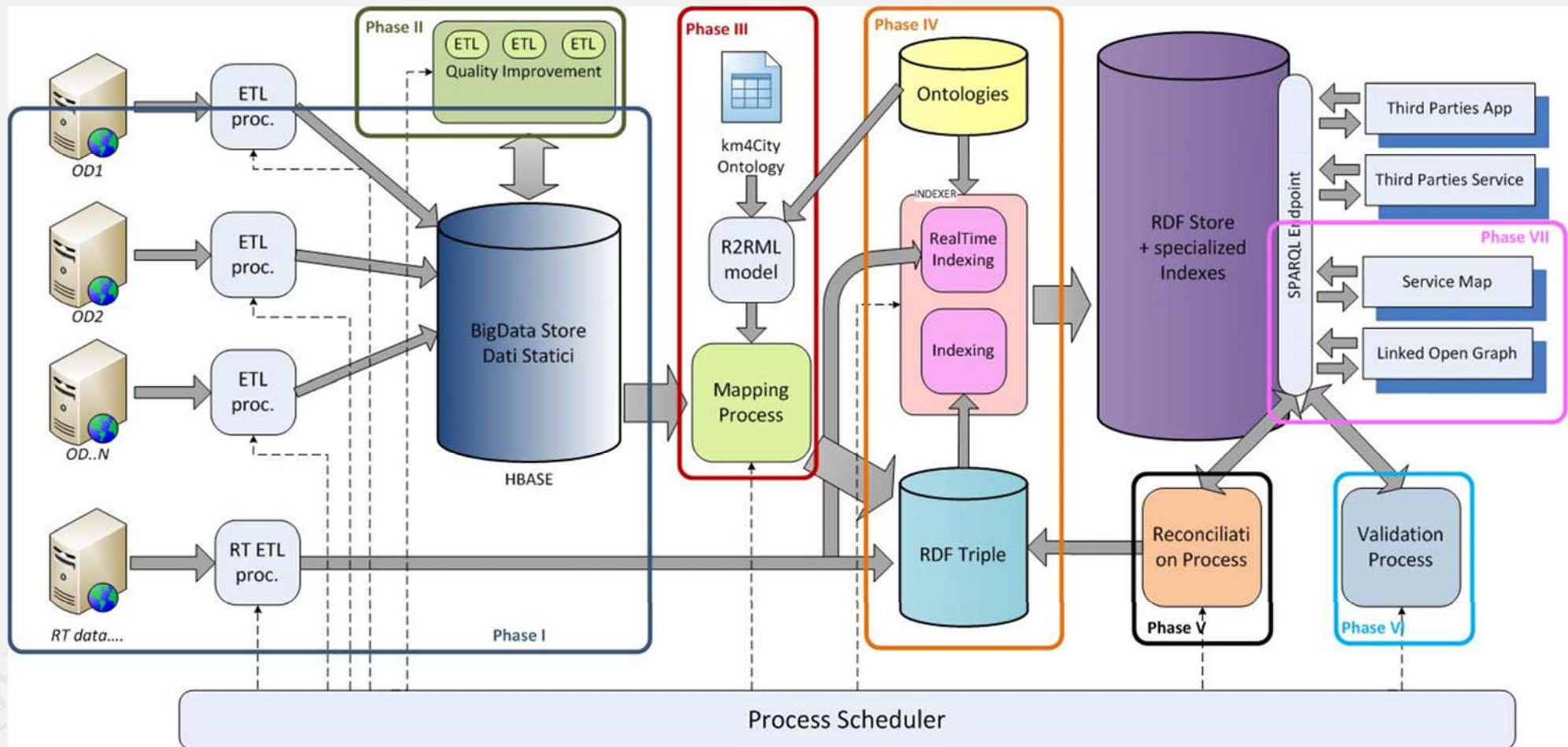
# SiiMobility Project

Developing  
ETL  
processes  
with PDI



# Sii-Mobility project

<http://www.sii-mobility.org>

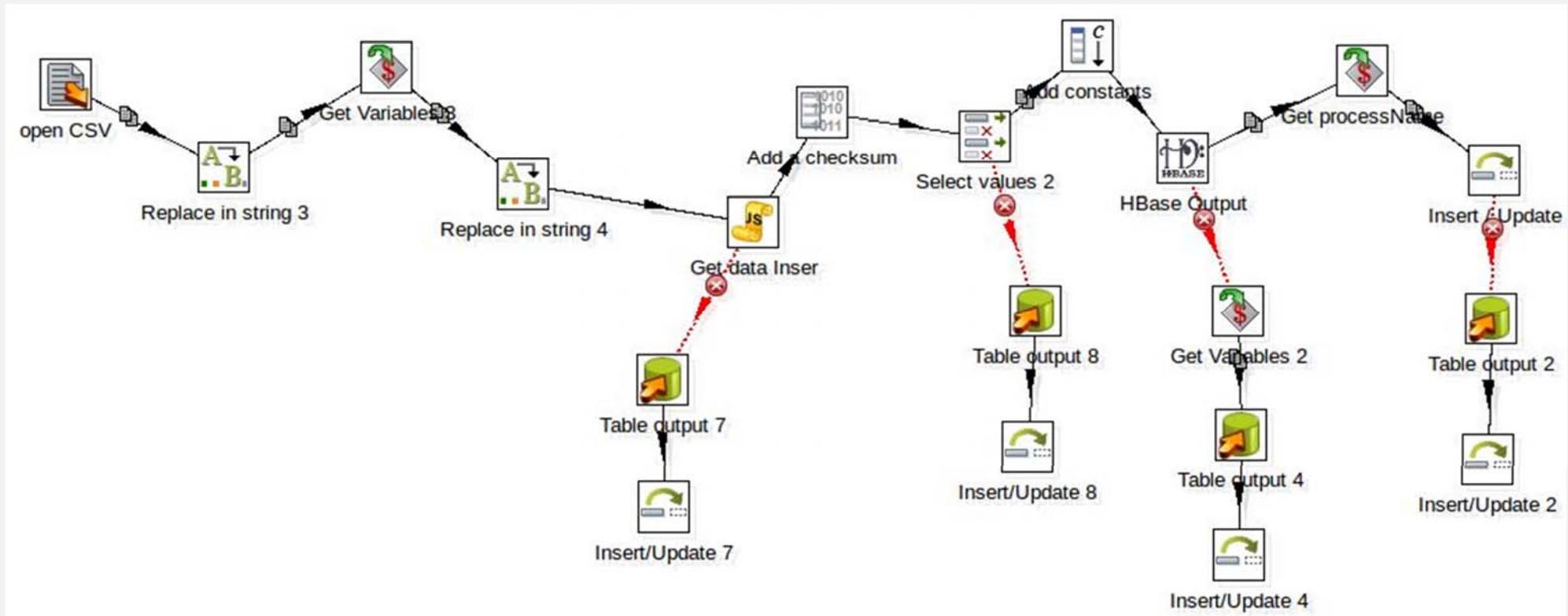


# Transformation Service Data

- To process the service data for Sii-Mobility project.
  - static data from Tuscan region.
- 3 phases:
  - **INGESTION** phase;
  - **QUALITY IMPROVEMENT (QI)** phase;
  - **TRIPLES GENERATION** phase.

# Ingestion phase

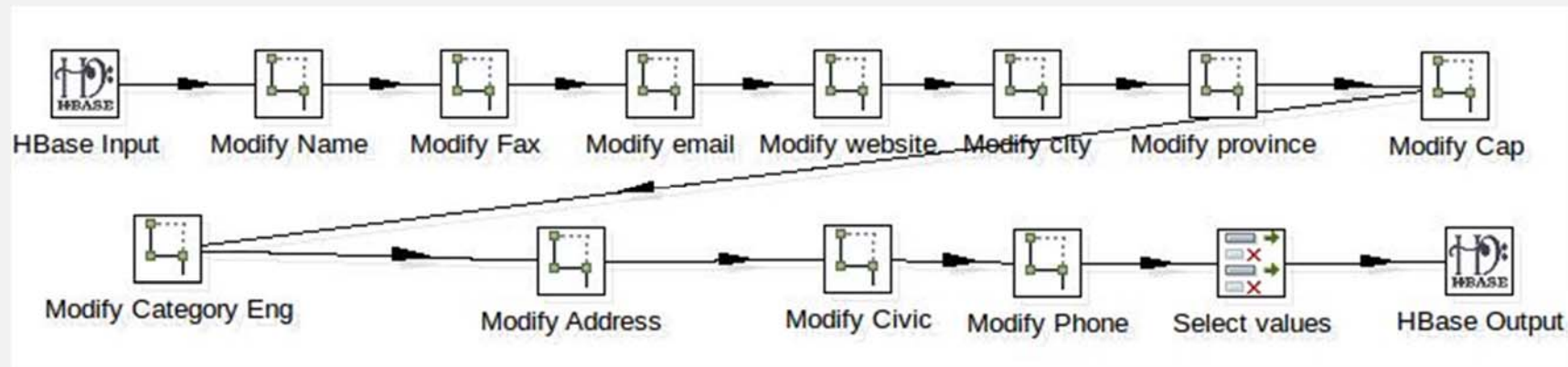
To importing and storage data (in a database) for later use.





# QI phase

To enhance the quality of raw data and to produce reliable and useful information for next applications.



In this case every Step is a transformation.

What is the Data quality ?

# QI phase

Data quality's aspect:

- **Completeness:** presence of all information needed to describe an object, entity or event (e.g. Identifying).
- **Consistency:** data must not be contradictory. For example, the total balance and movements.
- **Accuracy:** data must be correct, i.e. conform to actual values. For example, an email address must not only be well-formed [nome@dominio.it](#), but it must also be valid and working.

# QI phase

- **Absence of duplication:** tables, records, fields should be stored only once, avoiding the presence of copies. Duplicate information involve double handling and can lead to problems of synchronization (consistency).
- **Integrity** is a concept related to relational databases, where there are tools to implement integrity constraints. Example a control on the types of data (contained in a column), or on combinations of identifiers (to prevent the presence of two equal rows).

# QI phase

## Mapping

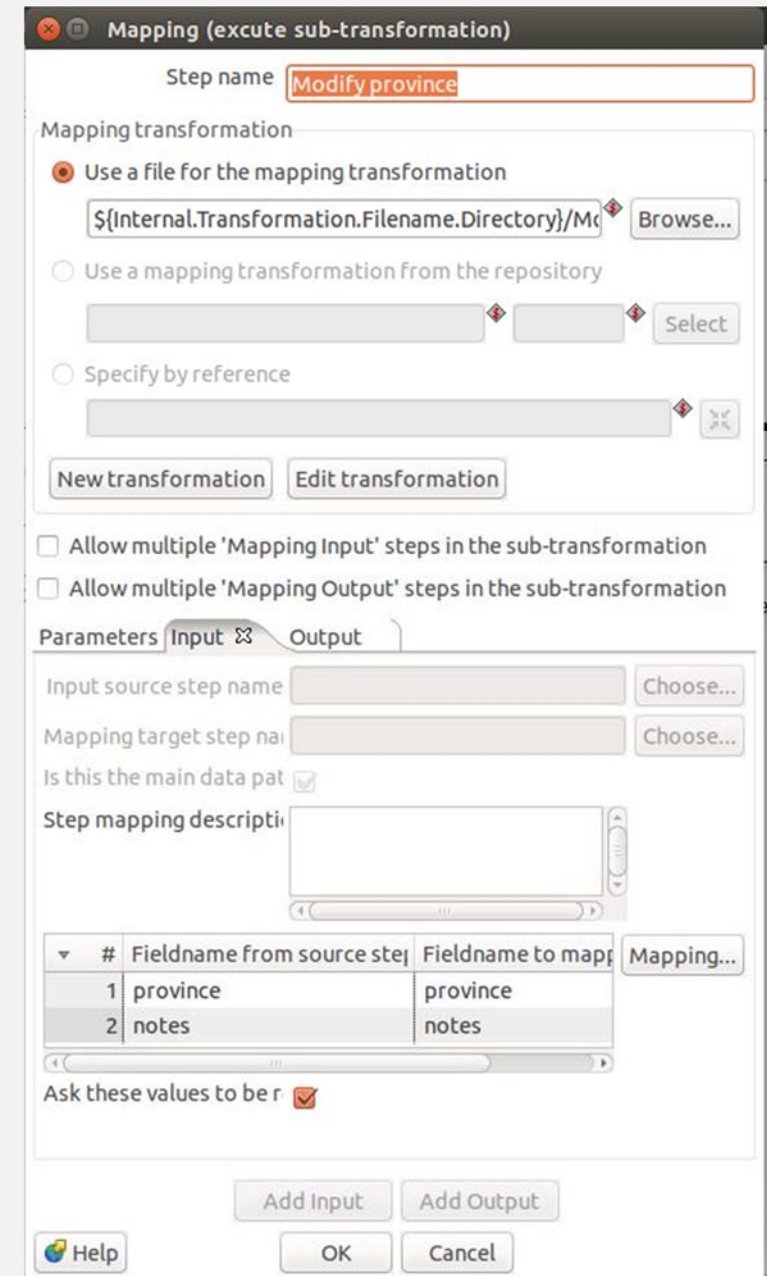
- A mapping is the Kettle solution for transformation re-use.
- For example, if you have a complex calculation that you want to re-use everywhere you can use a mapping.
- These interface steps define the fields structure of the incoming and returning rows. So when a parent transformation calls a sub-transformation the parent row fields are mapped to the fields that the sub-transformation accepts as input. A similar mapping happens when the processed rows are returned to the parent.



# QI phase

## Mapping

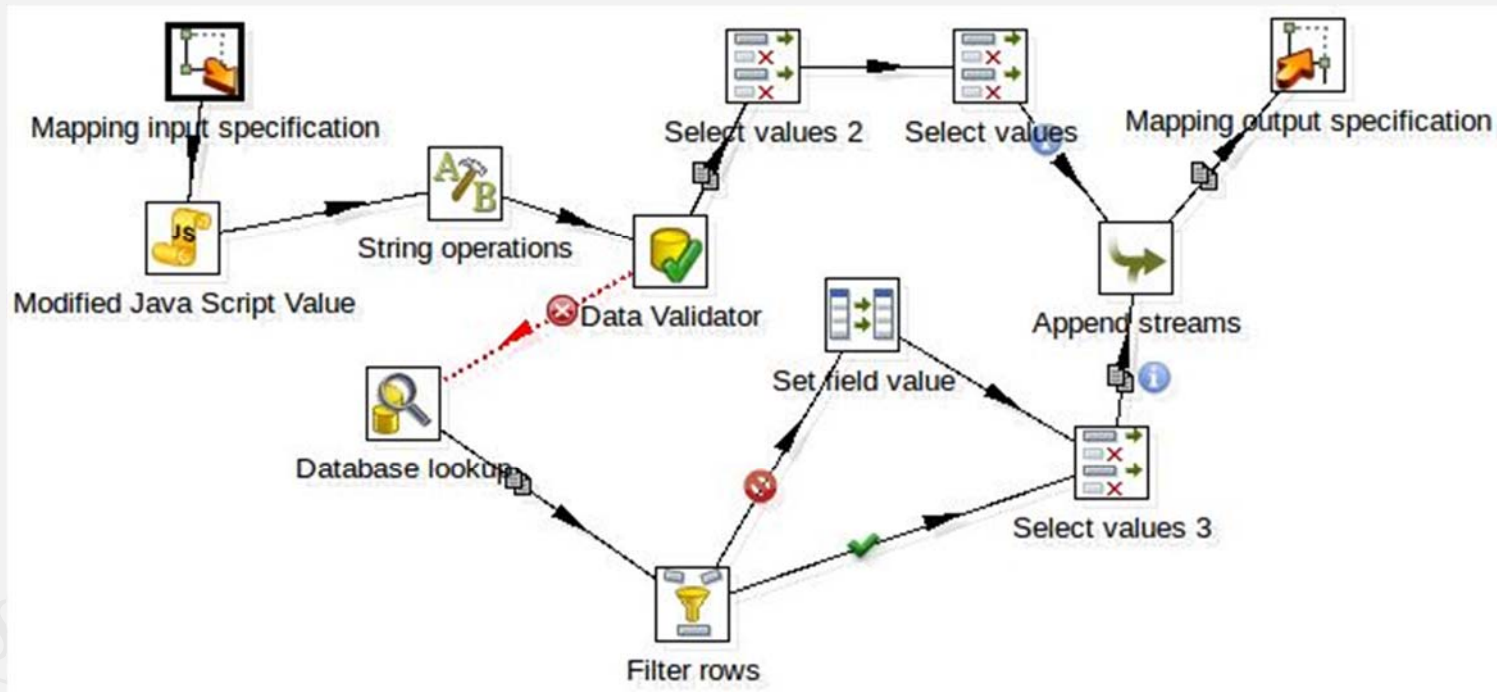
- To re-use a transformation you:
  - specify the sub-transformation to execute;
  - can define or pass Kettle variables down to the mapping;
  - can specify the input fields that are required by your sub-transformation;
  - can specify the output fields that are required by your sub-transformation.
- You can see this how a function that returns output values calculated on a specific input data.



# QI phase

## Mapping

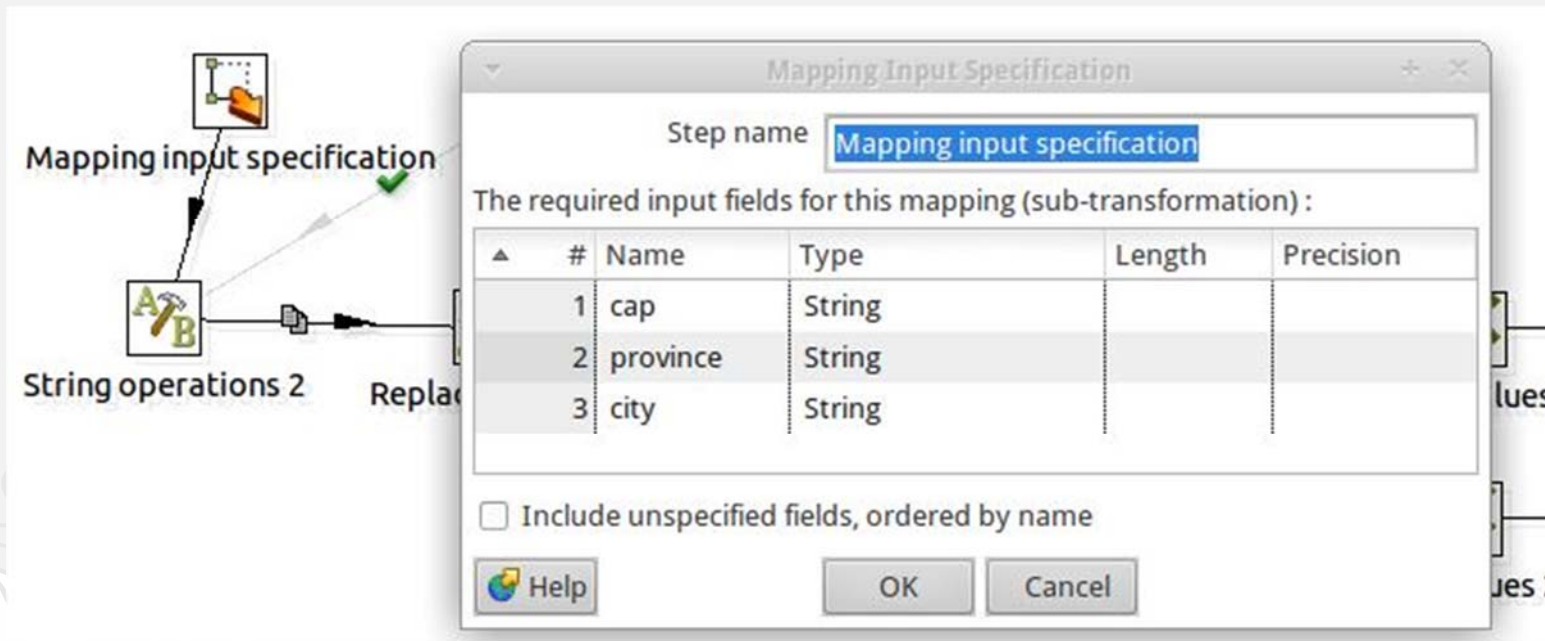
- A mapping is also called a sub-transformation because it is a transformation just like any other with a couple of key differences.



# QI phase

## Every mapping needs

- a Mapping Input step to define the fields that are **required** for the correct mapping execution.
- a Mapping Output step to define the fields that are **generated** by the mapping.



Mapping input specification

String operations 2

Replac

Mapping Input Specification

Step name Mapping input specification

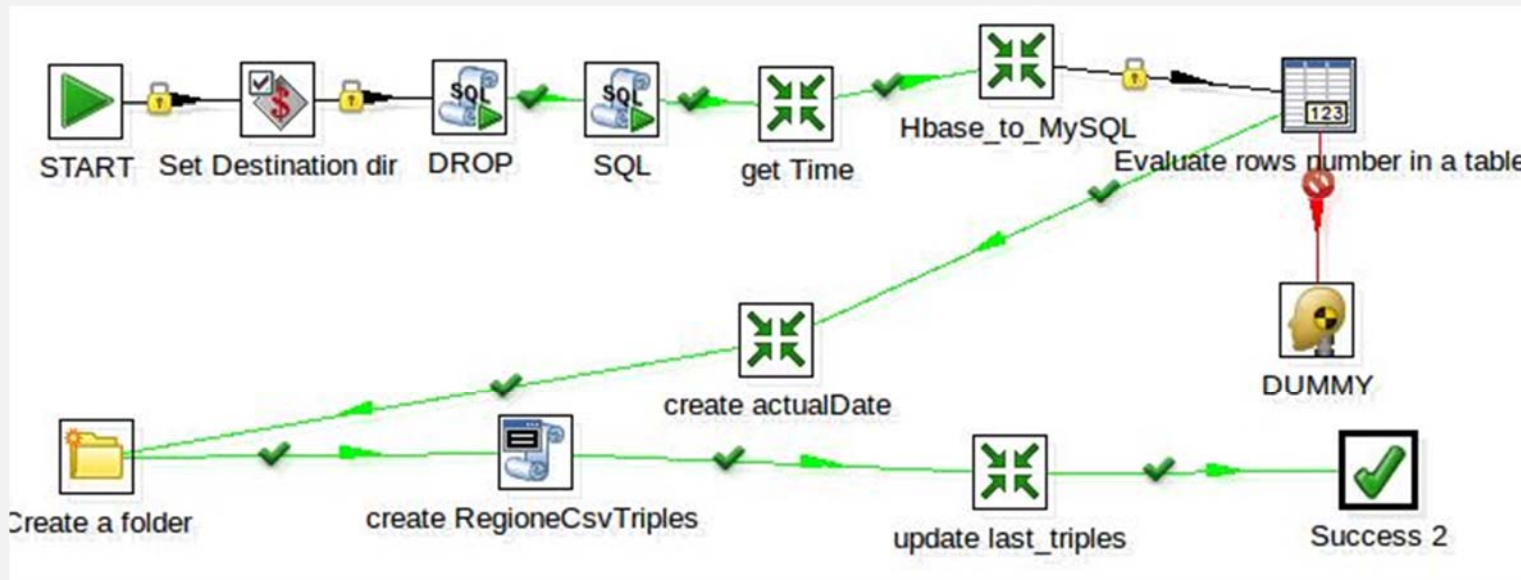
The required input fields for this mapping (sub-transformation):

#	Name	Type	Length	Precision
1	cap	String		
2	province	String		
3	city	String		

Include unspecified fields, ordered by name

Help OK Cancel

# RDF Triples generation phase

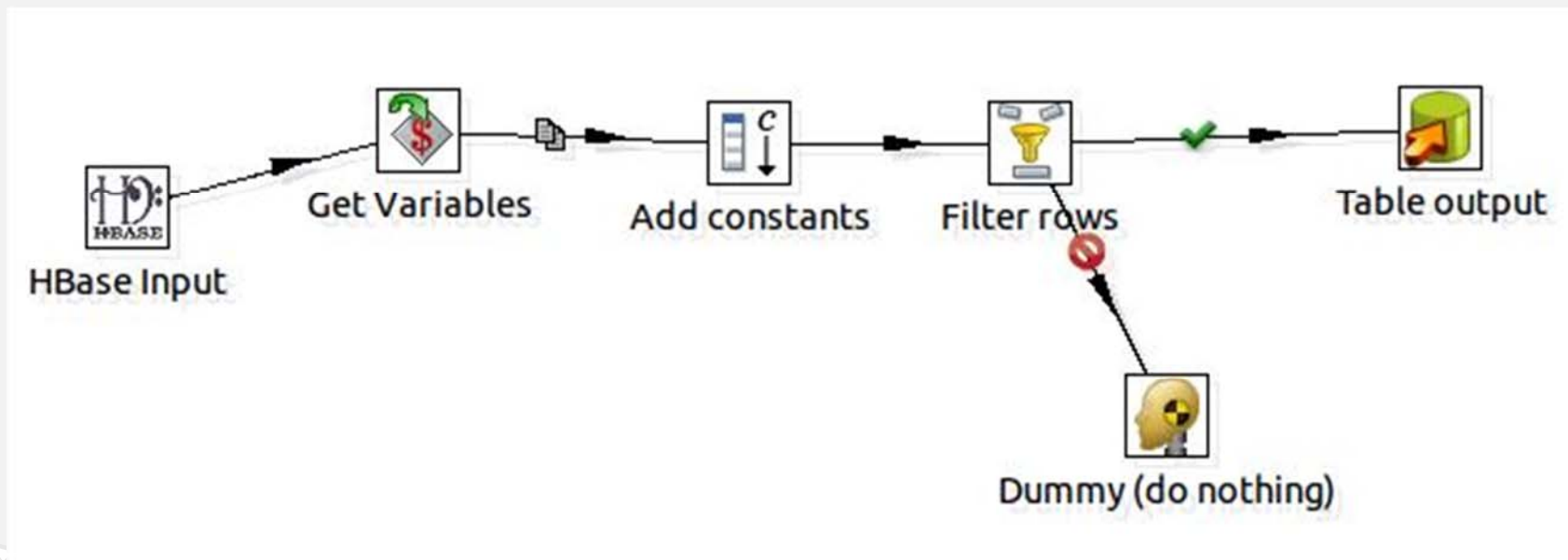


The triples are generated with the **km4city** ontology and then loaded on Virtuoso RDF store.



# RDF Triples generation phase

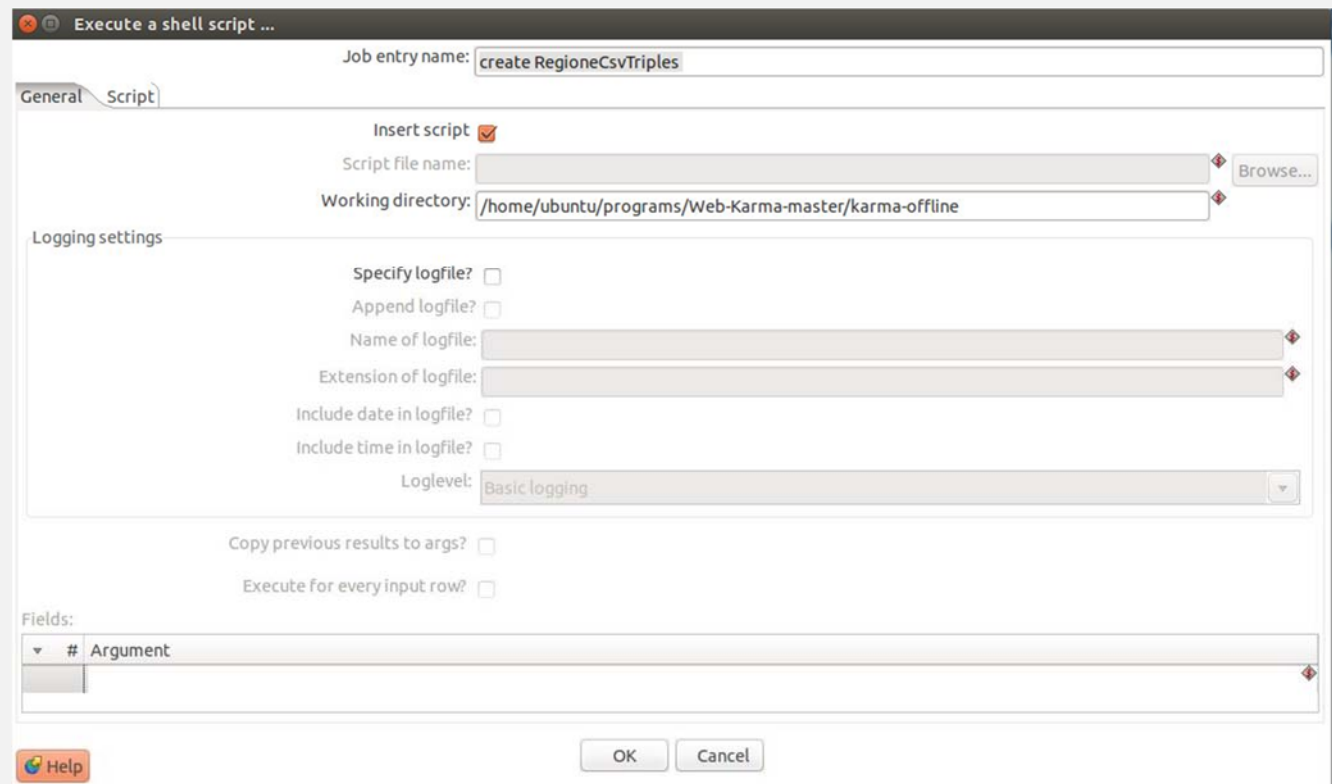
1. Load data from Hbase in order to copy them into a MySQL table.



# RDF Triples generation phase

2. Create triple RDF from data loaded through a specific script.

- You can use the **Execute a shell script Step**.

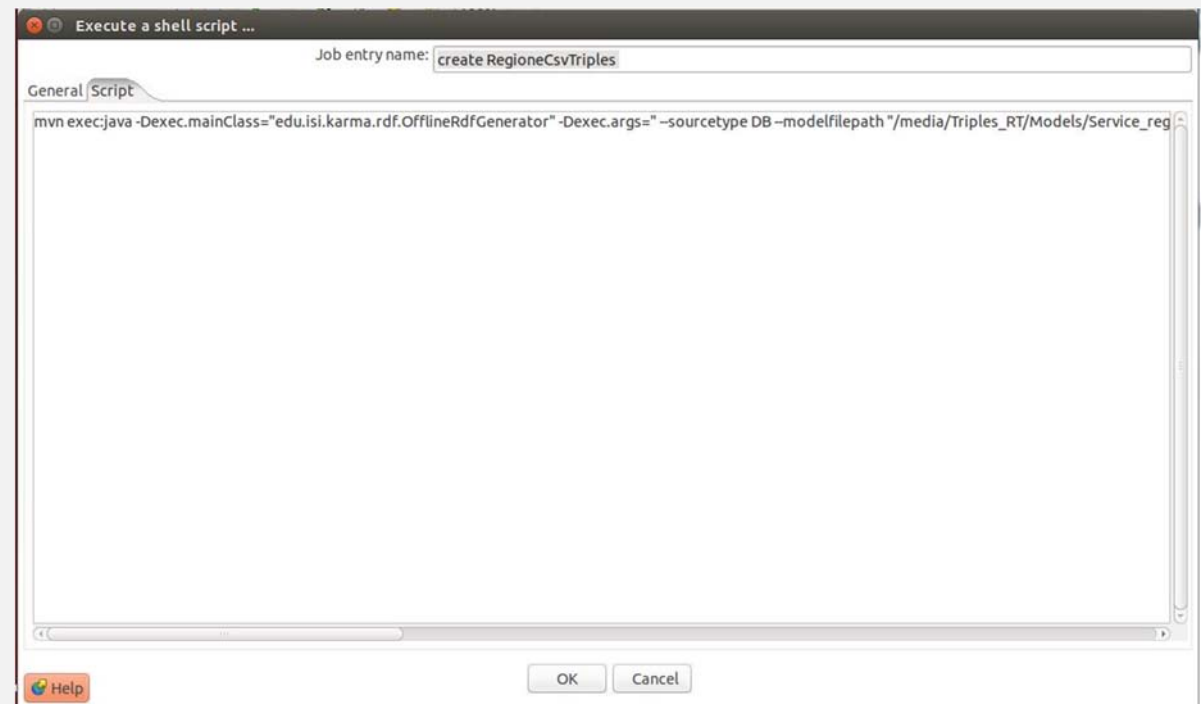
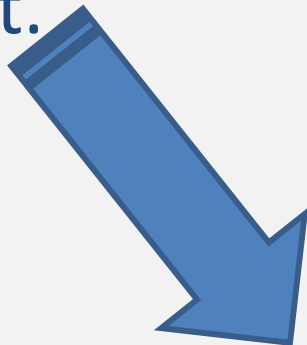


- You can check the option “Insert Script” if you want to execute the script in the Script tab instead of executing the Script file name.

# RDF Triples generation phase

## Execute a shell script Step

- Insert the specific command to execute the script.



```
mvn exec:java -Dexec.mainClass="edu.isi.karma.rdf.OfflineRdfGenerator" -Dexec.args="--sourcetype DB --
modelfilepath "/media/Triples_RT/Models/Service_region.ttl" --outputfile
${DestinationDir}/${processName}.n3 --dbtype MySQL --hostname 192.168.0.01 --username x --password x --
portnumber 3306 --dbname Mob --tablename ${processName}" -Dexec.classpathScope=compile
```

# RDF Triples generation phase

## Execute a shell script Step

```
mvn exec:java -Dexec.mainClass="edu.isi.karma.rdf.OfflineRdfGenerator" -  
Dexec.args=" --sourcetype DB --modelfilepath  
"/media/Triples_RT/Models/Service_region.ttl" --outputfile  
${DestinationDir}/${processName}.n3 --dbtype MySQL --hostname  
192.168.0.01 --username x --password x --portnumber 3306 --dbname Mob --  
tablename ${processName}" -Dexec.classpathScope=compile
```

- In input you specify the mapping model, the database table (where you get source data) and the connection parameters to database.
- In output you specify the file name (.n3) where the triples RDF will be stored.

# Transformation Parking

- To process the parking data for Sii-Mobility
  - real time data from Osservatorio Trasporti of Tuscany region (MIIC).



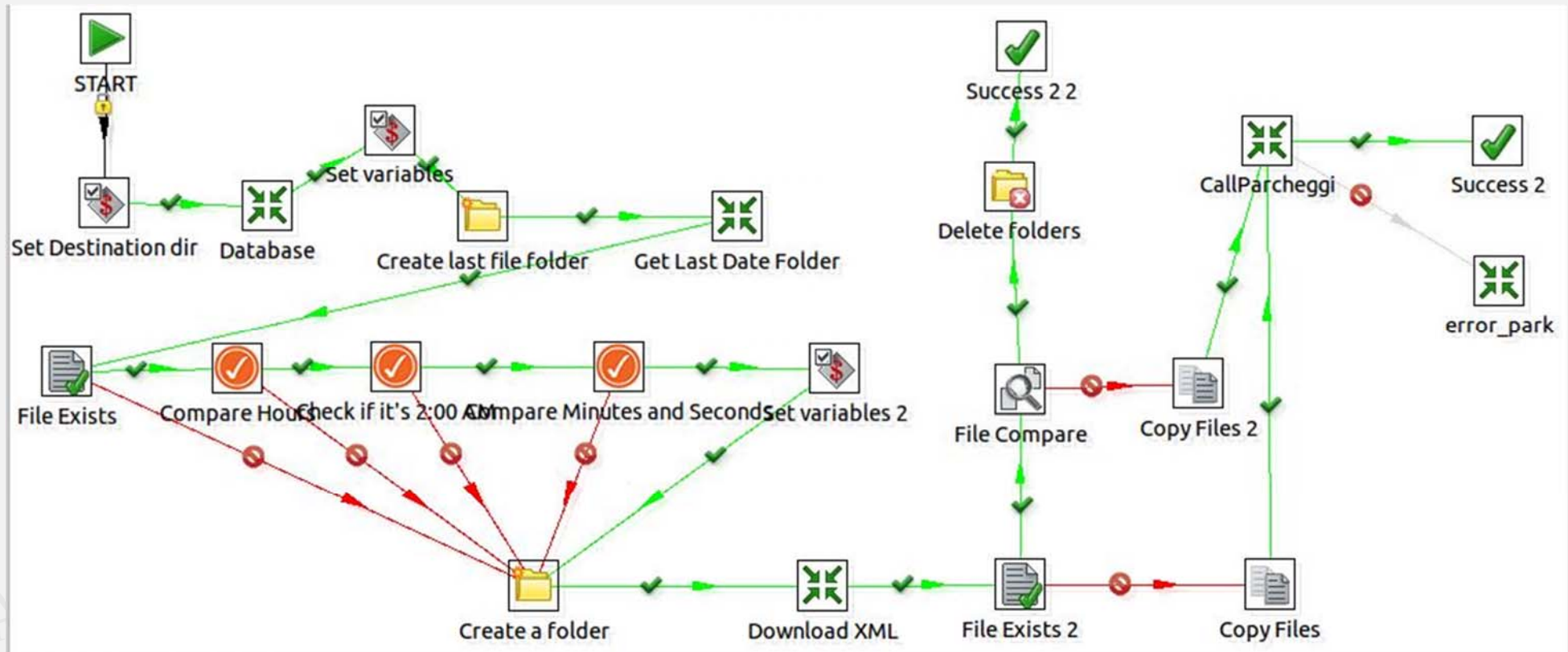
The screenshot shows the SIM Consultazione web interface. It features a navigation menu on the left with 'Classi Dati' expanded to show 'TPL', 'infrastrutture di trasporto', and 'Tempo reale'. The main content area is titled 'Consultazione Tempo reale' and contains a table with the following data:

Classe dati	Metadati	Interfaccia Input	Interfaccia Output	Validità	Consultazione	Mappa
Sensori				-		
Parcheggi				-		
Emergenze				-		
Rilievi AVM				-		
Meteo				-		

- 2 phases:
  - **INGESTION** phase;
  - **TRIPLES GENERATION** phase.

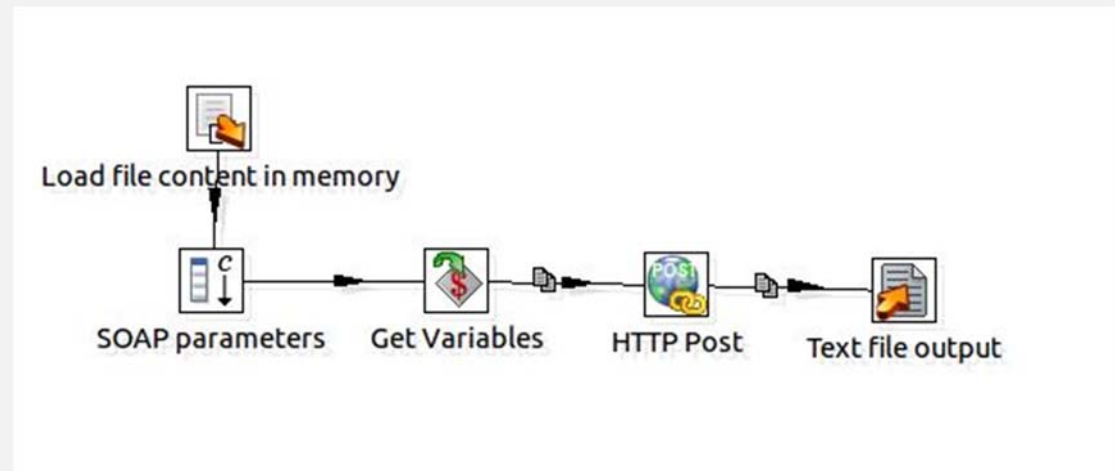
# Ingestion phase

To importing and storage data (in a database) for later use.



# Ingestion phase

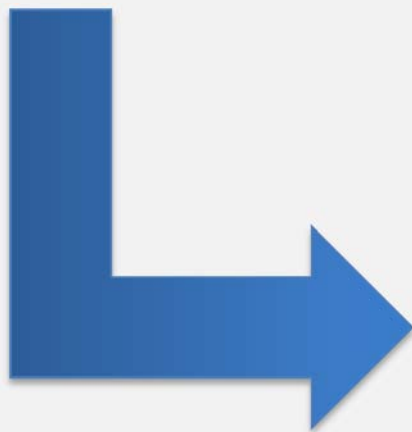
1. Taken an XML file (request.xml) that will be used to invoke the web service (forms the HTTP Post body)



2. Creation of static fields that are passed to HTTP post and HTTP headers such as SOAP action, content-type, username and password.

# Ingestion phase

3. Adding the parameter catalog to identify a sensors group.
4. Invocation of web service with **HTTP Post step**.
5. Storing data on HBase (CallParccheggi transformation).



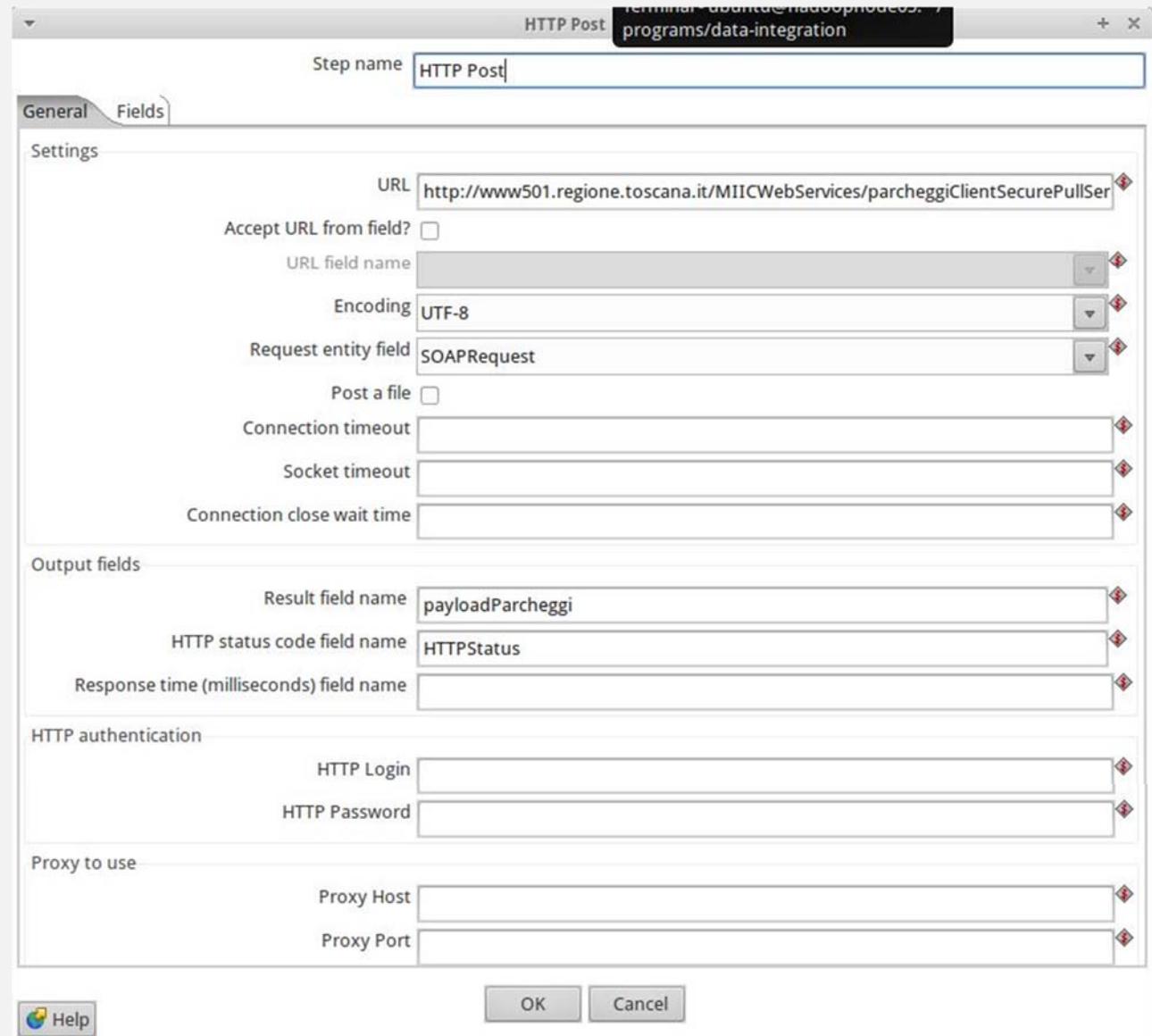
#	Alias	Key	Column family	Column name
1	FinalKey	Y		
2	actualDate	N	Family1	actualDate
3	carParkIdentity	N	Family1	carParkIdentity
4	carParkOccupancy	N	Family1	carParkOccupancy
5	carParkStatus	N	Family1	carParkStatus
6	catalog	N	Family1	catalog
7	exitRate	N	Family1	exitRate
8	fillRate	N	Family1	fillRate
9	numberOfVacantParkingSpaces	N	Family1	numberOfVacantParkingSpaces
10	occupiedSpaces	N	Family1	occupiedSpaces
11	process	N	Family1	process
12	situationRecordCreationTime	N	Family1	situationRecordCreationTime
13	situationRecordObservationTime	N	Family1	situationRecordObservationTime
14	supplierIdentification	N	Family1	supplierIdentification
15	timestamp	N	Family1	timestamp
16	totalCapacity	N	Family1	totalCapacity
17	validityStatus	N	Family1	validityStatus



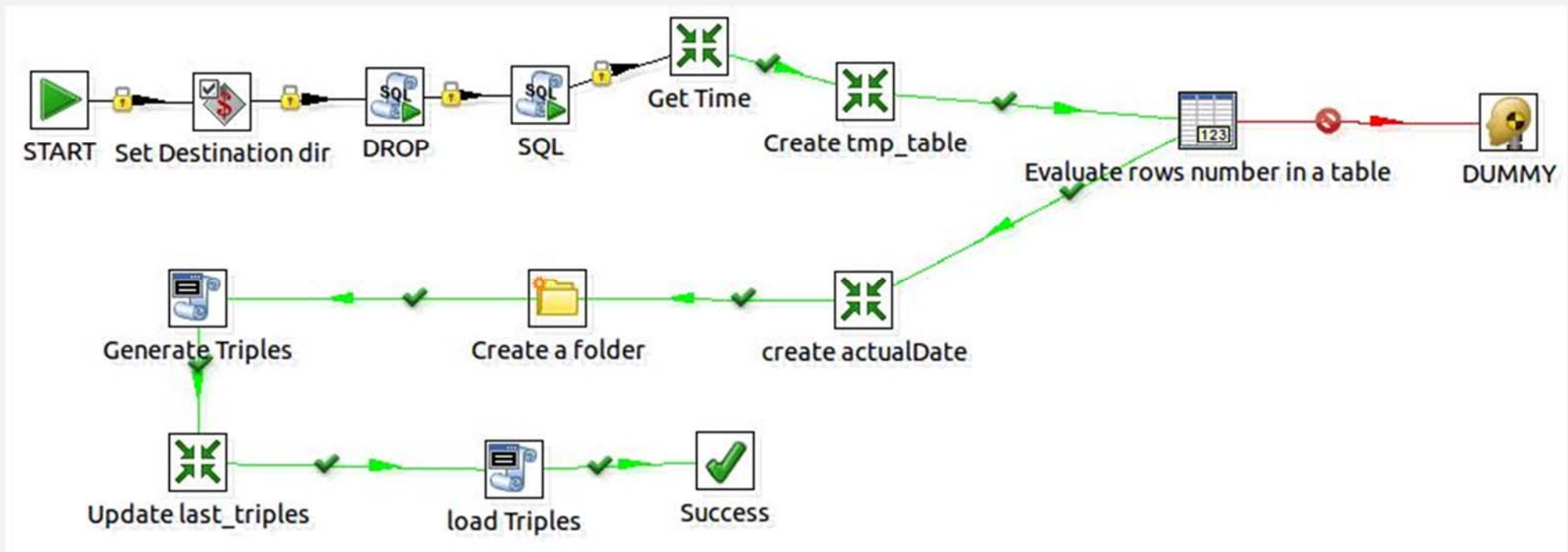
# Ingestion phase

## HTTP Post

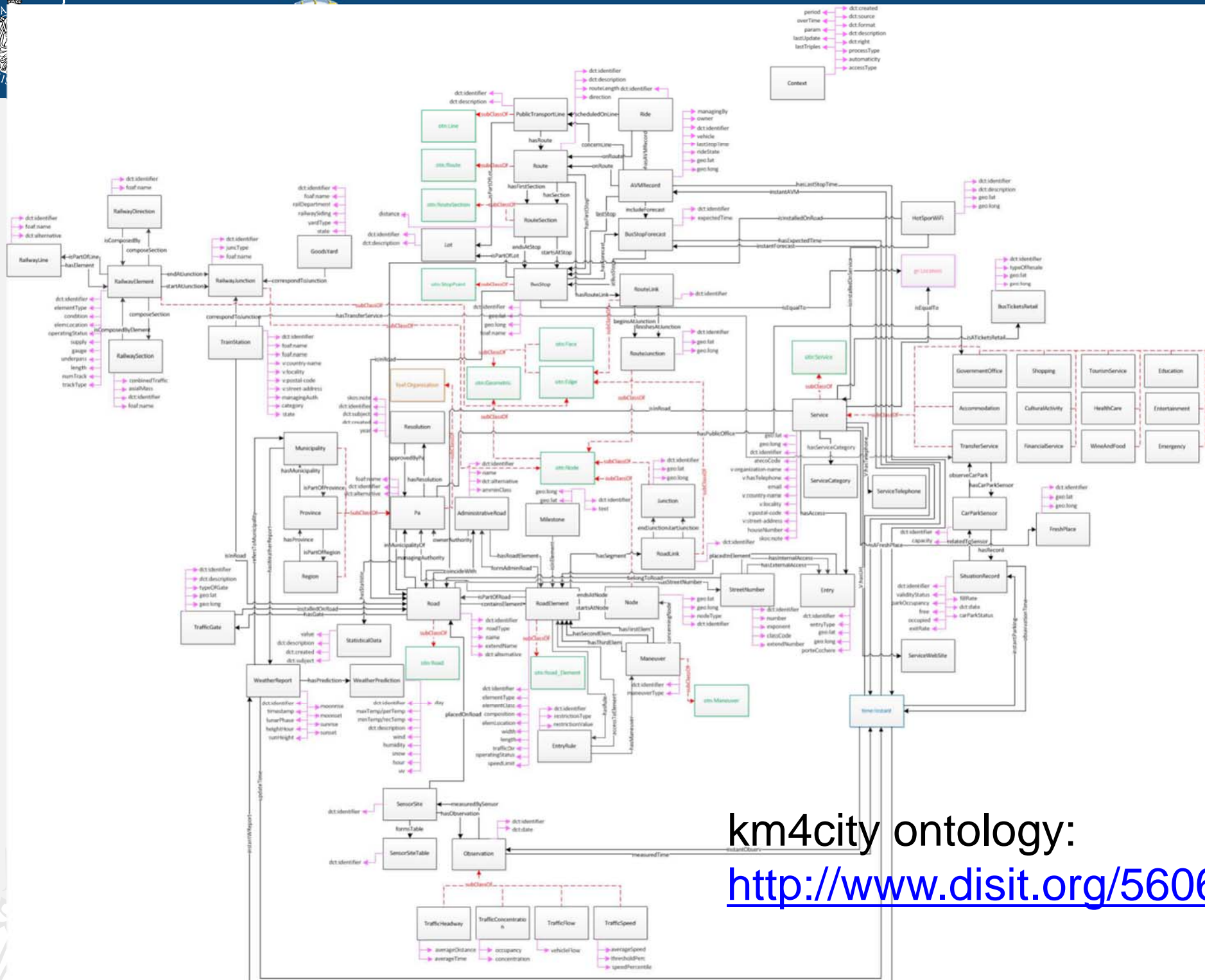
- This step performs the invocation of the web service using a SOAP (Simple Object Access Protocol) protocol.
- You can specify the service endpoint (URL).



# RDF Triples generation phase



The triples are generated with the **km4city** ontology  
(<http://www.disit.org/5606>) and then loaded on OWLIM  
RDF store.



km4city ontology:  
<http://www.disit.org/5606>

# Scheduler

- For **Real Time data** (car parks, road sensors, etc.) the ingestion and triple generation processes should be performed periodically (no for **static data**).
- Use of a scheduler to manage periodic execution of ingestion and triple generation processes.
  - This tool throws the processes with predefined interval determined in phase of configuration.



# RDF Triples generated

Macro Class	Static Triples	Real Time Triples loaded
Administration	2.431	0
Local Public Transport	644.405	0
Metadata	416	0
Point of Interest	471.657	0
Sensors (Traffic and parking)	0	11.111.078
Street-guide	68.985.026	0
Temporal	0	1.715.105
Total	70.103.935	12.826.183

**Triples monthly**

**21.691.882**

# Quality Improvement, QI

Class	%QI	Total rows	Class	%QI	Total rows
Accoglienza	34,627	13256	Georeferenziati	38,754	2016
Agenzie delle Entrate	27,124	306	Materne	41,479	539
Arte e Cultura	37,716	3212	Medie	42,611	116
Visite Guidate	38,471	114	Mobilita' Aerea	41,872	29
Commercio	42,105	323	Mobilita' Auto	38,338	196
Banche	41,427	1768	Prefetture	39,103	449
Corrieri	42,857	51	Sanità	42,350	1127
Elementari	42,004	335	Farmacie	42,676	2131
Emergenze	42,110	688	Università	42,857	43
Enogastronomia	42,078	5980	Sport	52,256	1184
Formazione	42,857	70	Superiori	42,467	183
Accoglienza	34,627	13256	Tempo Libero	25,659	564

**Service data** from Tuscany region.

**%QI** = improved service data percentual after QI phase.

# Process Work

	Service Data (static data)	Road / rail graph (static data)	MIIC (real time data)	LAMMA (real time data)	Total
DataSet	29	117	170	285	601
Processes	29	11	170	285	495

**MIIC:** parking data + traffic sensors data. Processes scheduled every 1800 sec.

**LAMMA:** Weather forecasts. Processes scheduled every 21600 sec.

**Service Data, Road / rail graph:** Processes started manually.

4

# ETL processes implementation

Developing  
ETL  
processes  
with PDI





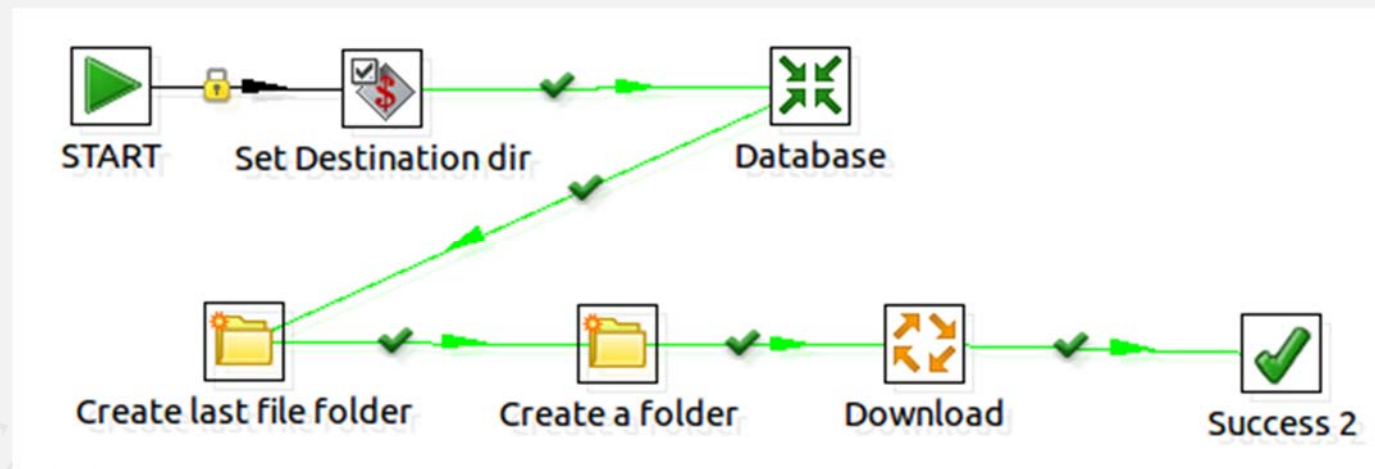


# First phase



# Job Data Ingestion CSV

- This job acquires the source file from Open Data Portal of Tuscany region, stores it in a specific folder and loads data in specific HBase table.
  - The acquired dataset is “Strutture ricettive” in CSV format.



# Phase 1: Set storage root folder

- PDI step: **Set Destination dir**
- Set the variable that indicates the root of the path where the source file will be stored. The relative value should be **/Sources/Categoria**.



Set variables...

Job entry name:

---

Properties file

Name of properties file

Variable scope

---

Settings

Variable substitution?

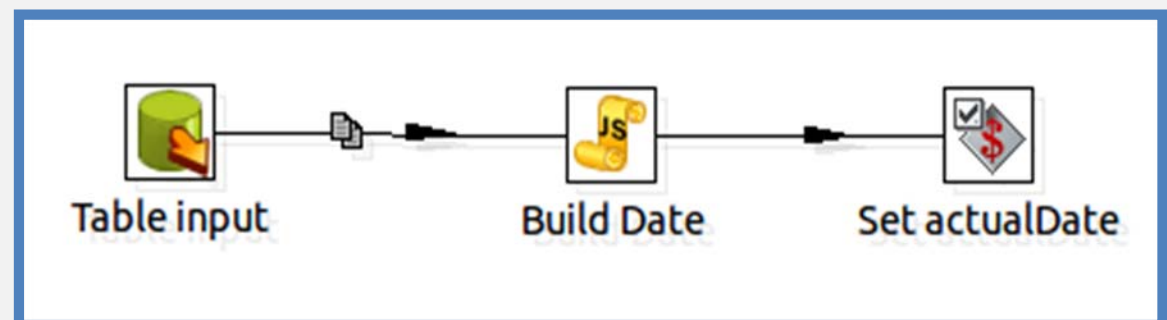
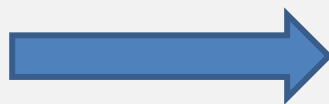
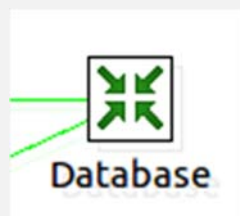
---

Variables :

▲	#	Variable name	Value	Variable scope type
	1	DestinationDir	/media/Sources/Elaborato	Valid in the root job

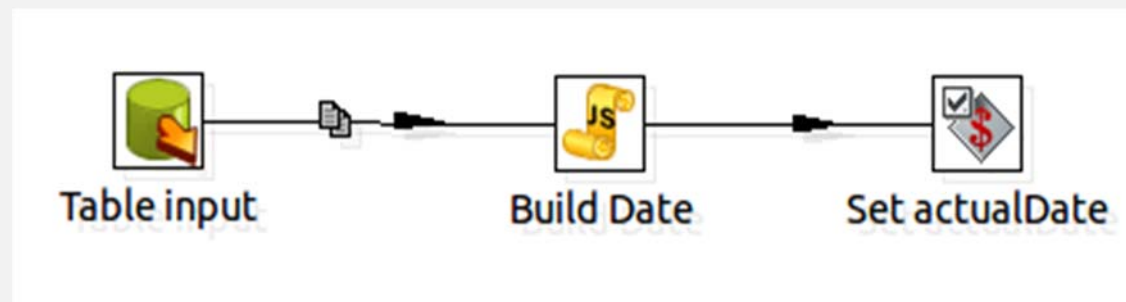
# Phase 2: Creating storage folder

- PDI step: Transformation executor
- This step allows you to execute a transformation within the job. In this case the invoked transformation is **Database.ktr**.



# Phase 2: Creating storage folder

- PDI steps: Table input, Modified Java Script value, Set variables
- The invoked transformation extracts from current date the variables to build the path in which the downloaded file will be stored. The path must be **/Sources/Categoria/NomeProcesso/Anno\_mese/Giorno/Ora/MinutiSecondi/file.xxx** .



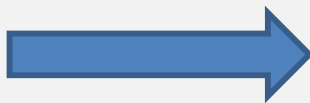
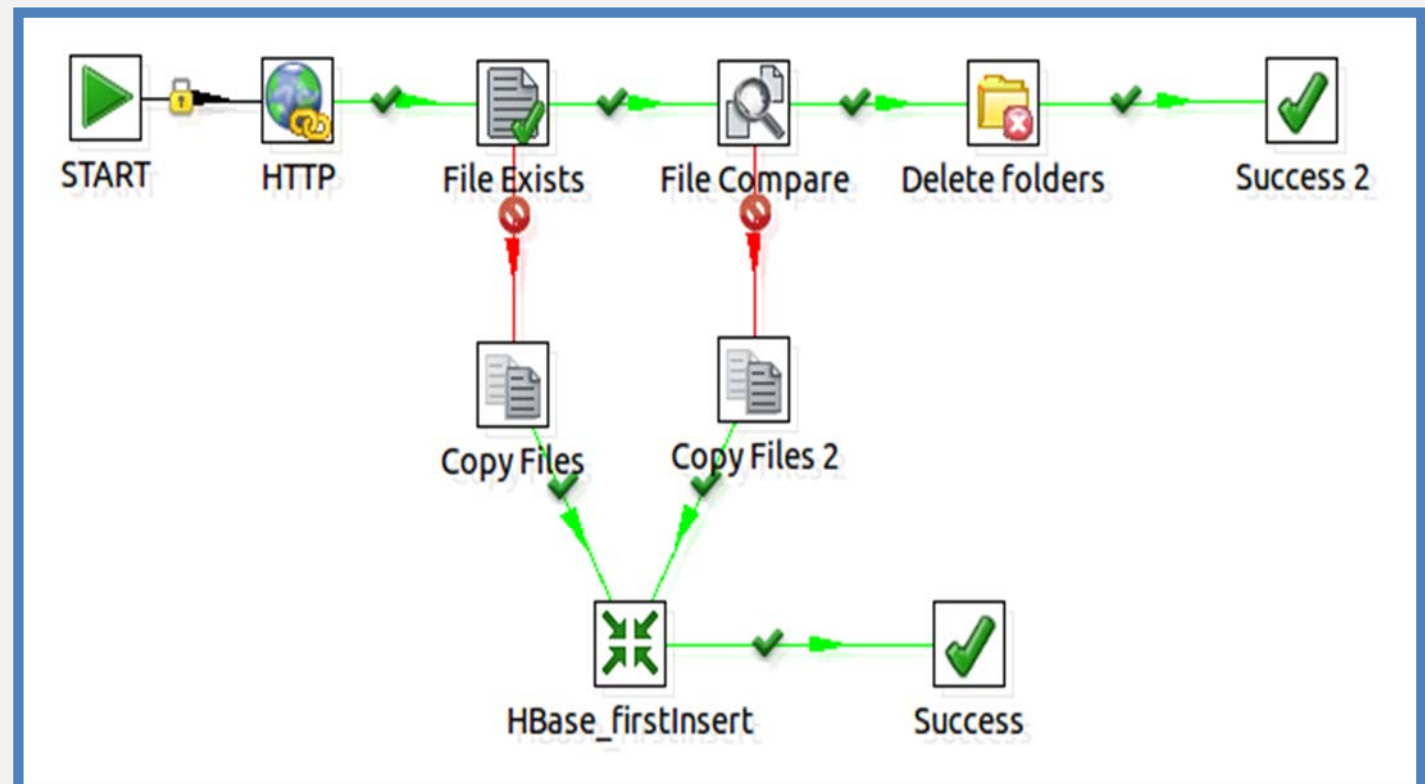
# Phase 2: Creating storage folder

- **PDI steps: Create a folder**
- Create the folders that will host the source files and the copy of latest version of file that you downloaded.



# Phase 3: Download/Store dataset

- PDI step: Job
- With this step the main job can launch another job (in this case **Download.kjb**).



# Phase 3: Download dataset

- PDI step: HTTP post
- Set **URL** to download the interest file and **Target file** that defines the file name (Strutture ricettive.csv) and the storage path built previously.



URL:

contains URL

Username:

Password:

Port for upload:

Proxy port:

Proxy for hosts:

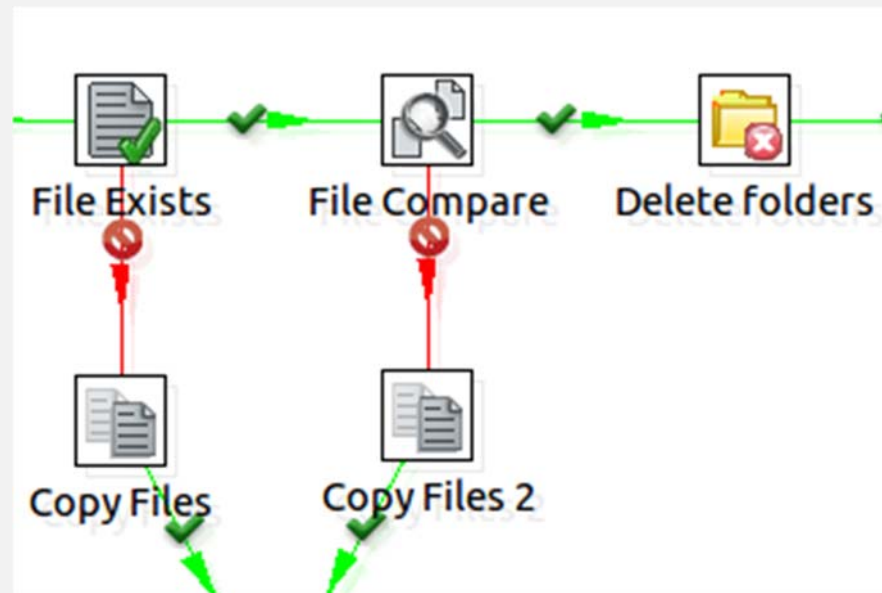
Upload file:

Target file:



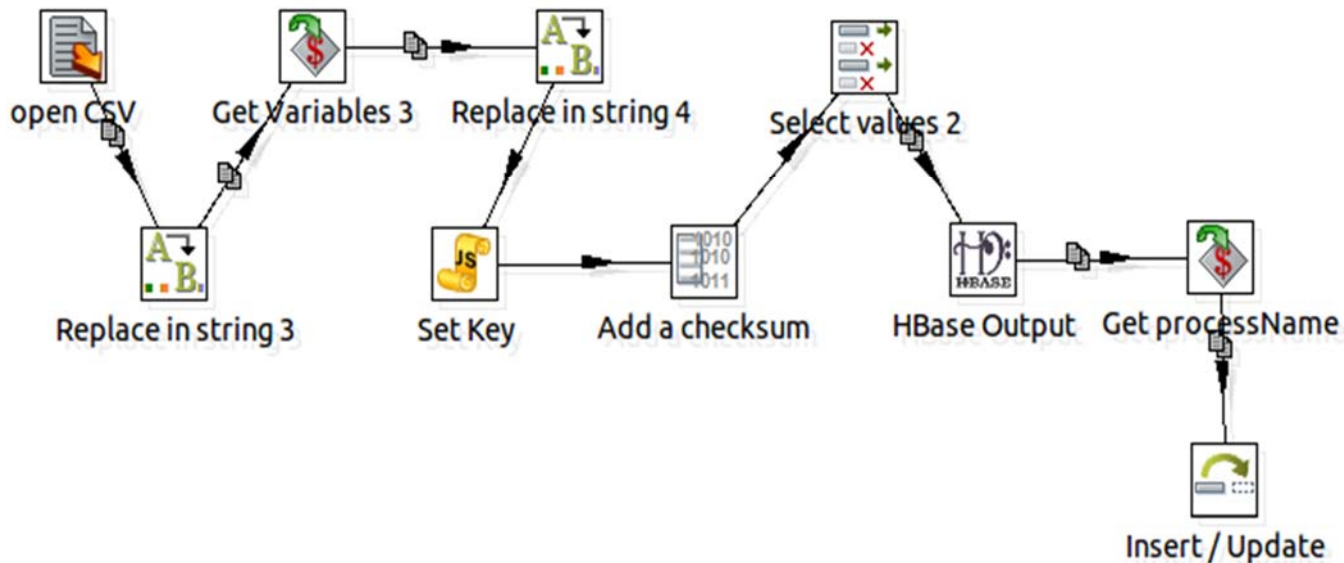
# Phase 3: Store dataset

- **PDI steps: File exists, File compare, Copy files, Delete folders**
- Check if the downloaded file already exists for not storing identical copies.



# Phase 4: Load data into HBase

- **PDI step: Transformation**
- The invoked transformation (Hbase\_firstInsert) extracts the interest fields from downloaded source file, sets the storage key and load data into a specific HBase table.



# Phase 4: Load data into HBase

- PDI step: Text file input
- Select the CSV file (Strutture ricettive.csv), choose the separator type used and select the fields to be imported.



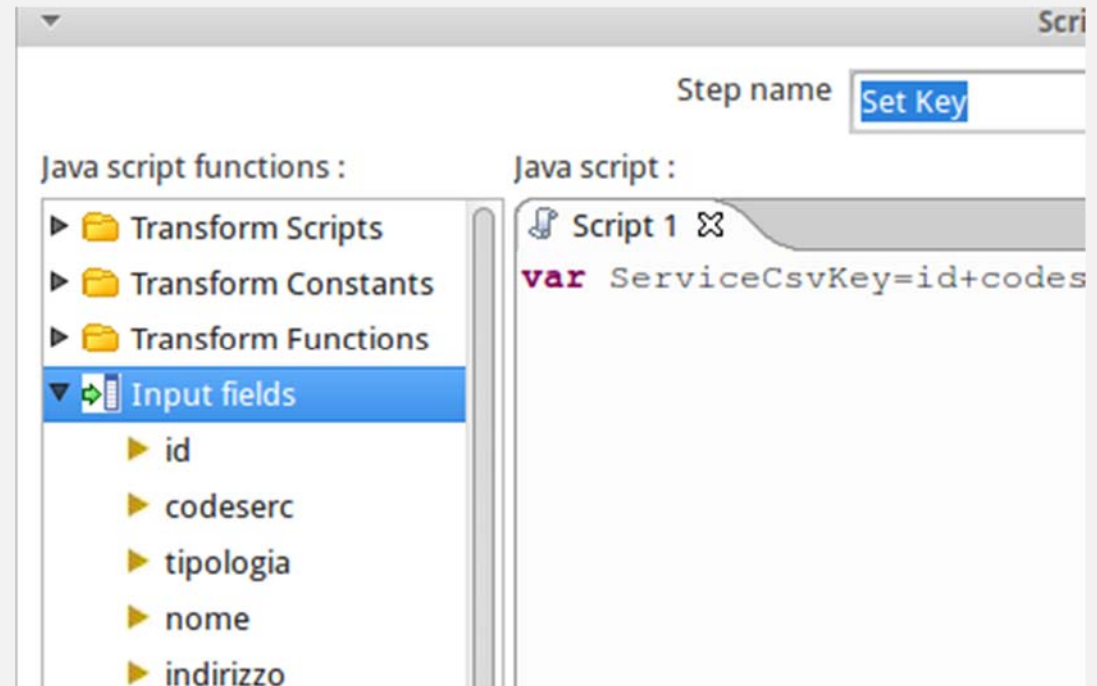
Step name

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

▲	#	Name	Type	Format	Position	Length
	1	id	String			
	2	codeserc	String			
	3	tipologia	String			
	4	nome	String			
	5	indirizzo	String			
	6	cap	String			
	7	citta	String			

# Phase 4.1: Defines a key

- PDI step: **Modified Java Script value**
- Define a variable by concatenating the input fields to define the **storage key** field.

Step name: Set Key

Java script functions:

- Transform Scripts
- Transform Constants
- Transform Functions
- Input fields**
  - id
  - codeserc
  - tipologia
  - nome
  - indirizzo

Java script:

```
var ServiceCsvKey=id+codeserc
```

# Phase 4.1: Defines a key

- **PDI step: Add a checksum**
- Choose which algorithm (MD5, CRC32) to use to encode the **storage key** field (the result will be a new output field).



+ ×

Step name

Type

Result type

Result field

Compatibility Mode

Fields used in checksum

#	Field
1	ServiceCsvKey

# Phase 4.2: Load data

- PDI step: **Select values**
- Select the fields (one or more) that you want to load into HBase.



Step name

Select & Alter Remove Meta-data

Fields :

▲ #	Fieldname	Rename to	Length	Precision
1	id			
2	codeserc			
3	tipologia			
4	nameFix2	nome		
5	indirizzo			
6	cap			
7	citta			

## Phase 4.2: Load data

- **PDI step: Hbase output**
- Perform the real data storage in HBase table specifying the Zookeeper host (on the local 127.0.0.1) and the port (2181).
- Specify the HBase storage table.
- Define a mapping of the input fields on the specific HBase table.



HBase output

Step name

Configure connection
Create/Edit mappings

Zookeeper host(s)

Zookeeper port

URL to hbase-site.xml

URL to hbase-default.xml

HBase table name

Mapping name

Store mapping info in step meta data

Disable write to WAL

Size of write buffer (bytes)

# Phase 4.3: Update MySQL

- **PDI step: Insert / Update**
  1. create a connection to MySQL database by specifying the connection name and type, the host name, the database name, the port number, and the access username and password.
  2. choose the MySQL table where data are written;
  3. specify the table fields (one or more) to update (in this case the field **last update**).



**Insert / Update**

Step name:

Connection:

Target schema:

Target table:

Commit size:

Don't perform any updates:

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

▲	#	Table field	Comparator	Stream field1	Stream f
	1	process	=	process	

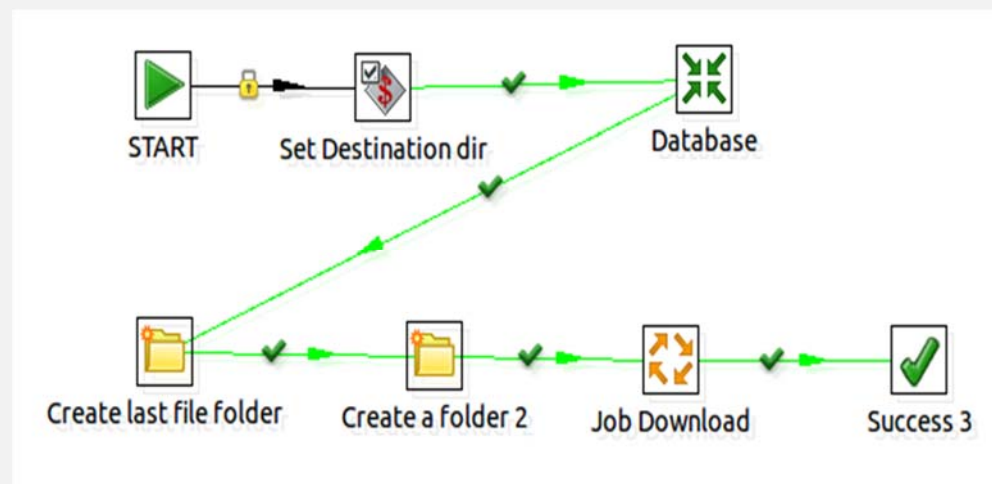
Update fields:

▲	#	Table field	Stream field	Update
	1	last_update	actualDate	Y



# Job Data Ingestion KMZ

- This job acquires the source file from Open Data Portal of Florence municipality, stores it in a specific folder and loads data in a specific HBase table.
  - The acquired dataset is “Distributori\_di\_carburante ” in KMZ format.
  - The Job is similar to the previous one, but differs in some points.



# Download.kjb 1/2

- In step **HTTP** the field URL is set using the variable `${PARAM}` containing the value retrieved from MySQL table. This value is the web address for download the source files.



Transfe

Name of job entry:

General Headers

URL:

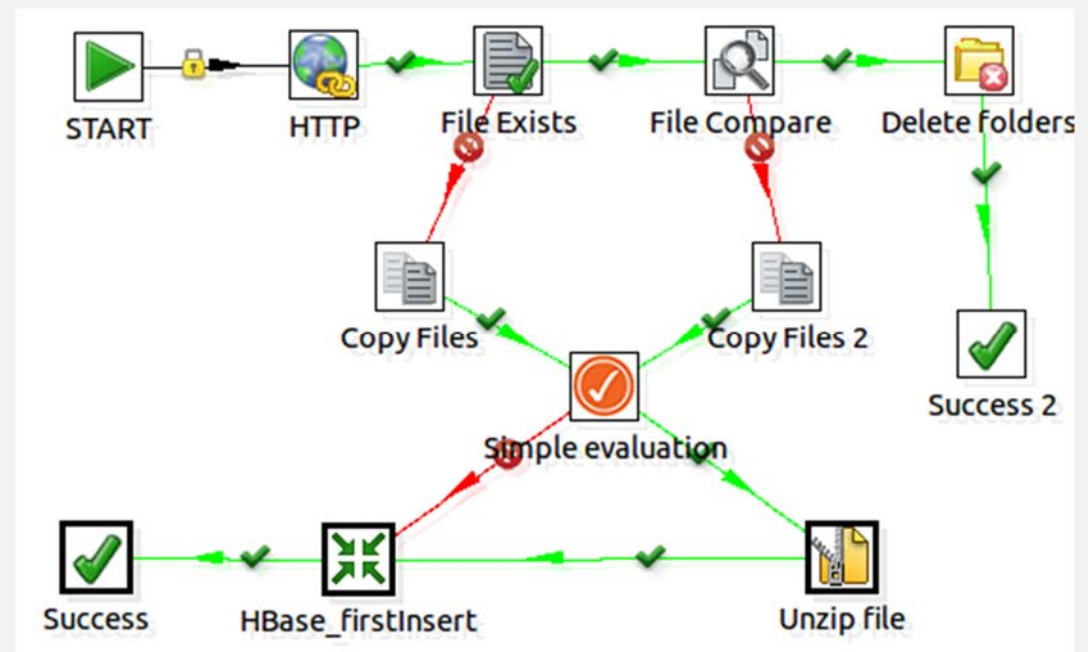
Run for every result row?

Input field which contains URL

Authentication

# Download.kjb 2/2

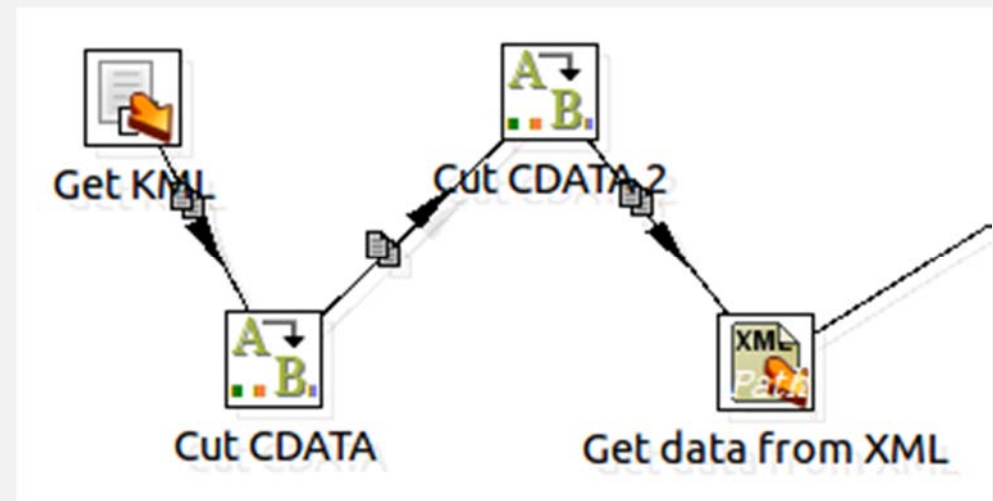
- The new step **simple evaluation** checks the format of the dataset downloaded. For KMZ files the execution flow is routed to step **Unzip file** that extracts the KML file and stores it in the folder `${DestinationDir}/${processName}/1Last_file`.



# HBase\_firstinsert 1/2

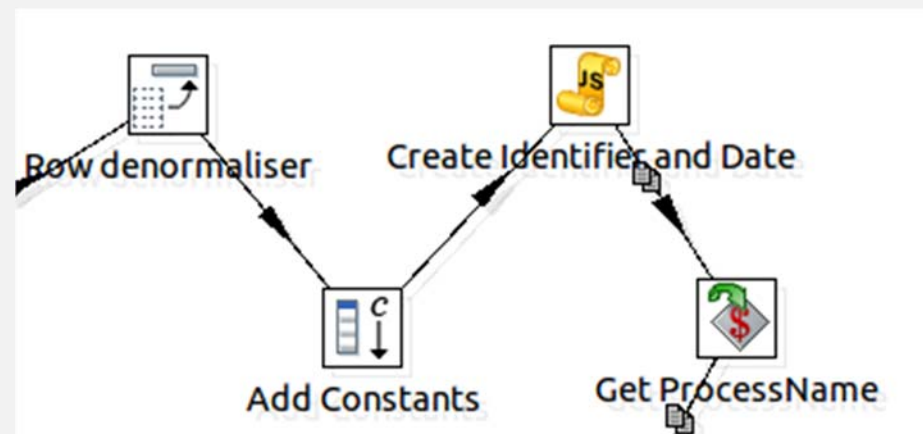
- The step **Get KML** (type Load content file in memory) retrieves the entire contents of the downloaded source file, which will be saved within the field Field Content.

After some modifications, the output field is passed to the step **Get data from XML** and the fields tag, span and coordinates are extracted through the use of XPath .



# HBase\_firstinsert

- The step **Row denormaliser** creates a stream of rows: a line composed of fields ID, CODSTRADA and NAME (taken from the field span defined in the previous step) for each instance of the field coordinates.
- For each line are then defined three new fields set with fixed values: locality, country-name and initials.



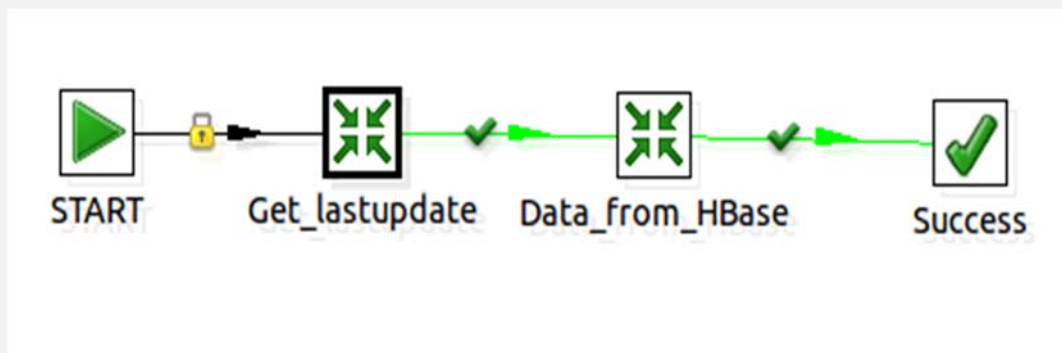


# Second phase



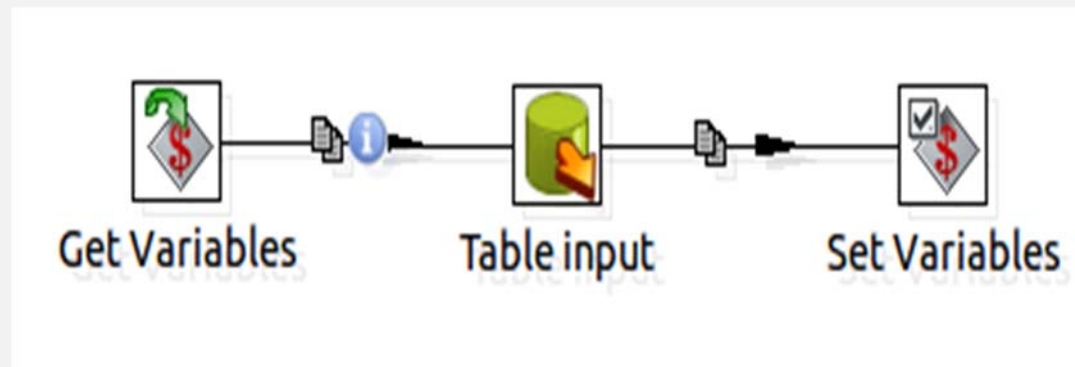
# Job Quality Improvement CSV

- This job reads data from HBase table created in Data Ingestion phase, improves the quality of data and re-load data into HBase database (in new table).
- This job invokes two transformations to:
  - get dataset update field;
  - apply the quality improvement (QI) to fields stored in HBase table. Each field has its own QI transformation.



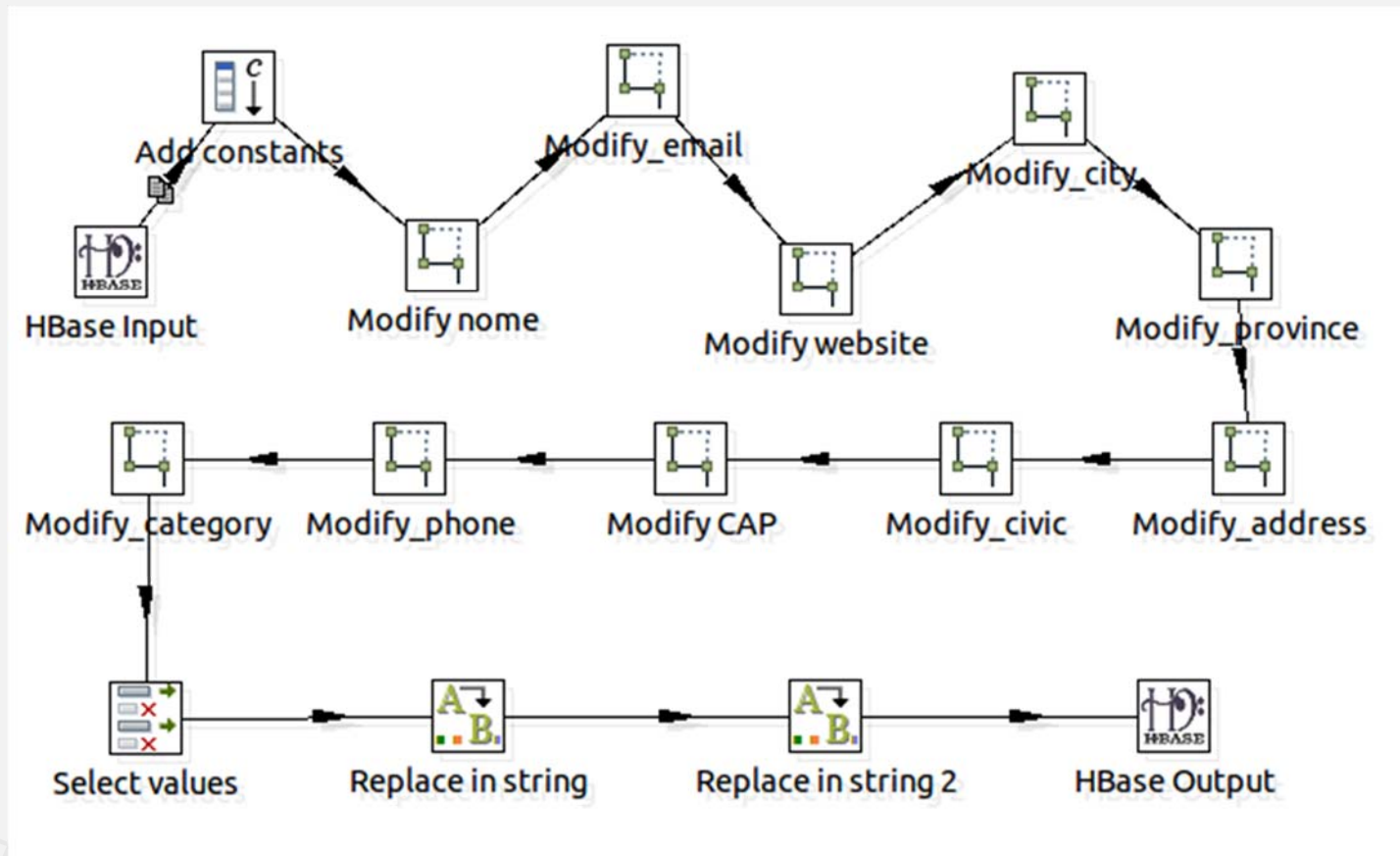
# Phase 1: Get dataset update field

- PDI step: **Get variables, Table Input, Set variables**
- Retrieve last update field from MySQL table (starting from process name passed as parameter) and set it as variable.





# Phase 2: Data Quality Improvement



# Phase 2.1: Reading HBase table

- PDI step: HBase input
- Retrieve data from specific HBase table specifying the Zookeeper host (on the local 127.0.0.1), the port (2181) and the mapping. The table and the mapping are those defined in ingestion phase.



HBase input

Step name

Configure query | Create/Edit mappings | Filter result set

Match all  Match any

▲	#	Alias	Type	Operator	Comparison value	F
	1	process	String	Substring	\${processName}	

## Phase 2.2: Add new fields

- **PDI step: Add constants**
- Define new fields that were not present in the source file (in this case notes and categoryEng).



▼ Add

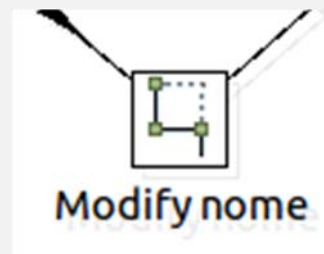
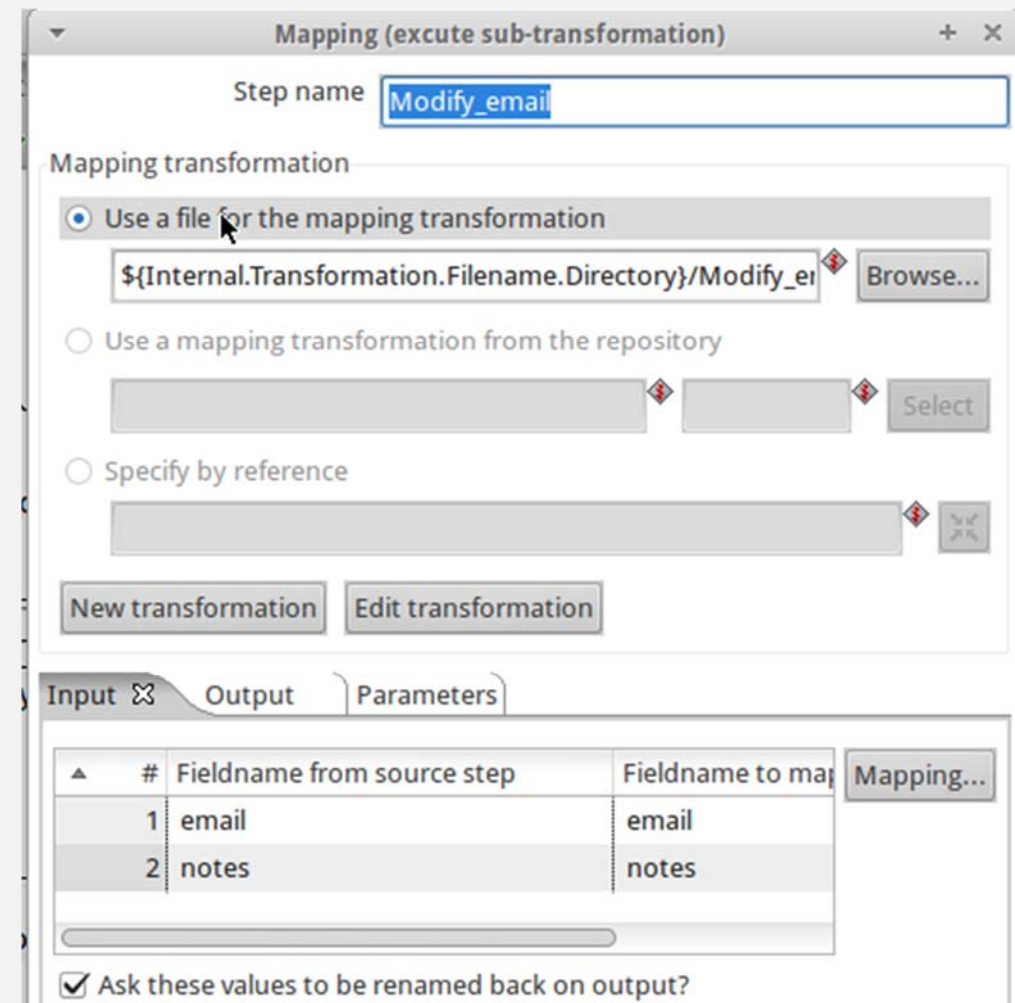
Step name

Fields :

▲ #	Name	Type	Format	Length	Precision
1	notes	String			
2	categoryEng	String			

# Phase 2.3: QI Field

- **PDI step: Simple mapping**
- Invoke the QI transformation for a specific field. First, set the mapping between the fields of the input/output stream and those defined inside the transformation invoked.

Mapping (excute sub-transformation)

Step name

Mapping transformation

Use a file for the mapping transformation

Use a mapping transformation from the repository

Specify by reference

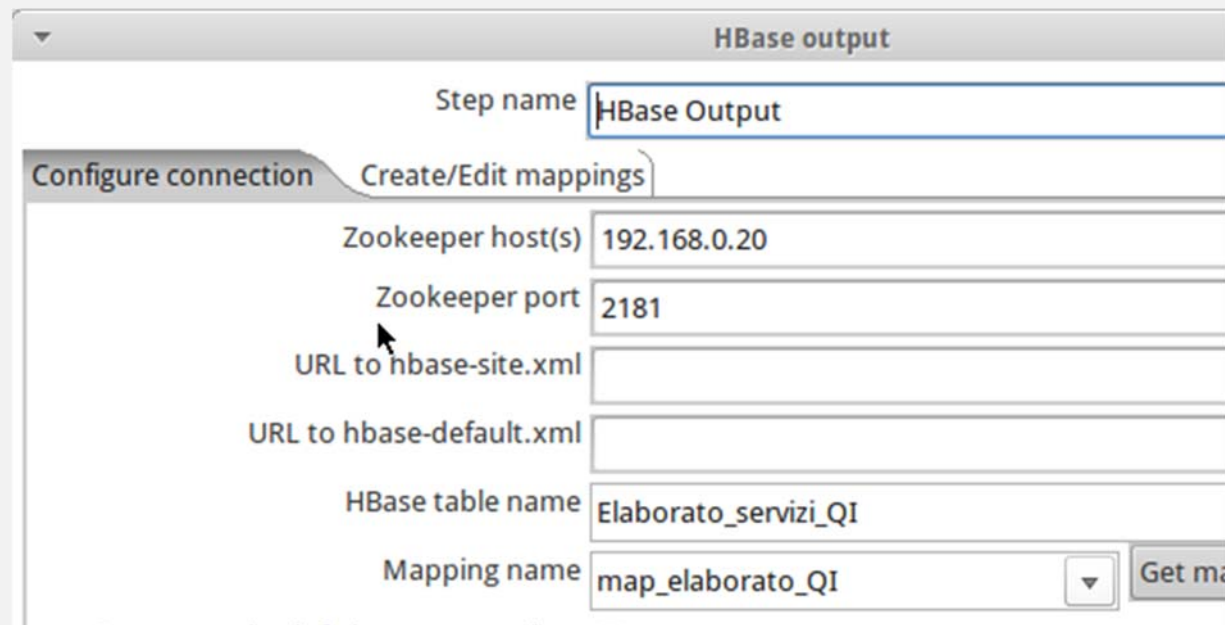
Input  Output  Parameters

#	Fieldname from source step	Fieldname to map	Mapping...
1	email	email	
2	notes	notes	

Ask these values to be renamed back on output?

# Phase 3: Load data into HBase

- **PDI step: HBase output**
- Perform the storage of improved data (after QI phase) in new table of HBase database.
- For the new table define a new mapping of the input fields.



HBase output

Step name

Configure connection | Create/Edit mappings

Zookeeper host(s)

Zookeeper port

URL to hbase-site.xml

URL to hbase-default.xml

HBase table name

Mapping name

# Job Quality Improvement KMZ

- This job reads data from HBase table created in Data Ingestion phase, improves the quality of data and re-load data into HBase database (in new table).
- This job invokes 3 transformations to:
  - get dataset update field;
  - apply the quality improvement (QI) to fields stored in Hbase;
  - in this case the dataset contains geotagged information (address and geographical coordinates), so it is made a reconciliation retrieving the street code (codice toponimo) from MySQL table.

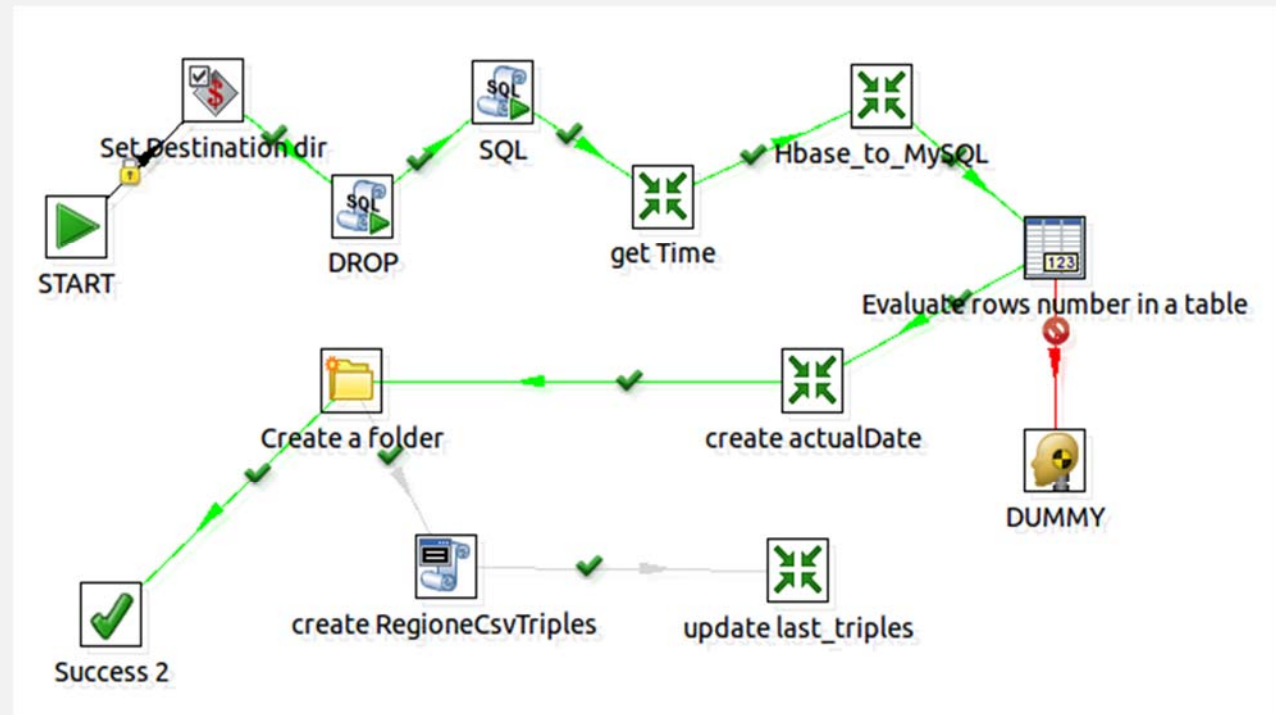


# Third exercise



# Job Triplification

- This job generates RDF triples from QI data based on a model built on relationships that are defined within a specific reference ontology. The RDF triples are stored in file in n3 format.





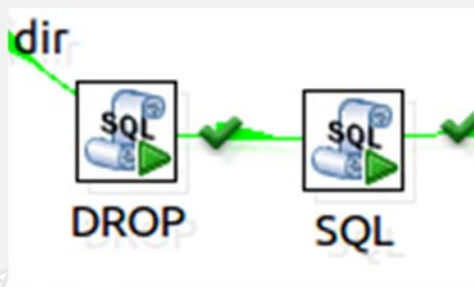
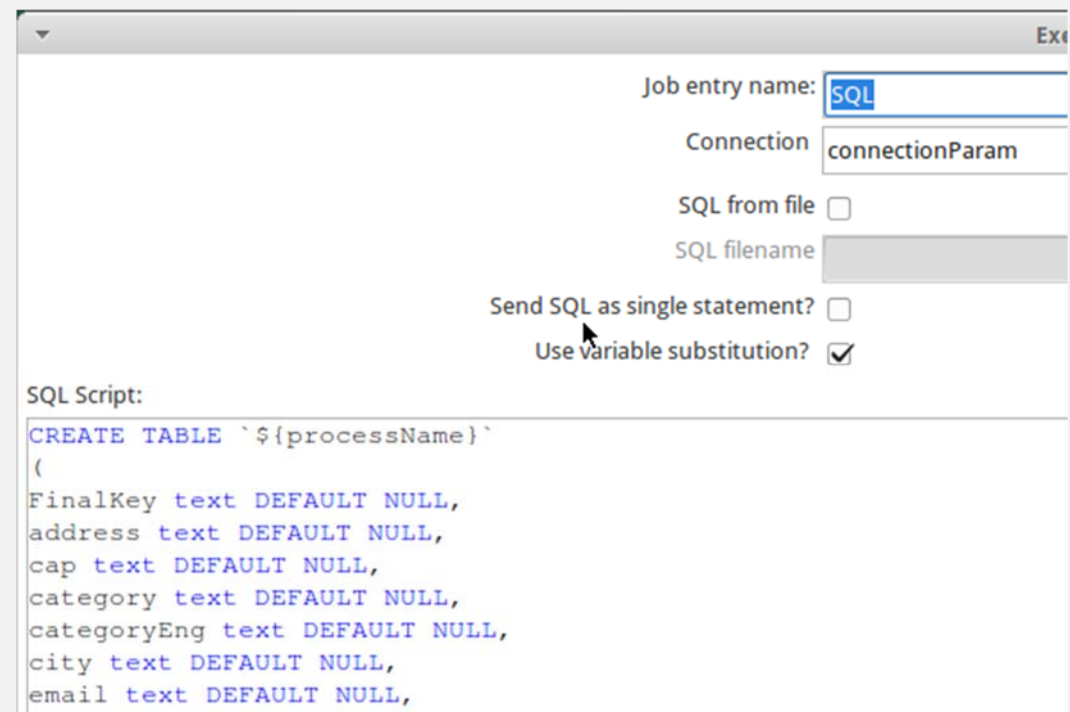
# Phase 1: Set storage root folder

- **PDI step: Set Destination dir**
- Set the variable that indicates the root of the path where the n3 triples files will be stored. The relative value should be **/Triples/Categoria**.



# Phase 2: Set temporary Mysql table

- PDI steps: SQL job entry
- Create a temporary MySQL table that will contain data extracted from HBase. If the table already exists, it is first deleted and then recreated.

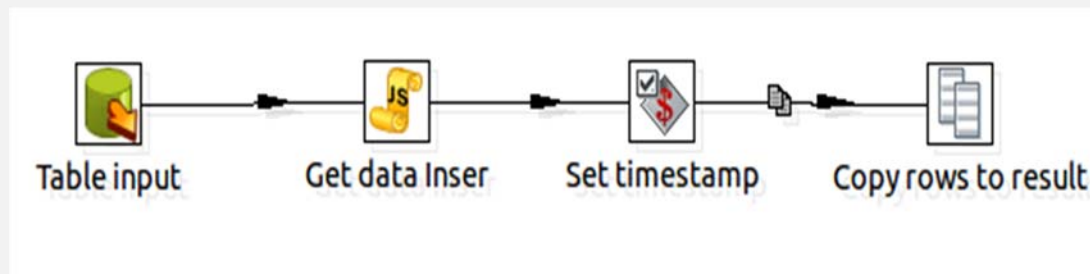



The screenshot shows the configuration for an SQL job entry. The 'Job entry name' is 'SQL', and the 'Connection' is 'connectionParam'. The 'SQL from file' checkbox is unchecked, and the 'SQL filename' field is empty. The 'Send SQL as single statement?' checkbox is unchecked, and the 'Use variable substitution?' checkbox is checked. The 'SQL Script' field contains the following SQL code:

```
CREATE TABLE `${processName}`
(
  FinalKey text DEFAULT NULL,
  address text DEFAULT NULL,
  cap text DEFAULT NULL,
  category text DEFAULT NULL,
  categoryEng text DEFAULT NULL,
  city text DEFAULT NULL,
  email text DEFAULT NULL,
  ...
```

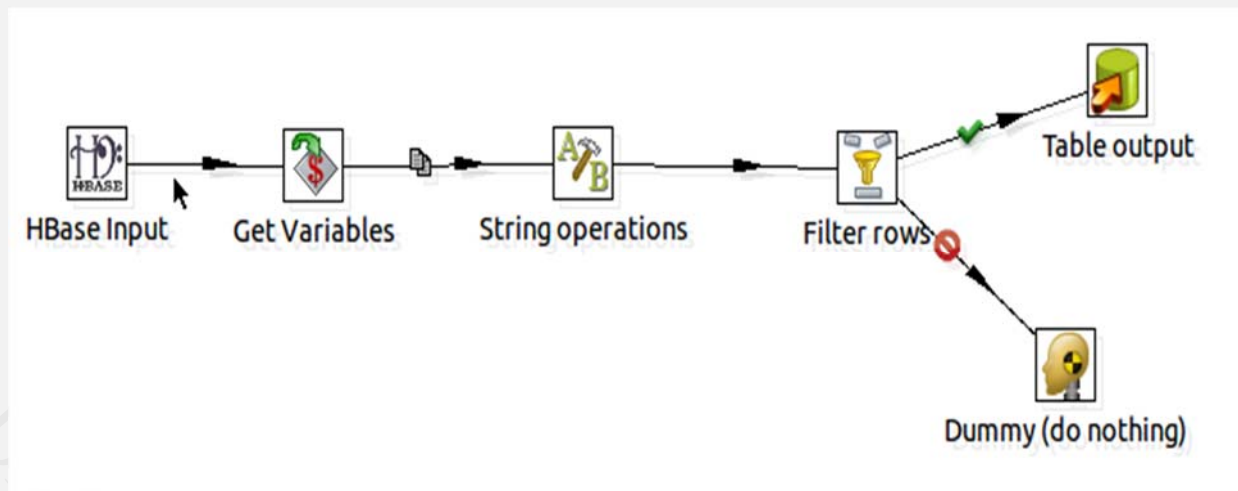
## Phase 3: Get last triple field

- PDI steps: Table input, Modified Java Script value, Set variables
- This transformation is invoked to get the last triple field from MySQL and to extract from current date the variables to build the path in which the triples file generated will be stored. The path must be `/Triples/Categoria/NomeProcesso/Anno_mese/Giorno/Ora/MinutiSecondi/file.xxx`.



# Phase 4: Data from Hbase to MySQL

- PDI step: HBase input, String operations , Get variables, Filter rows, Table Output
- This transformation is invoked to extract the improved data from HBase by filtering on the basis on process name and last triple field (retrieved previously), and to store it in MySQL table created previously.



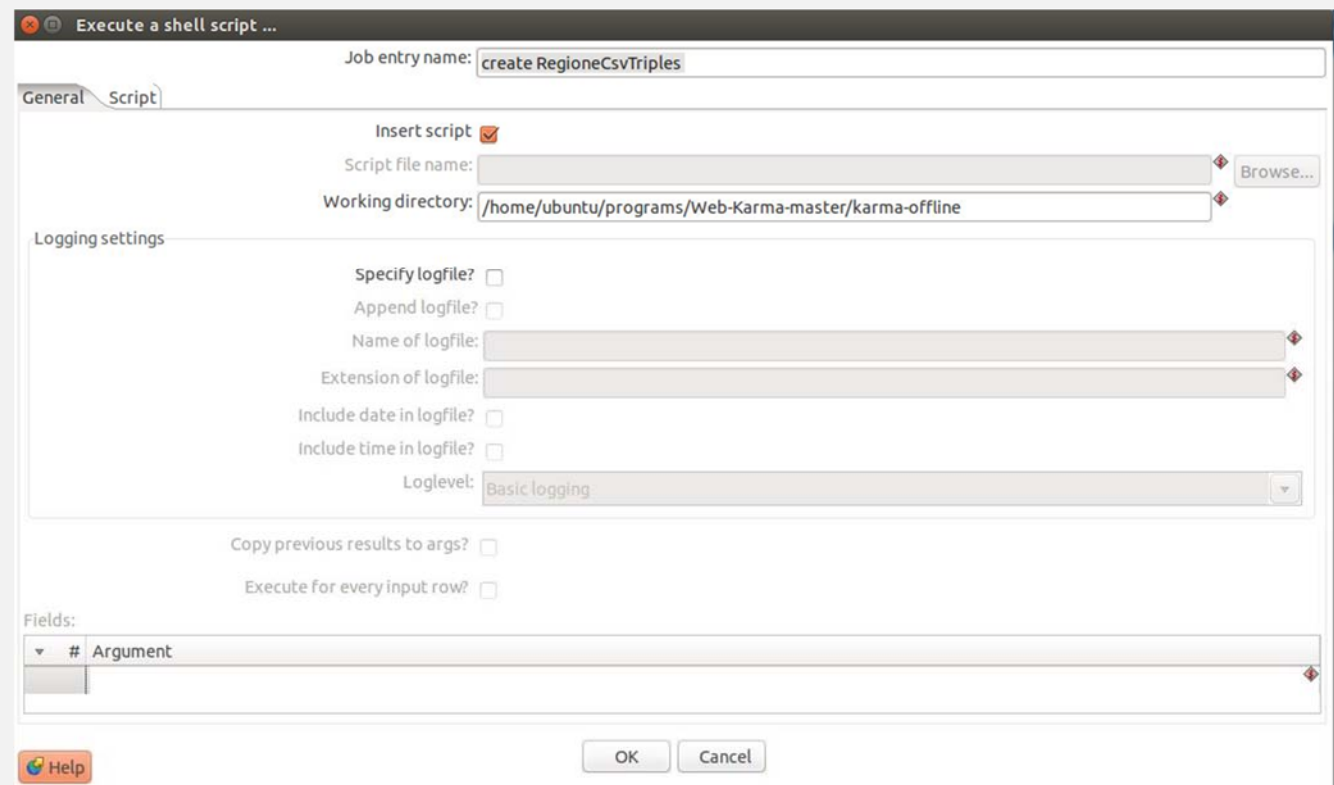
# Phase 5: Creating storage folder

- **PDI step: Create folder**
- Create the folders that will host the n3 triples file created in the next steps.



# Phase 6: RDF Triples generation

- PDI step: Execute a shell script Step

Execute a shell script ...

Job entry name: create RegioneCsvTriples

General Script

Insert script

Script file name:  Browse...

Working directory: /home/ubuntu/programs/Web-Karma-master/karma-offline

Logging settings

Specify logfile?

Append logfile?

Name of logfile:

Extension of logfile:

Include date in logfile?

Include time in logfile?

Loglevel: Basic logging

Copy previous results to args?

Execute for every input row?

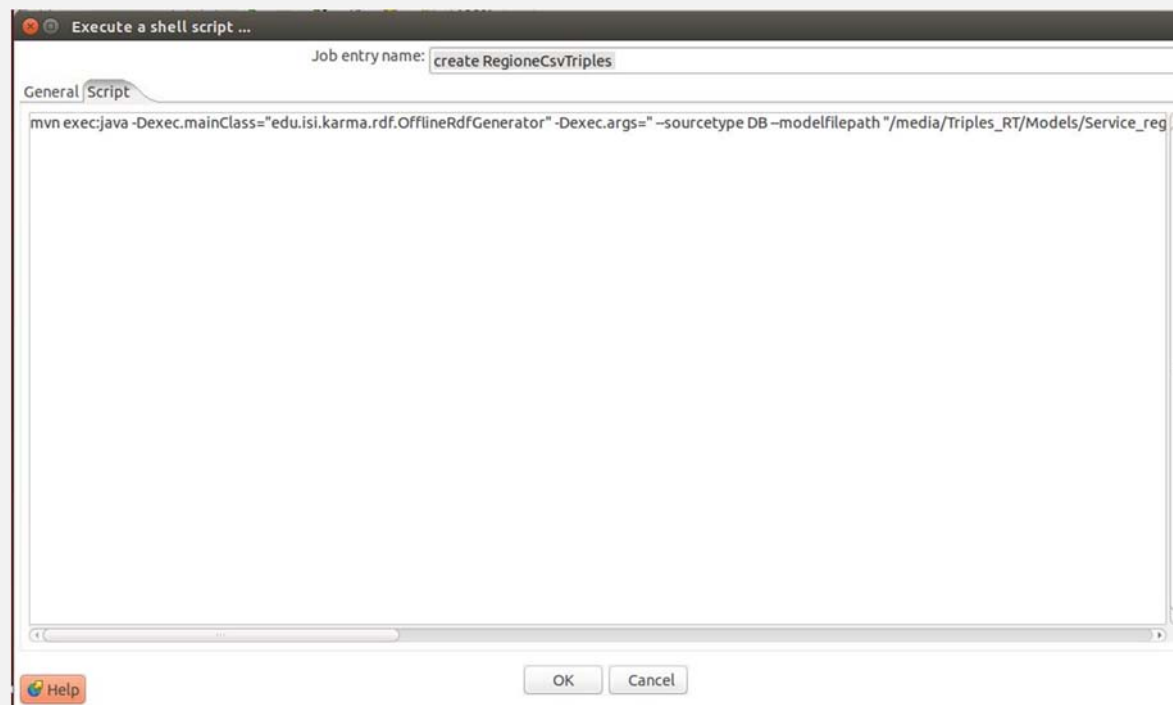
Fields:

#	Argument

Help OK Cancel

1. check the option "Insert Script" to execute the script in the Script tab
2. specify the working directory for the command or script

# Phase 6: RDF Triples generation



3. insert the specific command to create the RDF triples in the Script tab;

# Phase 6: RDF Triples generation

RDF triples command:

```
mvn exec:java -Dexec.mainClass="edu.isi.karma.rdf.OfflineRdfGenerator" -  
Dexec.args=" --sourcetype DB --modelfilepath  
"/media/Triples_RT/Models/Service_region.ttl" --outputfile  
${DestinationDir}/${processName}.n3 --dbtype MySQL --hostname  
192.168.0.01 --username x --password x --portnumber 3306 --dbname Mob --  
tablename ${processName}" -Dexec.classpathScope=compile
```

- In input you specify the mapping model, the Mysql table (where you get source data) and the connection parameters to database.
- In output you specify the file name (.n3) where the triples RDF will be stored.



# Reference

**LINK PAGINA WEB**

**<http://www.disit.org/drupal/?q=node/6690>**



# Smart City: data ingestion and mining

Parte 12 (2015-2016) of  
Course on KNOWLEDGE MANAGEMENT AND PROTECTION  
SYSTEMS

*Giacomo Martelli, Mariano di Claudio*

Dipartimento di Ingegneria dell'Informazione, DINFO

Università degli Studi di Firenze

Via S. Marta 3, 50139, Firenze, Italy

Tel: +39-055-2758515, fax: +39-055-2758570

**DISIT Lab**

<http://www.disit.dinfo.unifi.it> *alias* <http://www.disit.org>

[giacomo.martelli@unifi.it](mailto:giacomo.martelli@unifi.it)

Prof. Paolo Nesi, [paolo.nesi@unifi.it](mailto:paolo.nesi@unifi.it)