

# Smart City: data ingestion and mining

Corso del DISIT lab

**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>

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
- ☐ Sii-Mobility Project
- ☐ ETL processes implementation (within the Sii-Mobility Project)

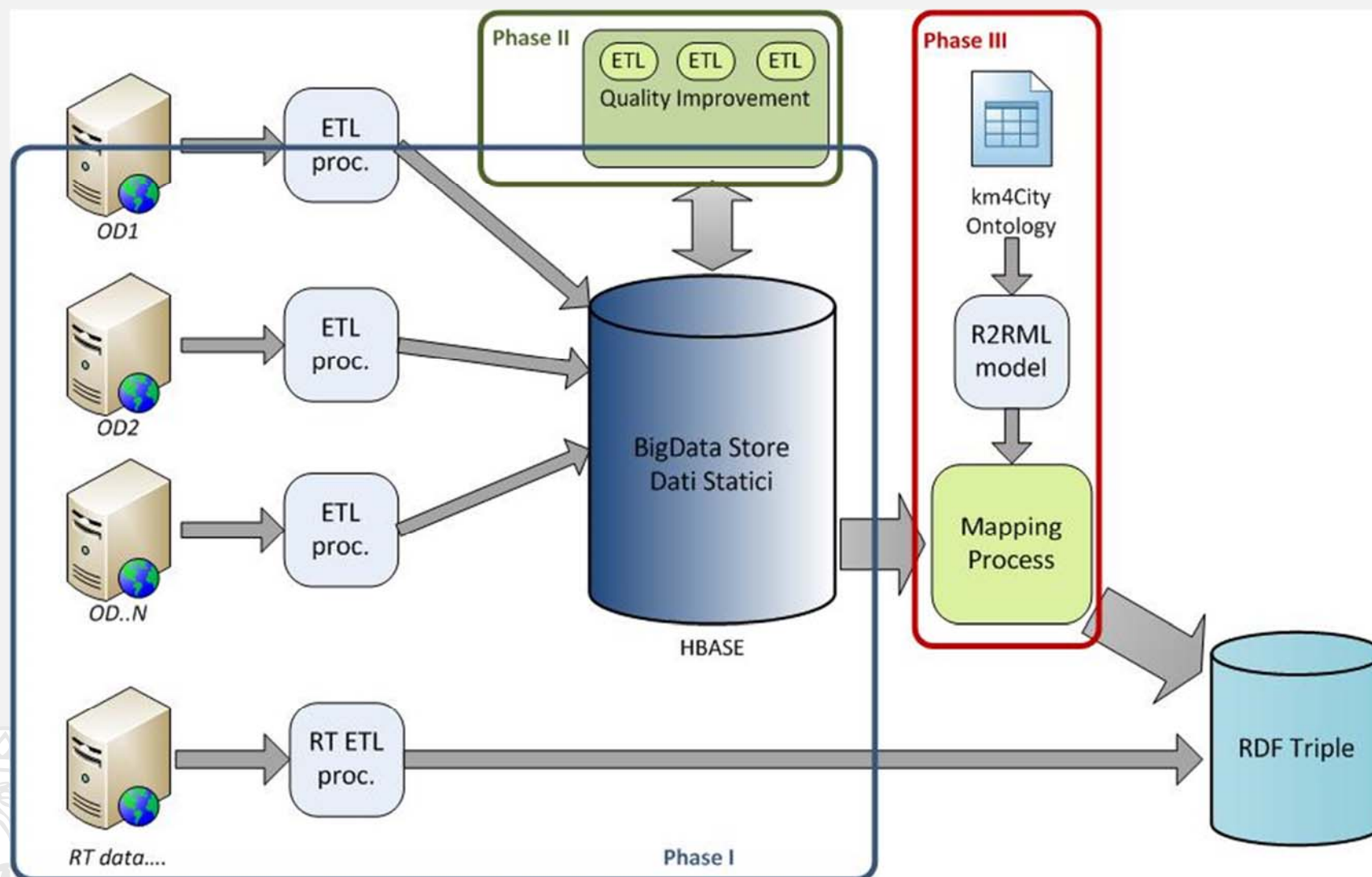


1

# Big Data: from Open Data to Triples



# Data Engineering Architecture



# Phase I: Data Ingestion

- **Acquisition of 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, JSON, 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** data to fit operational needs which may include improvements of quality levels (**Data Quality Improvement** phase).
- **Loading** data 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**.

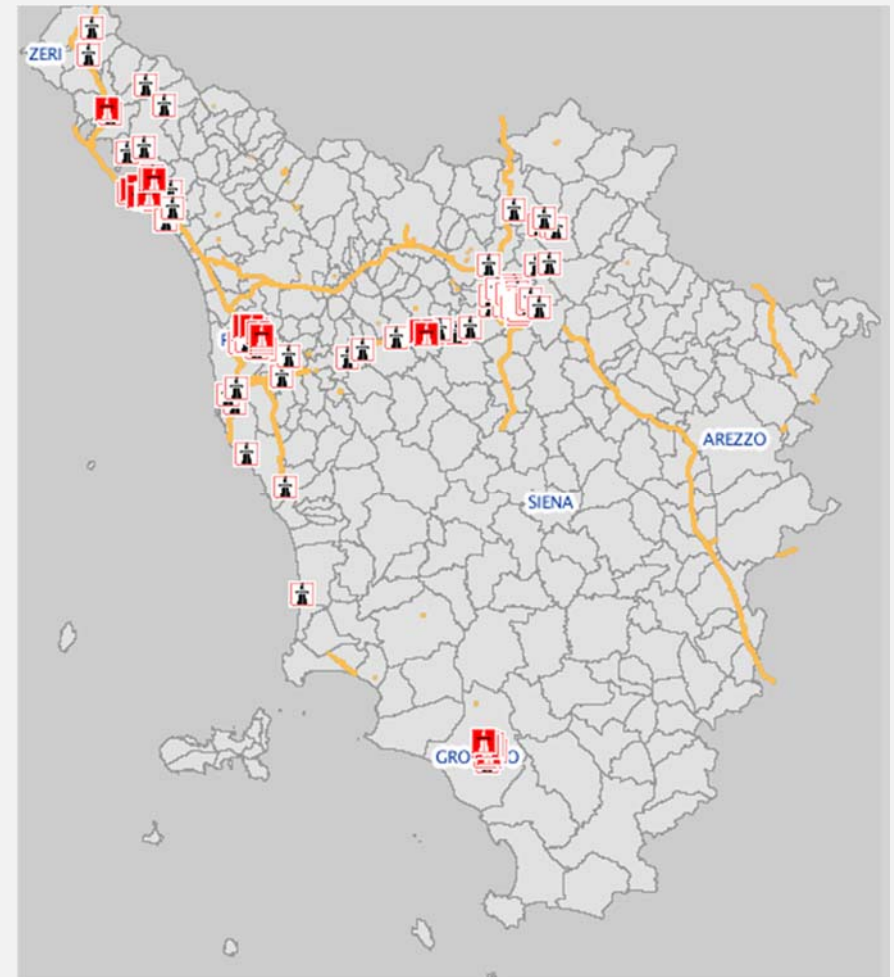
# Dataset

- **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.



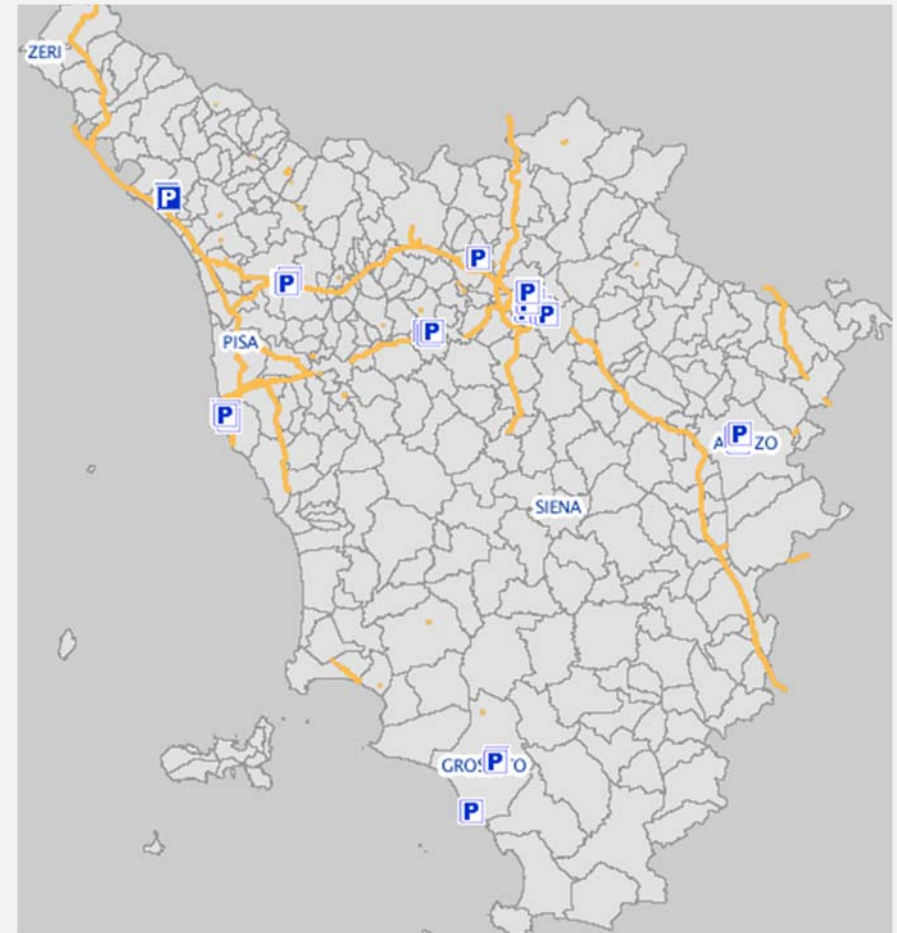
# Dataset: MIIC

- **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 site 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.
  - Scheduled times of local public transport (bus, tram, train, ferry).

# 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



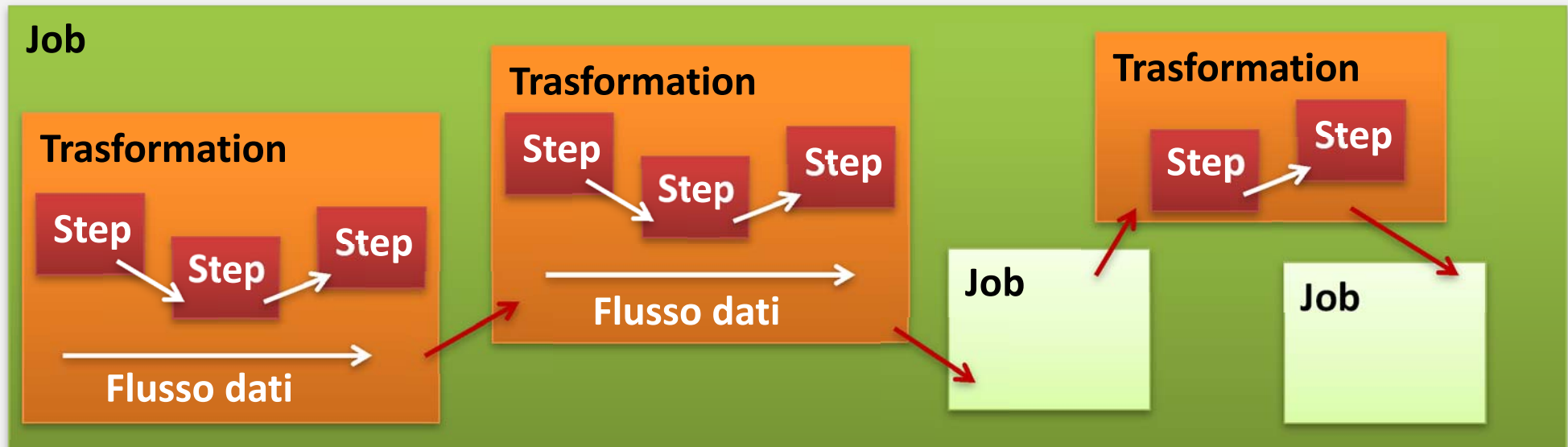
# Kettle: Concepts

- 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 executed in parallel on separated threads and there is no necessarily a beginning or end point of the 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 can be 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**.

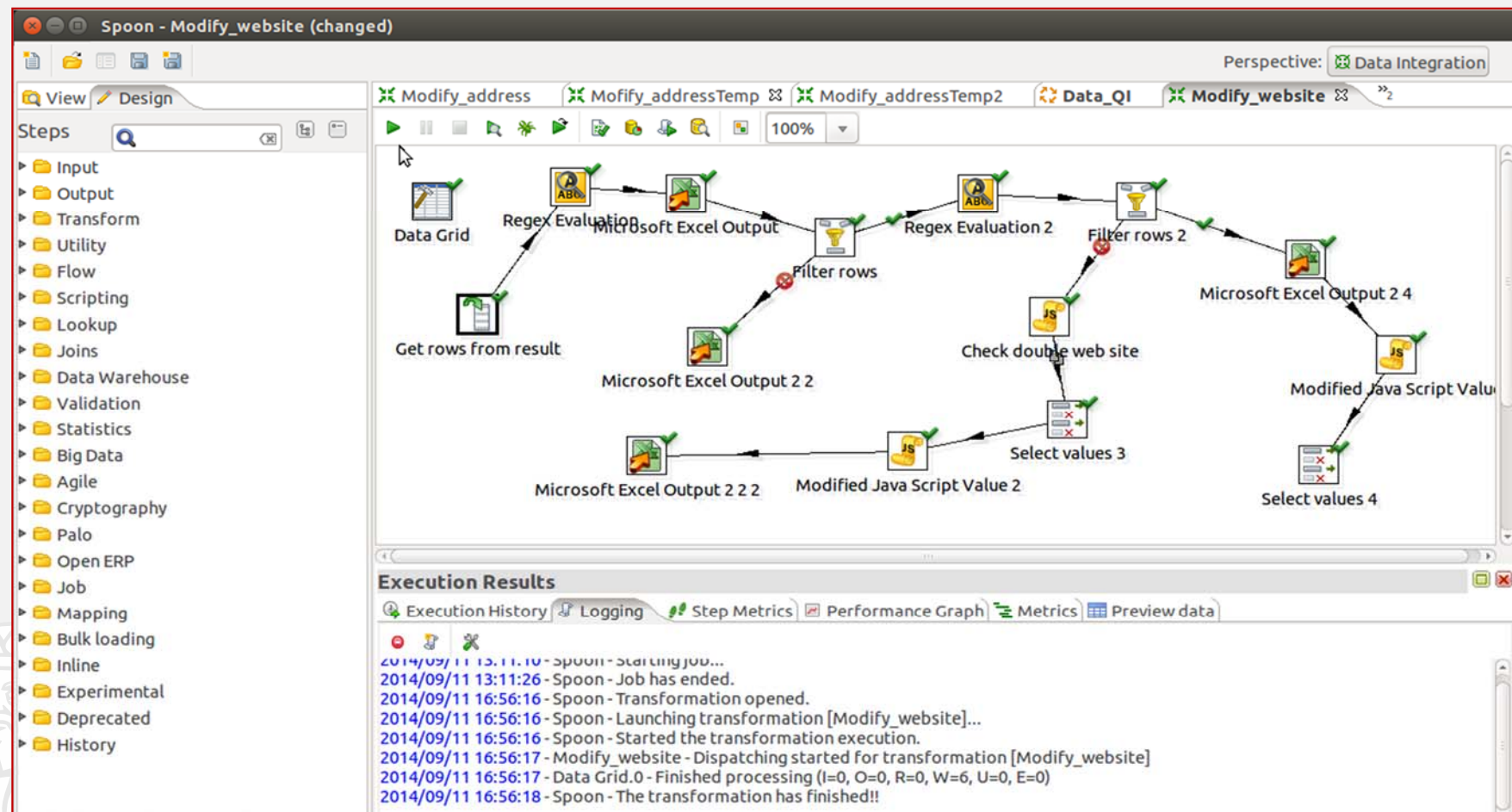
- **Input steps** process some kind of 'raw' resource (file, database query or system variables) and create an output stream of records from it.
- **Output steps** (the reverse of input steps): accept records, and store them in some external resource (file, database table, etc.).

# Type of Steps in Spoon (2/2)

- **Transforming steps** process input streams and perform particular actions on them (adding new fields/new records); these actions 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

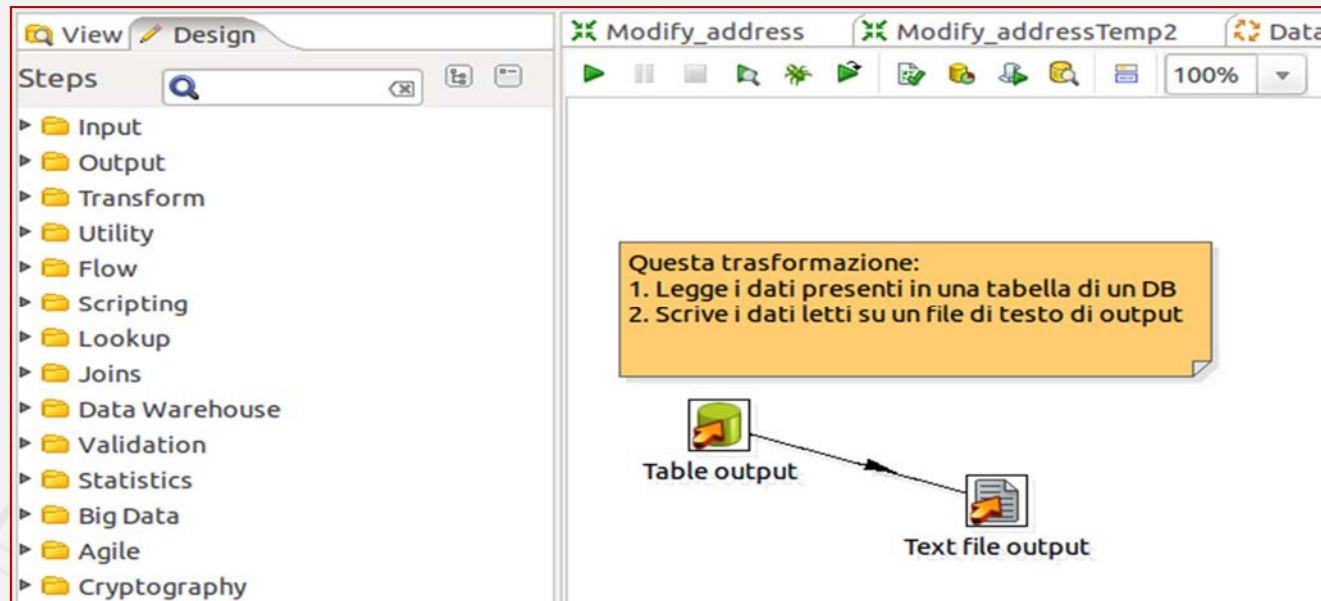
To run Spoon, just launch the command `./spoon.sh` from command line.





# Kettle: Transformations

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

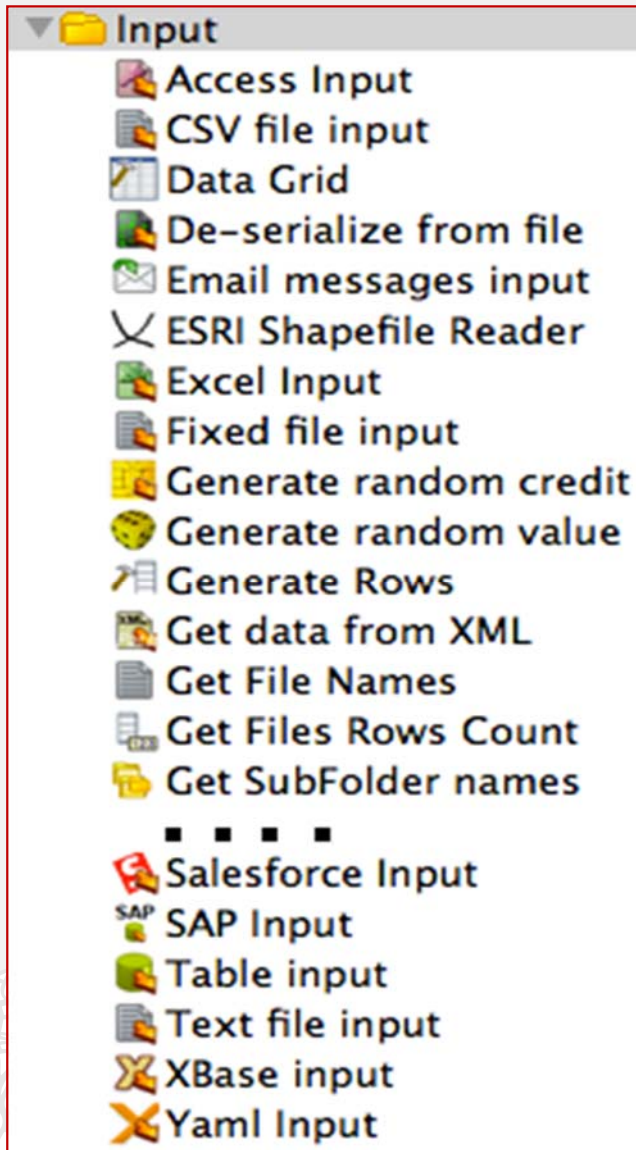


# Kettle: Transformations

- There are several possible steps organized by category: ***Input, Output, Utility, Scripting, Flow, Validation, Lookup, Statistics, etc...***
- Each category of step, in turn, offers several of possibilities.
- For example, the input category makes available different steps in order to get data from different sources: ***table of NoSQL/SQL database, CSV file, MS-Excel sheet, etc...***
- The hops between two steps don't define the execution sequence but represent the data flow and allow to pass the content of a Field from one step to the next one.



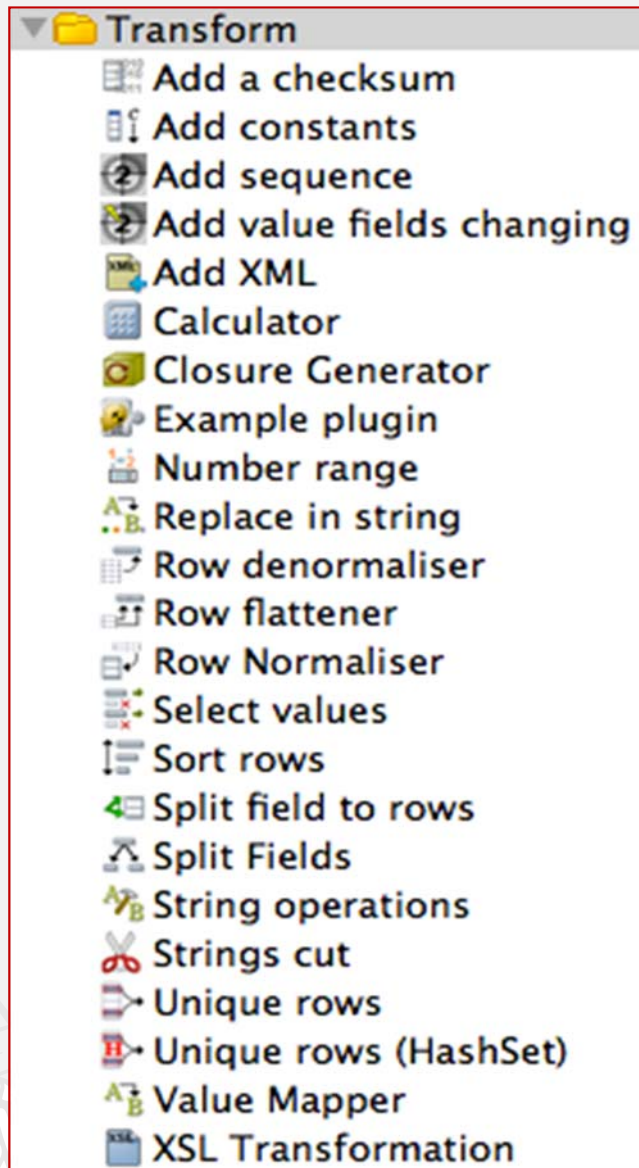
# Kettle: Transformations



## Input

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

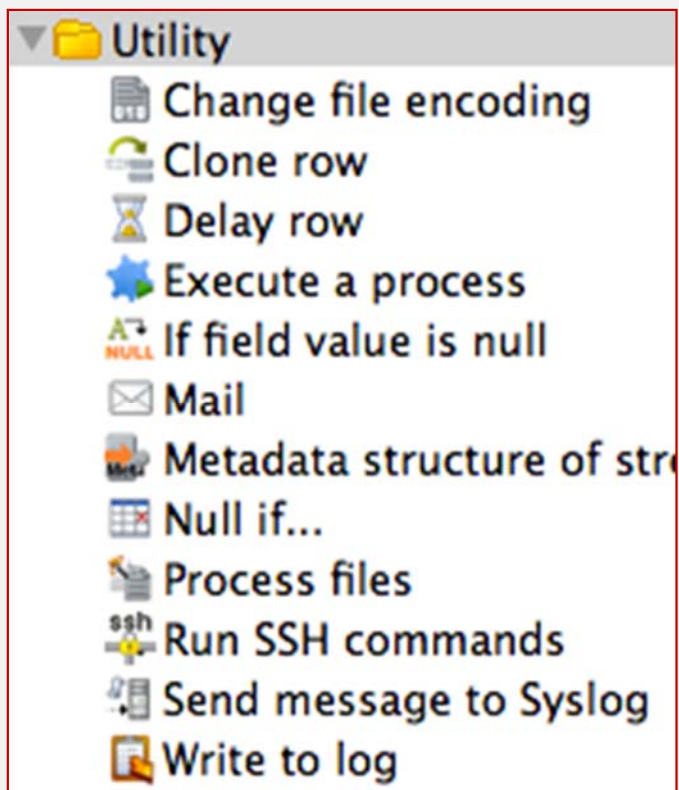
# Kettle: Transformations



## Transform

Collection of steps dealing with data operations: i.e. trim, fields separation, strings truncation, rows sorting, etc.....

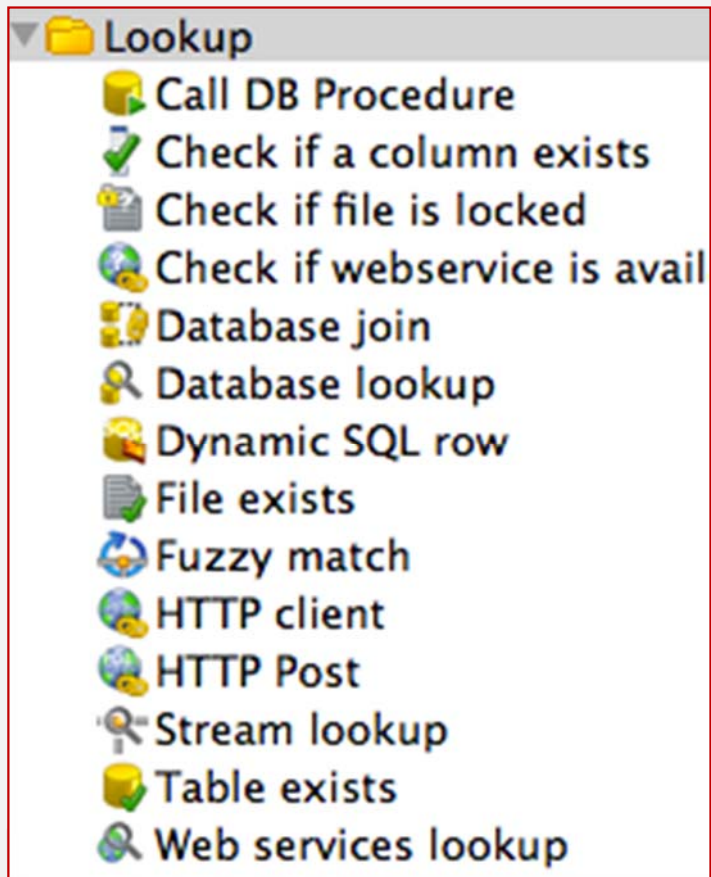
# Kettle: Transformations



## Utility

Collection of support steps with advanced features: i.e. log writing, check if a field is null, rows deletion, etc....

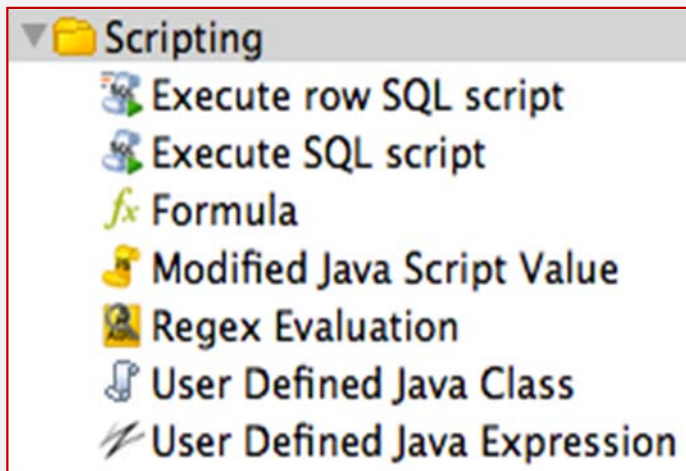
# Kettle: Transformations



## Lookup

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

# Kettle: Transformations

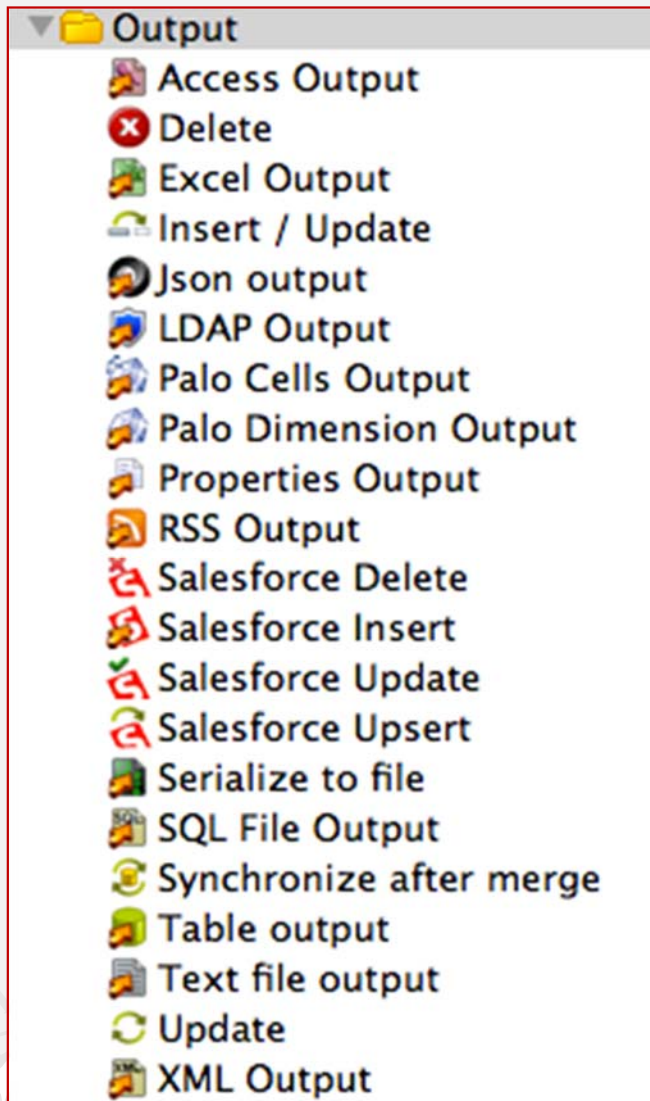


## Scripting

Set of steps used to define scripts in different languages (SQL, JavaScript, etc...).



# Kettle: Transformations



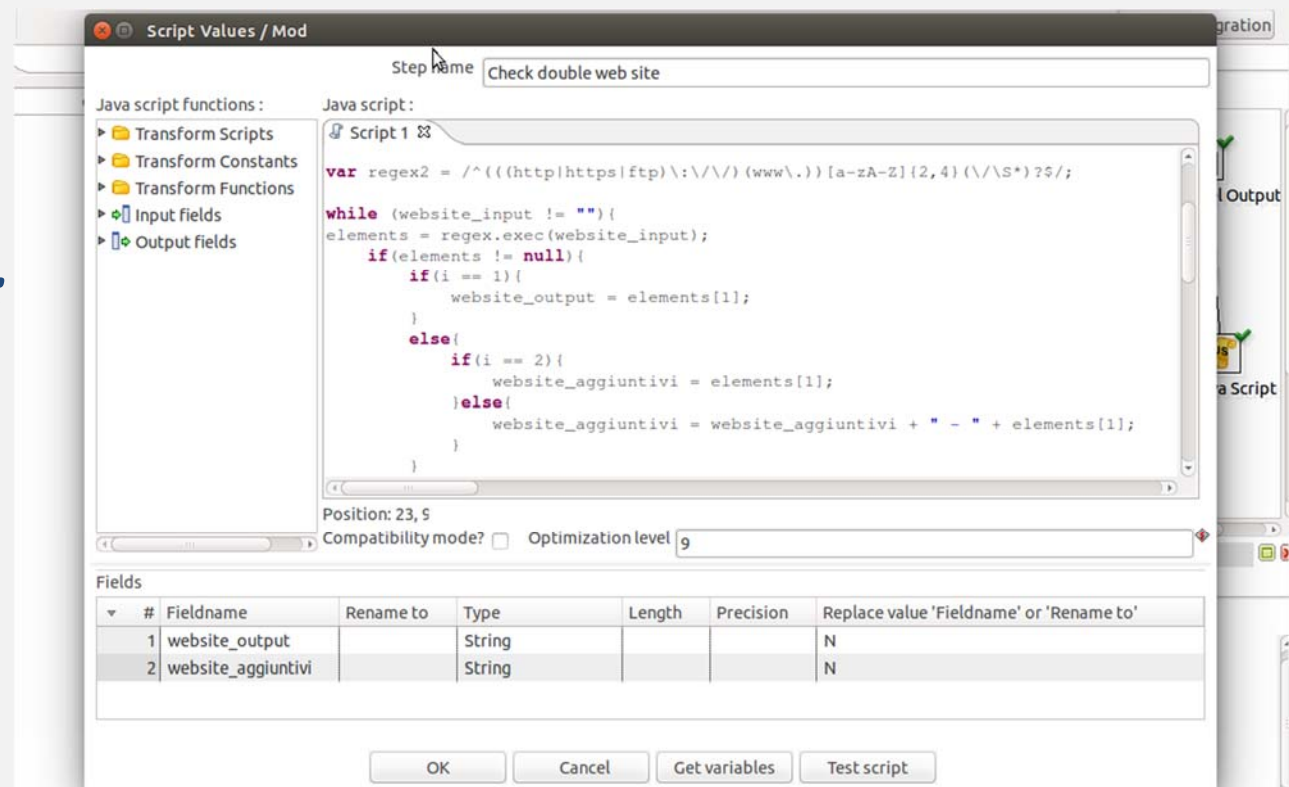
## Output

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

# Kettle: Transformations

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

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





# 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/Aggiorna1.kjb >> /tmp/cron1.log 2>&1
```

- The output is typically recorded on a log which can be analyzed in case of errors.

```
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



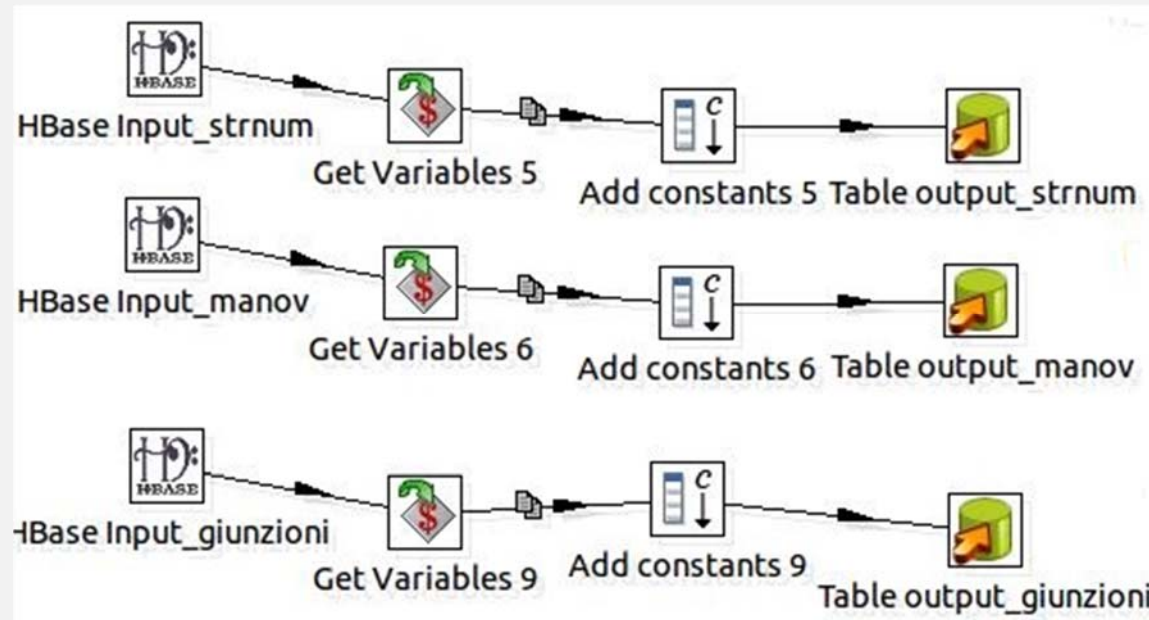
These steps (transformations) are executed sequentially (there is a single flow execution).

```

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.

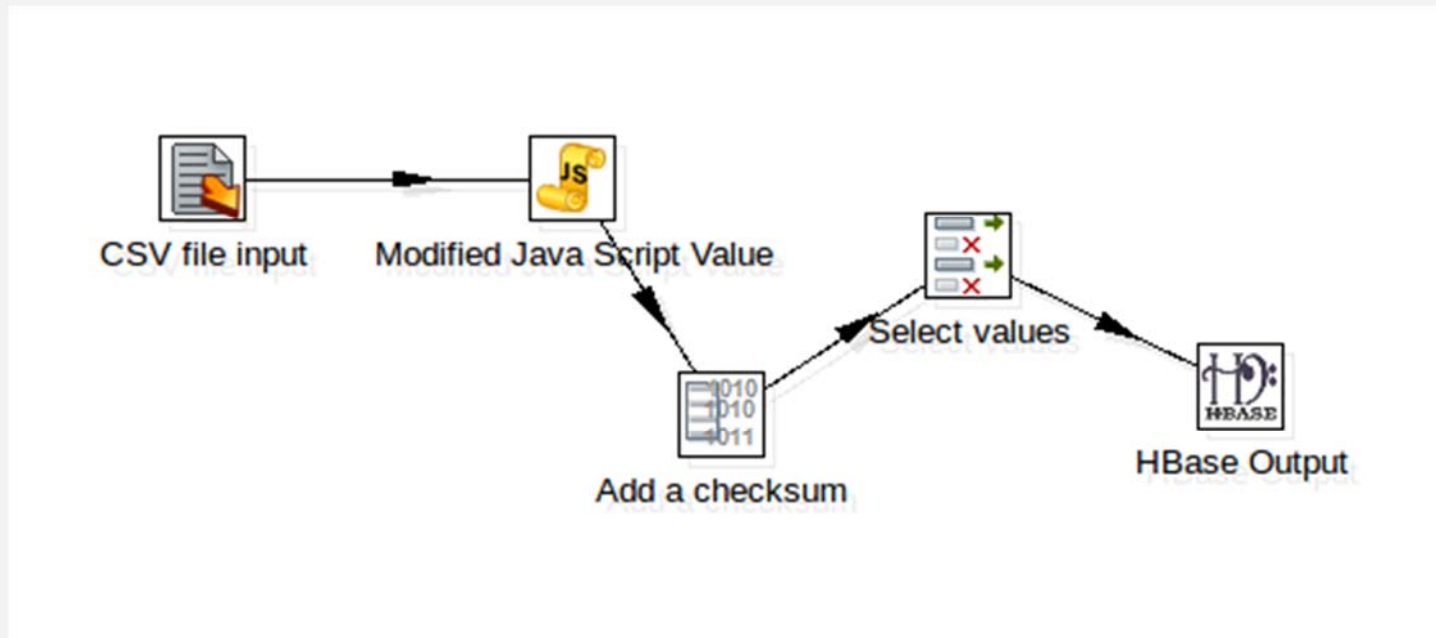
3

# ETL tool: Pentaho Data Integration (PDI)

## EXAMPLES

# Example 1 - Transformation

- This transformation acquires the museums dataset (file in CSV format) and defines a key to load data into a specific HBase table (“monuments”).



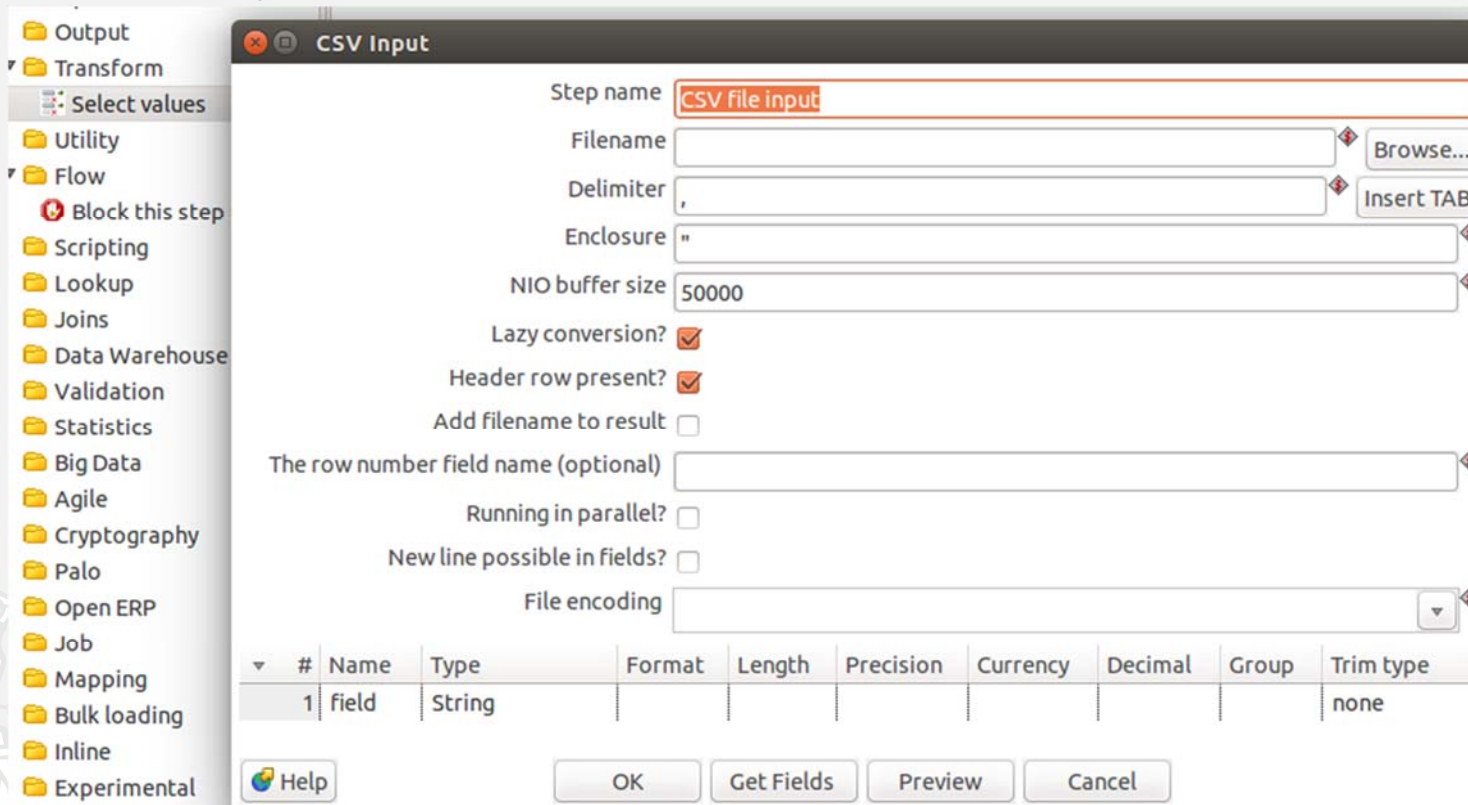
- There are five steps and the sequence is given by data flow.



# Hbase Output Transformation

## CSV file input

- In this step you select the CSV file, 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: **CSV file input**

Filename:  **Browse...**

Delimiter:  **Insert TAB**

Enclosure:

NIO buffer size:  50000

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:

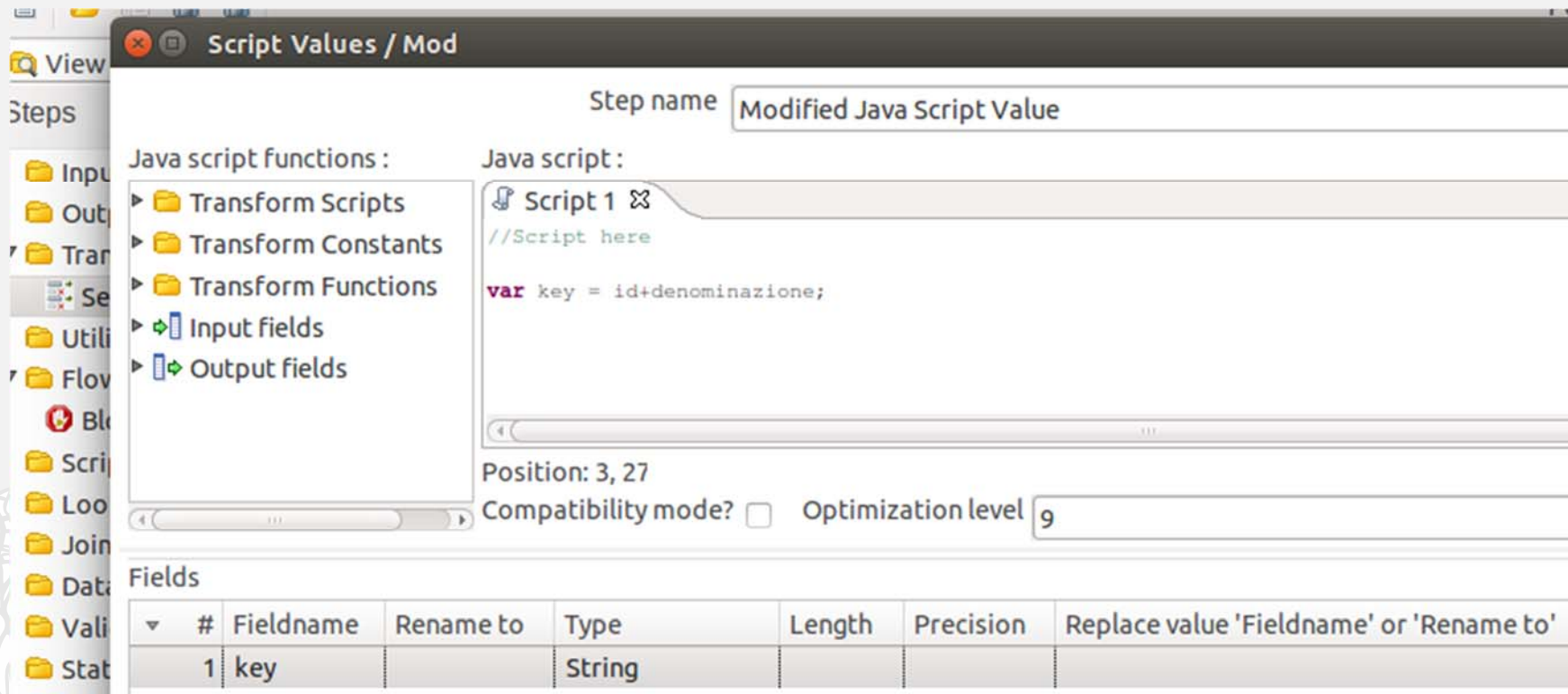
#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	field	String							none

**Help** **OK** **Get Fields** **Preview** **Cancel**

# Hbase Output Transformation

## Modified Java Script value

- In this step you can add JavaScript code. A variable is defined by concatenating two input fields and at the end the same variable is used to define an output field. The goal is define the key field to load data later.

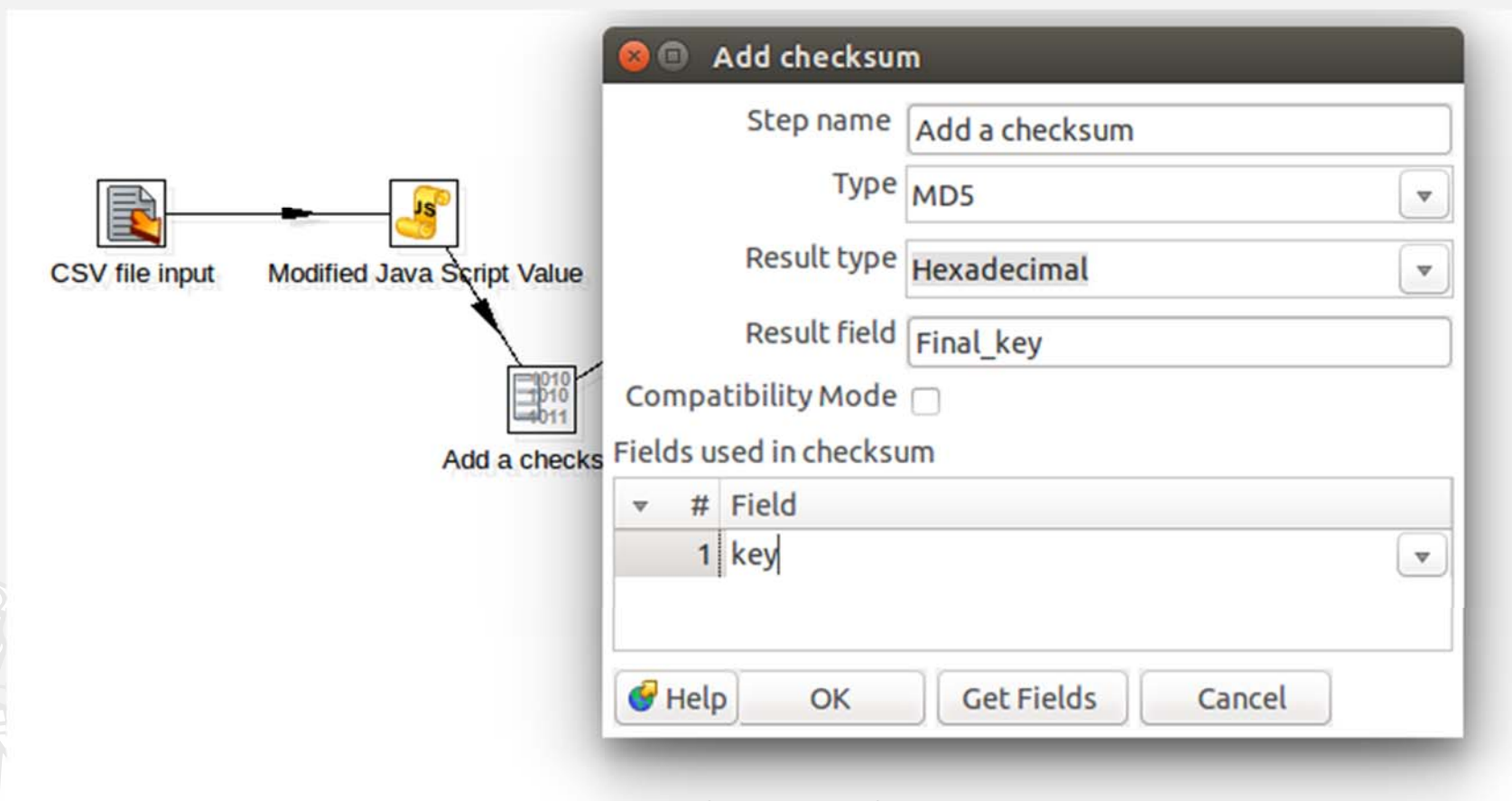




# Hbase Output Transformation

## Add a Checksum

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



The diagram illustrates a data transformation workflow. It starts with a 'CSV file input' icon, followed by a 'Modified Java Script Value' icon, and finally an 'Add a checksum' icon. A callout window titled 'Add checksum' is shown, detailing the configuration for this step.

**Add checksum configuration:**

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

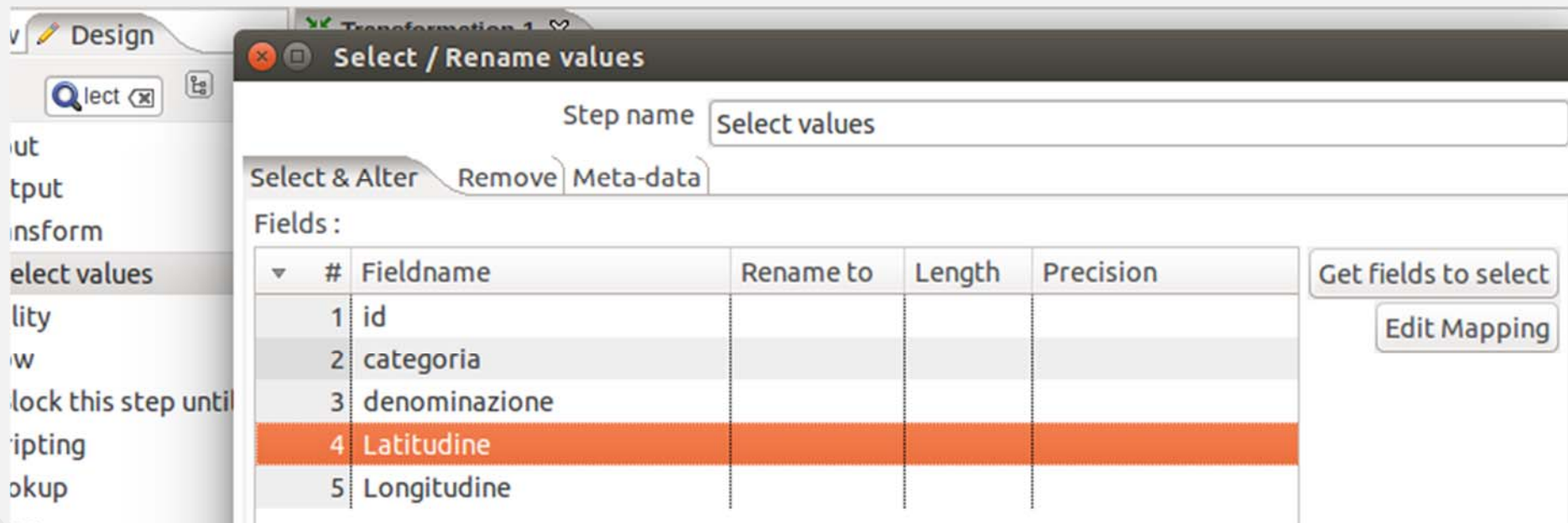
#	Field
1	key

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

# Hbase Output Transformation

## Select Values

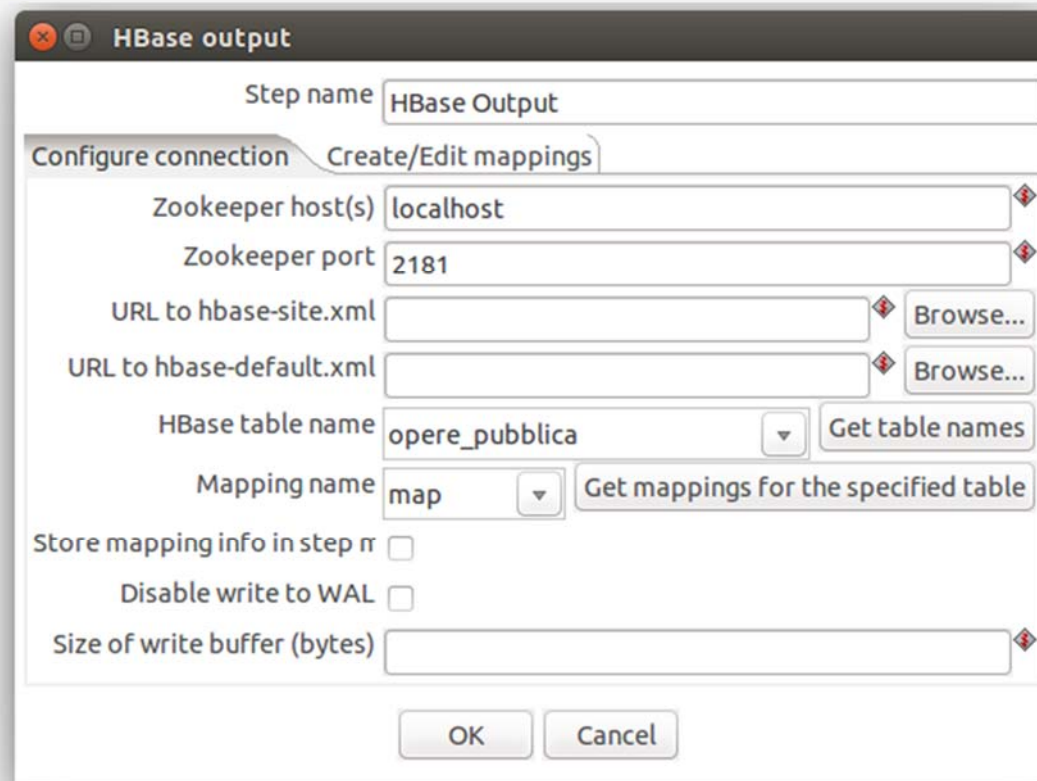
- In this step you can select the fields (one or more) that you want to pass to next step. Furthermore, you can also use the Remove option to select the fields that you want to block.



# Hbase Output Transformation

## Hbase Output

- In this step you set the parameters in order to load data into a HBase table.
- 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:

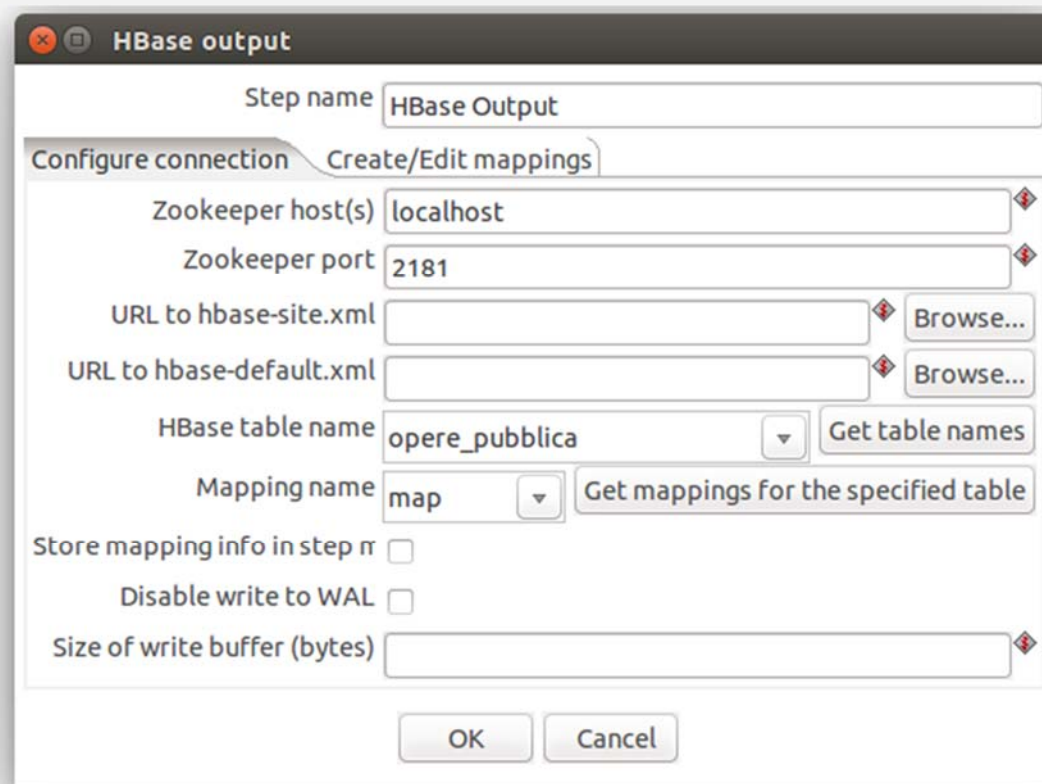
- Step name: HBase Output
- Zookeeper host(s): localhost
- Zookeeper port: 2181
- URL to hbase-site.xml: [empty] Browse...
- URL to hbase-default.xml: [empty] Browse...
- HBase table name: opere\_pubblica (dropdown) Get table names
- Mapping name: map (dropdown) Get mappings for the specified table
-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

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

# Hbase Output Transformation

## Hbase Output

- Subsequently, you select the specific HBase table, the relative mapping and the fields to be loaded. The mapping must be defined previously in the second tab.



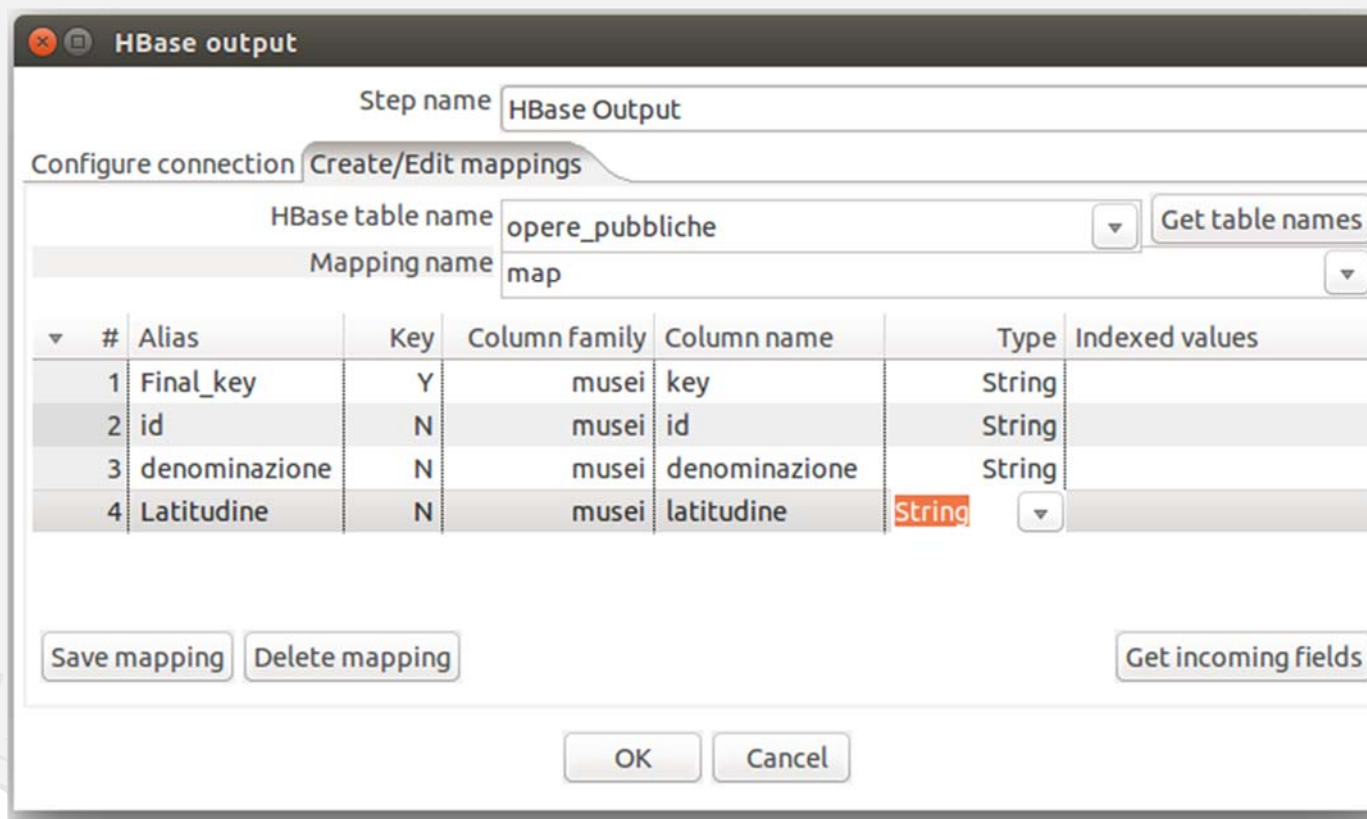
The screenshot shows a dialog box titled "HBase output" with two tabs: "Configure connection" and "Create/Edit mappings". The "Create/Edit mappings" tab is active. The dialog contains the following fields and controls:

- 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 (dropdown) Get table names
- Mapping name: map (dropdown) Get mappings for the specified table
- Store mapping info in step n: ☐
- Disable write to WAL: ☐
- Size of write buffer (bytes): [empty]
- OK and Cancel buttons at the bottom.

# Hbase Output Transformation

## Hbase Output

- In the second tab you can define or edit the mapping of a specific HBase table. A mapping simply defines metadata about the values that are stored in the table.



The screenshot shows a window titled "HBase output" with a "Step name" field set to "HBase Output". Below this, there are two tabs: "Configure connection" and "Create/Edit mappings", with the latter being active. In the "Create/Edit mappings" tab, there is a form with the following fields:

- HBase table name:** A dropdown menu showing "opere\_pubbliche" and a "Get table names" button.
- Mapping name:** A dropdown menu showing "map".

Below the form is a table with the following columns: #, Alias, Key, Column family, Column name, Type, and Indexed values.

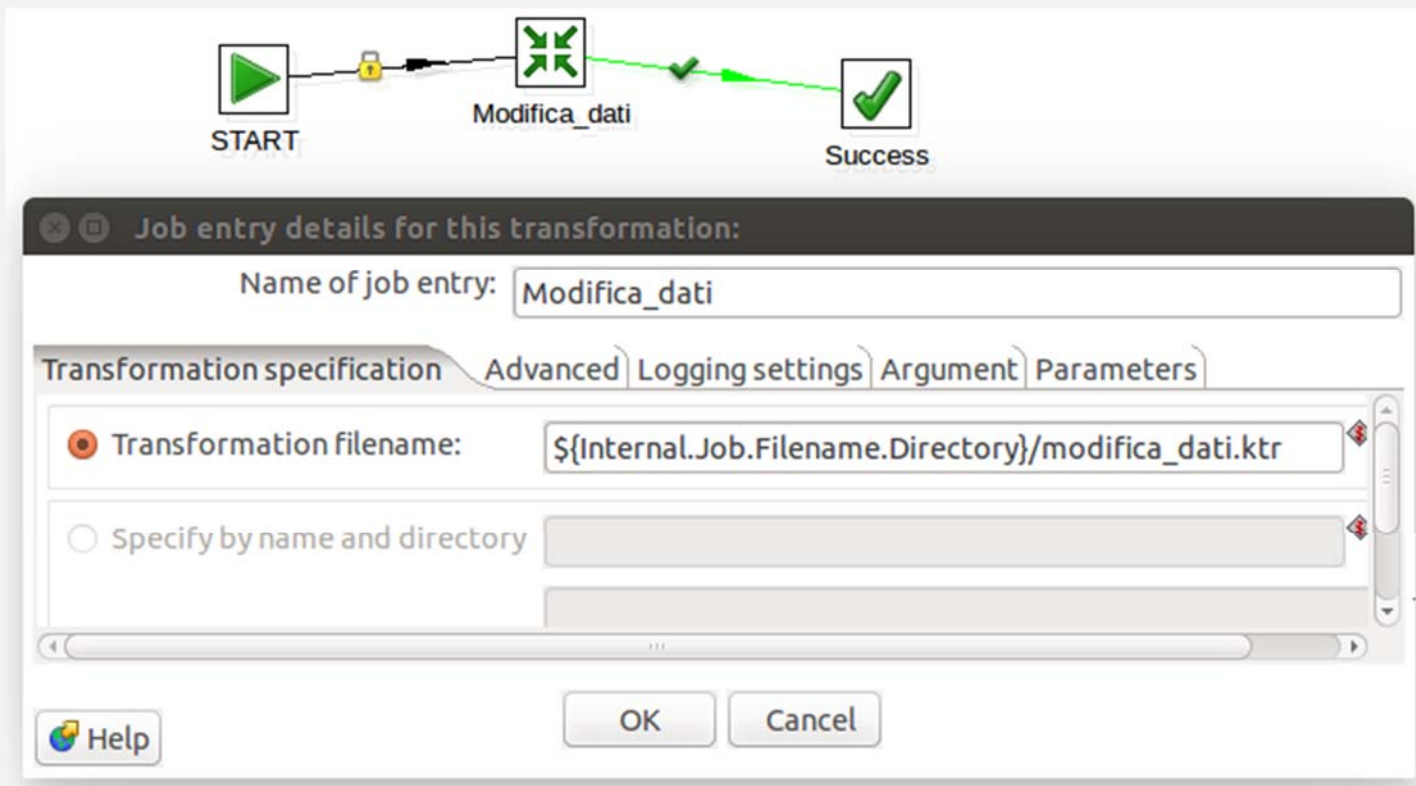
#	Alias	Key	Column family	Column name	Type	Indexed values
1	Final_key	Y	musei	key	String	
2	id	N	musei	id	String	
3	denominazione	N	musei	denominazione	String	
4	Latitudine	N	musei	latitudine	String	

At the bottom of the window, there are buttons for "Save mapping", "Delete mapping", "Get incoming fields", "OK", and "Cancel".



## Example 2 - Job

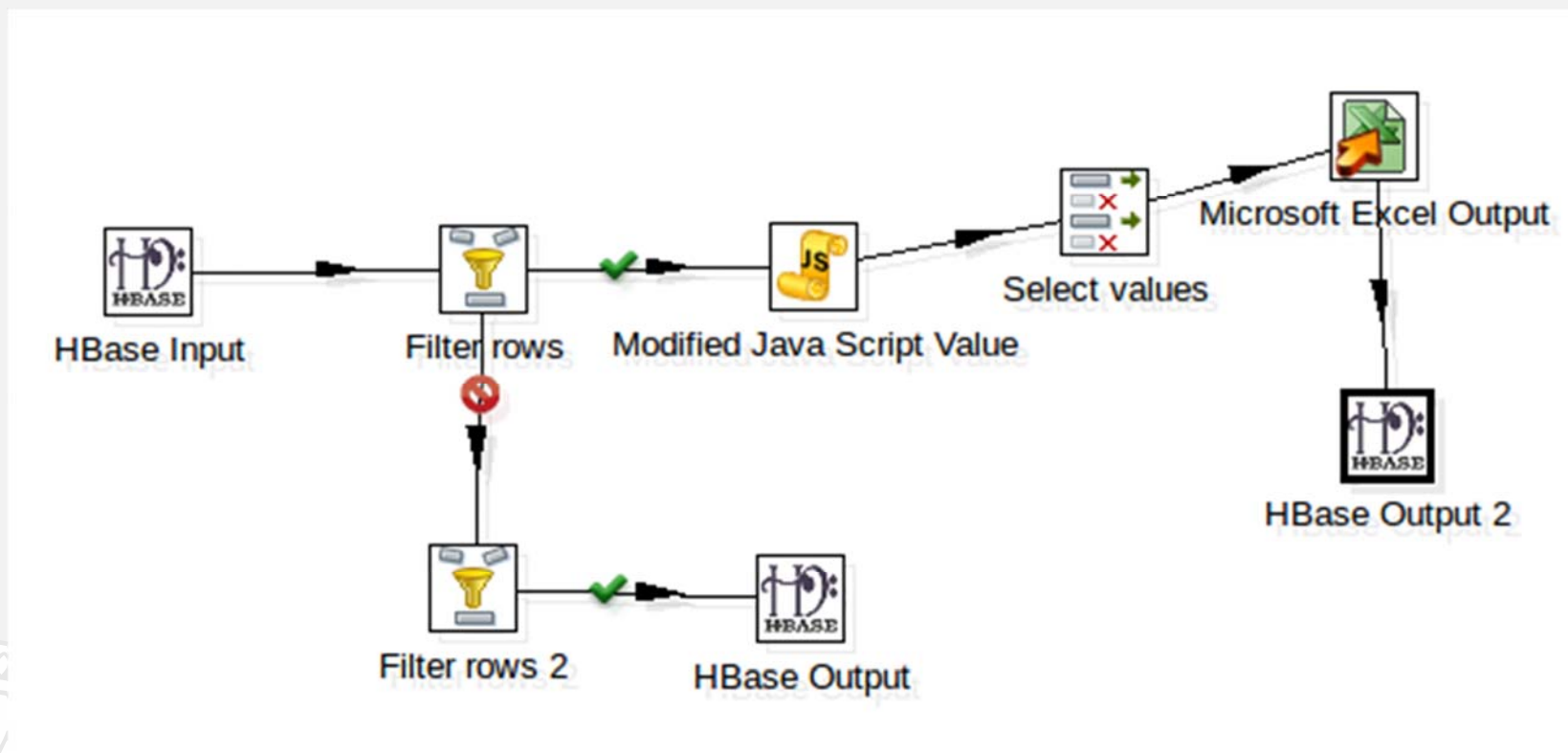
- This Job contains three steps: START (start step), Modifica\_dati (step that invokes a specific transformation) and Success (final step).



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

# Modifica\_dati Transformation

- This transformation loads data from HBase through the HBase Input Step. Then, the *Filter rows* and *Modified Java Script Value* steps are applied to perform some data cleaning before storing them again on HBase.



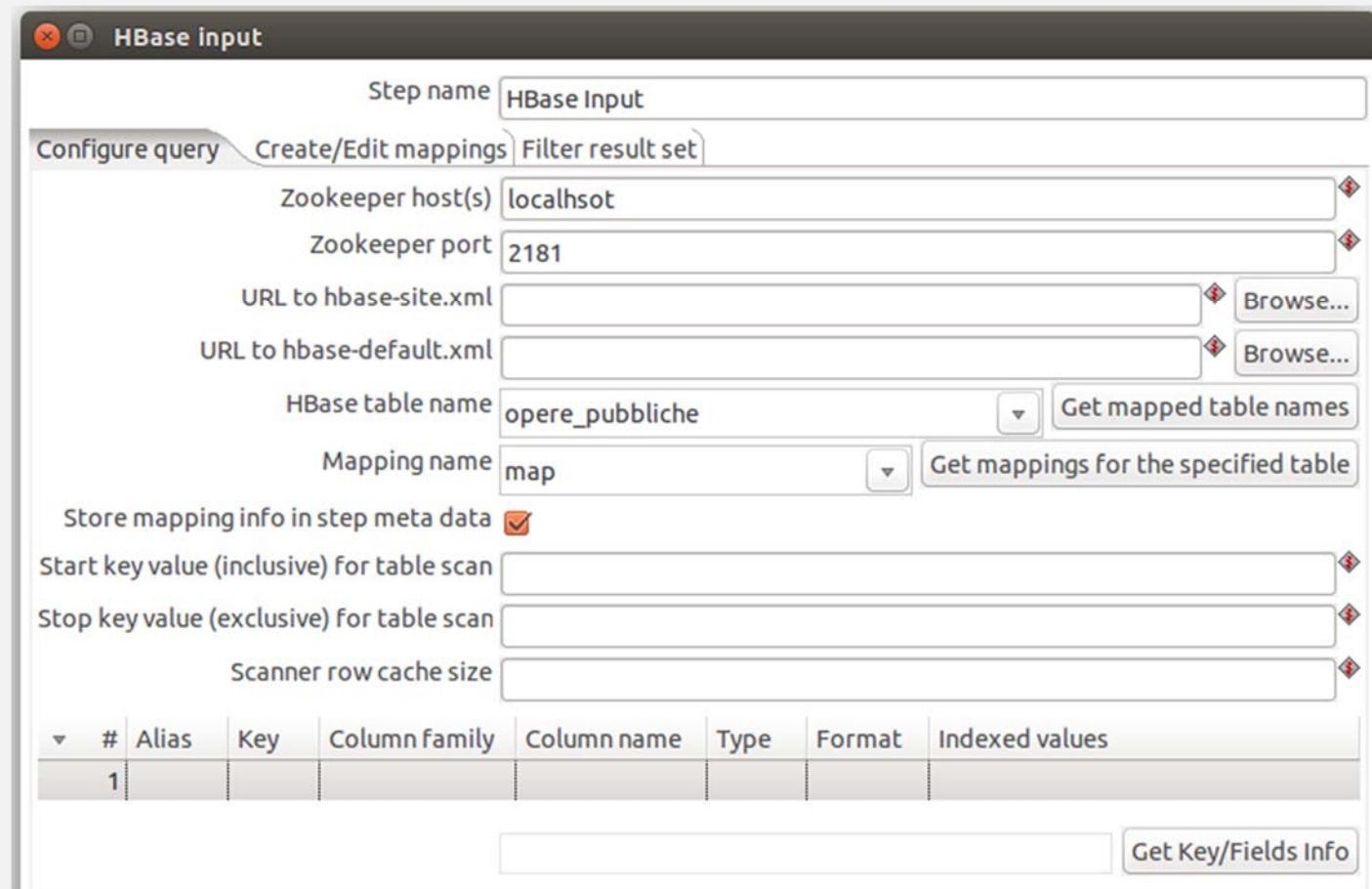
- In this case the data flow is splitted into two different smaller data streams based on a specific condition.



# Modifica\_dati Transformation

## Hbase Input

- In this step you set the parameters (port, IP address, HBase table and mapping) to retrieve the data from HBase.
- You can perform a filtering by setting some conditions in the *Filter result set* tab.



Step name: HBase Input

Configure query | Create/Edit mappings | Filter result set

Zookeeper host(s): localhsot

Zookeeper port: 2181

URL to hbase-site.xml: [Browse...]

URL to hbase-default.xml: [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: [ ]

Stop key value (exclusive) for table scan: [ ]

Scanner row cache size: [ ]

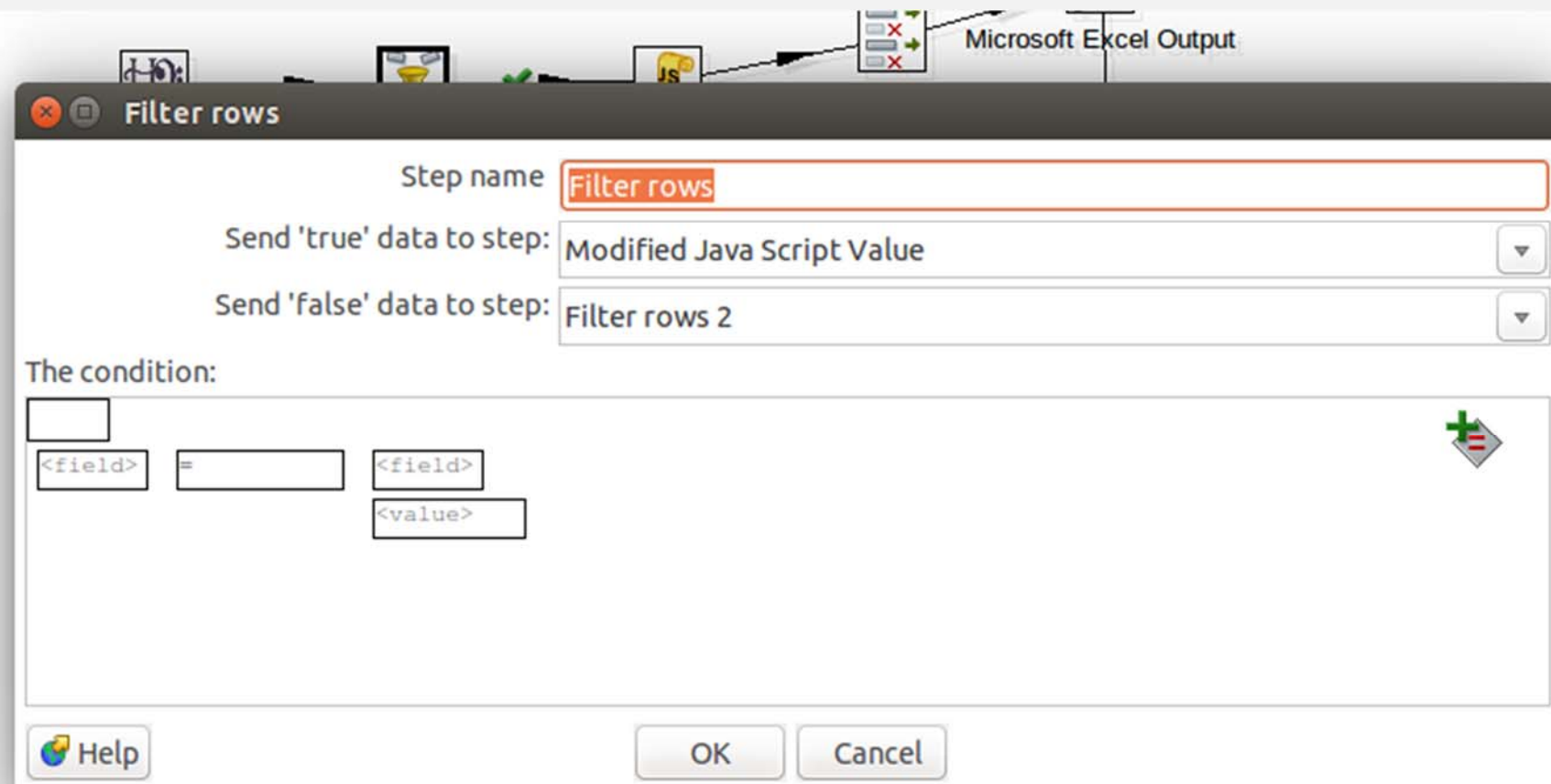
#	Alias	Key	Column family	Column name	Type	Format	Indexed values
1							

[Get Key/Fields Info]

# Modifica\_dati Transformation

## Filter rows

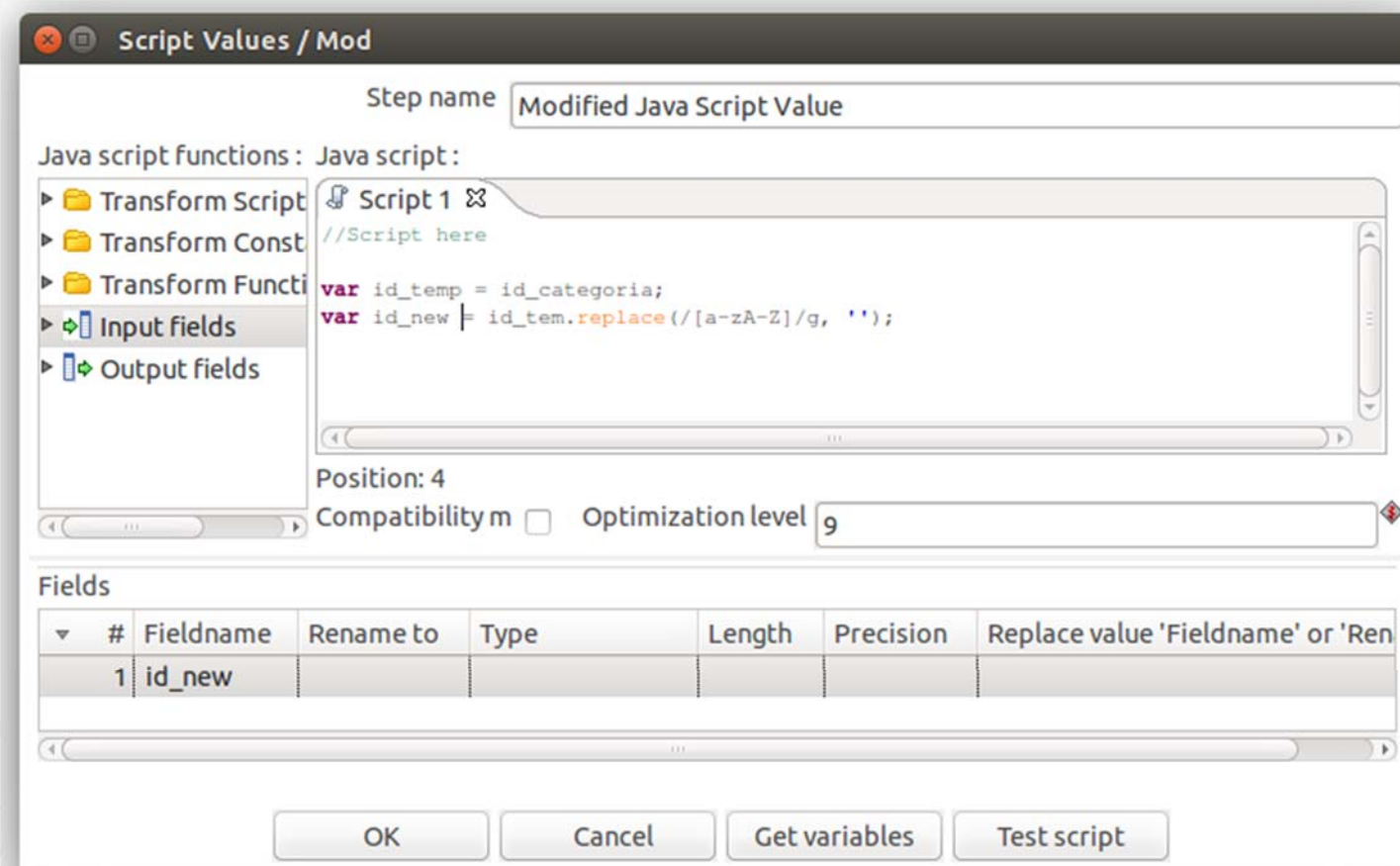
- It's another way to filter the data flow specifying a condition. In output this step creates a fork of the data flow.



# Modifica\_dati Transformation

## Modified Java Script Value

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



4

# Developing ETL processes with PDI



4

# Tool installation & configuration

Developing  
ETL  
processes  
with PDI

## ON LINUX UBUNTU 14.04

# Step 1: Downloading PDI

- Requisite: Oracle Java 7 JDK must be already installed on your system.
  - <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (link Oracle)
  - <https://help.ubuntu.com/community/Java> (link for 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 **5.0.1-stable** version and download the zip/tar.gz file.





## Step 2: Installation PDI

- To install PDI you must unpack the downloaded zip/tar.gz file into a specific folder.
  - the shell scripts must be executable if you use Linux operating systems. The relative command is:

```
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 (from a terminal window) the processes designed with Spoon.

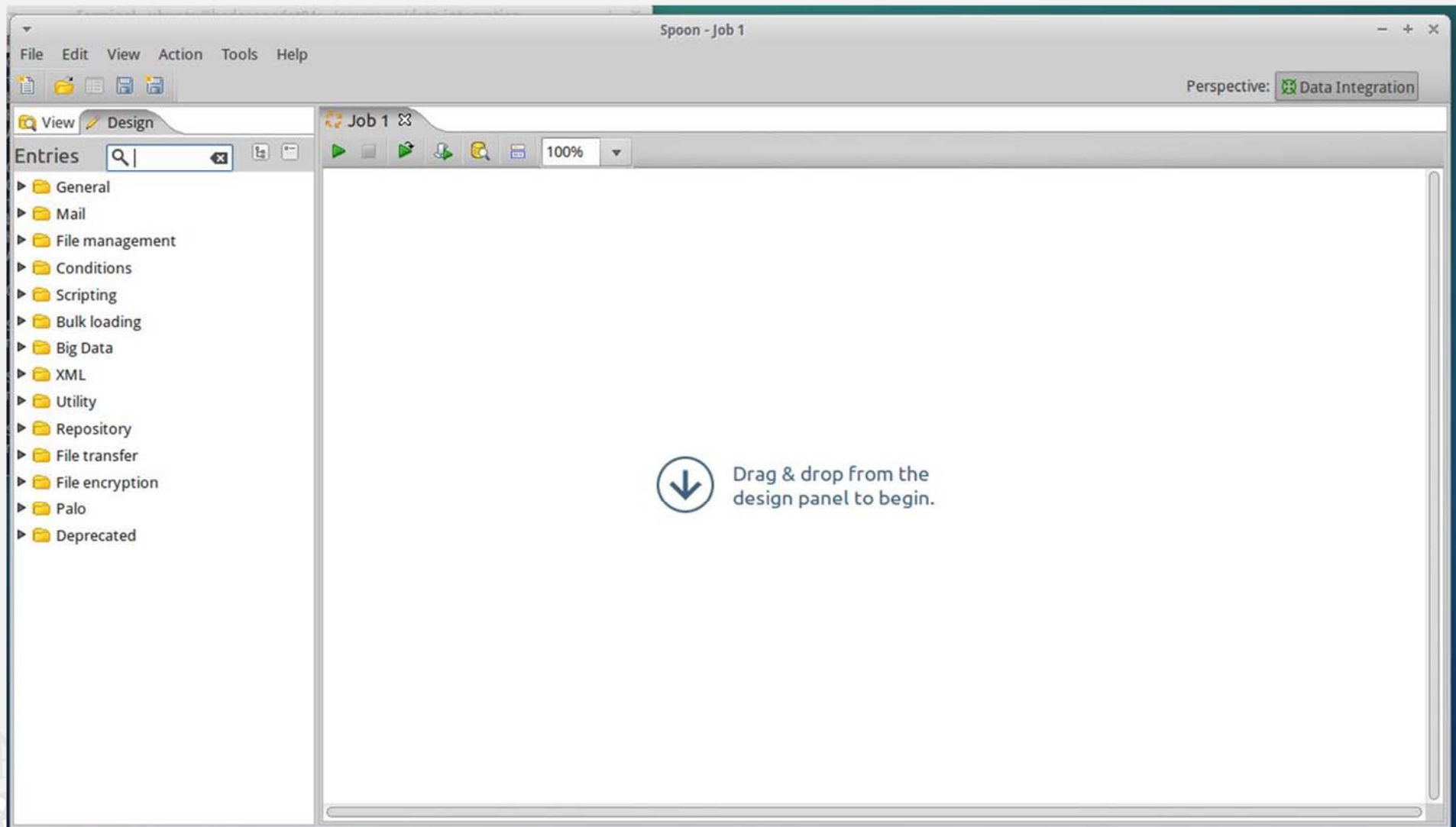


## Step 3: Running PDI

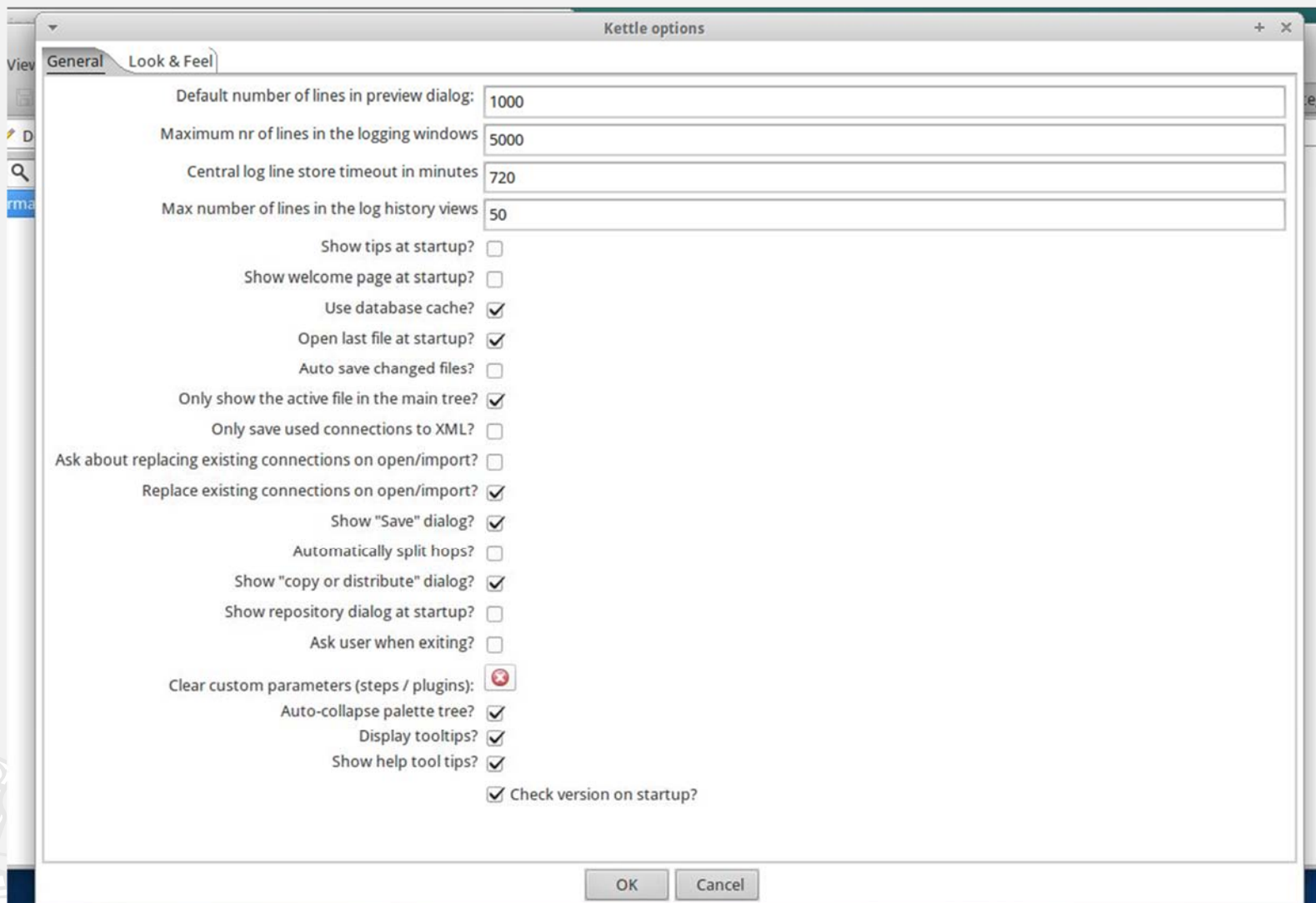
FILE/FOLDER	DESCRIPTION
<b>\PDI folder\data-integration</b>	Contains the Spoon designer and command line utilities
<b>\PDI folder\data-integration\spoon.sh</b>	Script file for starting the Spoon Designer on Linux and Macintosh
<b>\PDI folder\data-integration\Spoon.bat</b>	Script file for starting the 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 sources: 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 databases 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 an Apache web server, PHP and Perl.
  - There are versions of XAMPP for all platforms (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 (an instance of Hadoop Distributed File System (HDFS) is required).

## Step 6: Configure HBase

- Requisite: Oracle Java JDK must be already installed on your system (minimum version 1.6).
- Download the **0.90.5** version of HBase from this site (select the file that ends in *.tar.gz*):  
<http://archive.eu.apache.org/dist/hbase/hbase-0.90.5/>.
- To install HBase you must unzip/untar the downloaded file.

## Step 6: Configure HBase

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

- You just need to set these properties:

```
<?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>
```

- 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**

# Step 7: Downloading Karma

- Requisites:
  - Oracle Java JDK must be already installed on your system (version 1.7)
    - <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (link Oracle)
    - <https://help.ubuntu.com/community/Java> (link for Ubuntu).
  - Apache Maven must be already installed on your system (version 3.0)
    - <https://maven.apache.org/>
    - Make sure that M2\_HOME and M2 environment variables are set as described in Maven.
    - You can verify the installation of Maven through this commad (from command line): ***mvn -version***.

## Step 8: Installation Karma

- Download Karma from this site:  
<https://github.com/InformationIntegrationGroup/Web-Karma/archive/master.zip>
- To install HBase you must execute the following operations:
  1. unpack your zip file in a specific folder;
  2. open a shell (command line prompt) and go to the directory where you put Karma;
  3. On the shell prompt type ***mvn clean install*** (this command compiles and installs Karma).

## Step 9: Running Karma

- After installation, you can run Karma without an internet connection.
- To run Karma:
  1. Change to the karma-web sub-directory by executing `cd karma-web` in the shell (command line prompt);
  2. Start the server part of Karma by typing ***mvn jetty:run*** in the shell;
  3. Wait until you see the [INFO] Started Jetty Server in the shell;
  4. Open a Web Browser and go to page ***<http://localhost:8080>***.

4

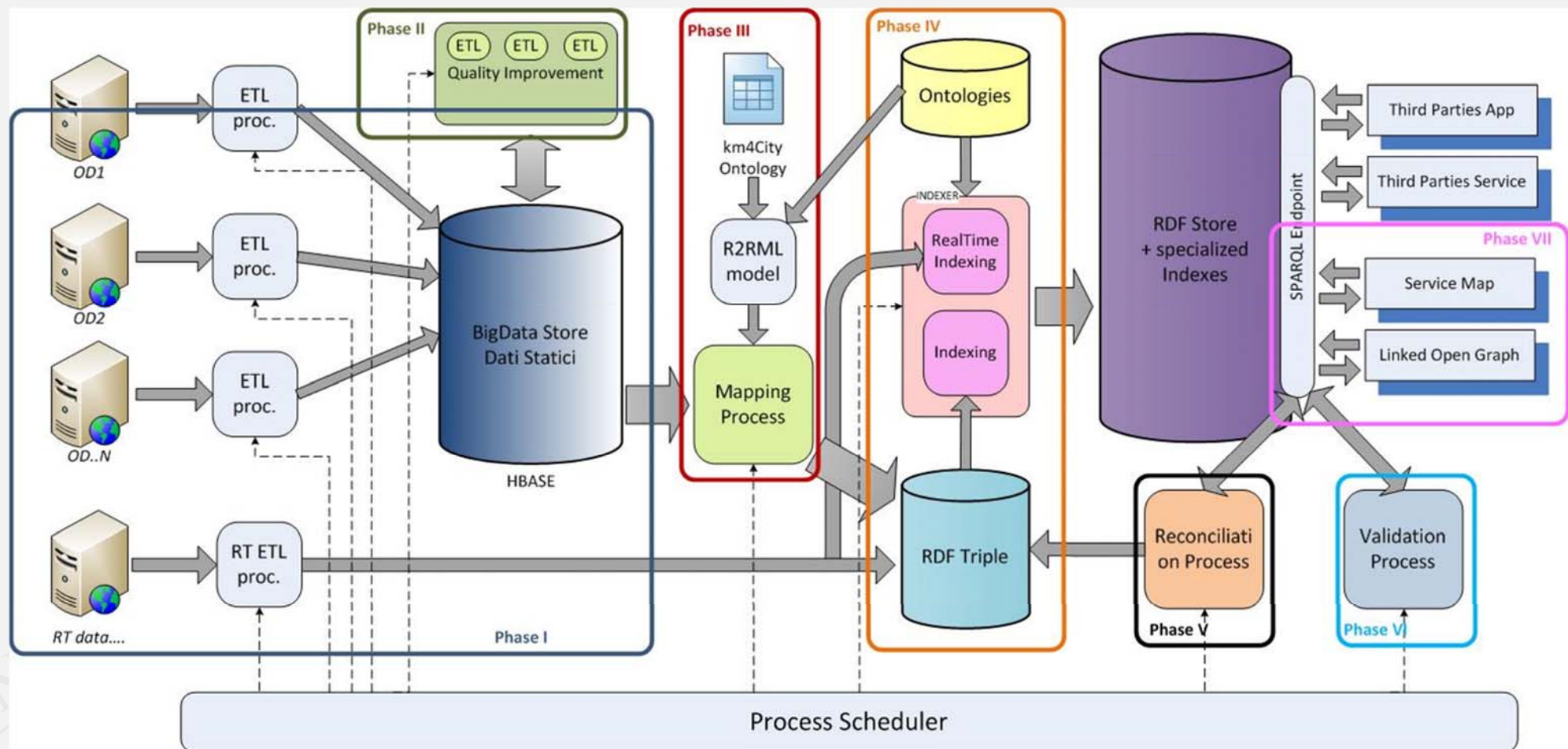
# Sii-Mobility Project

Developing  
ETL  
processes  
with PDI



# Sii-Mobility project

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



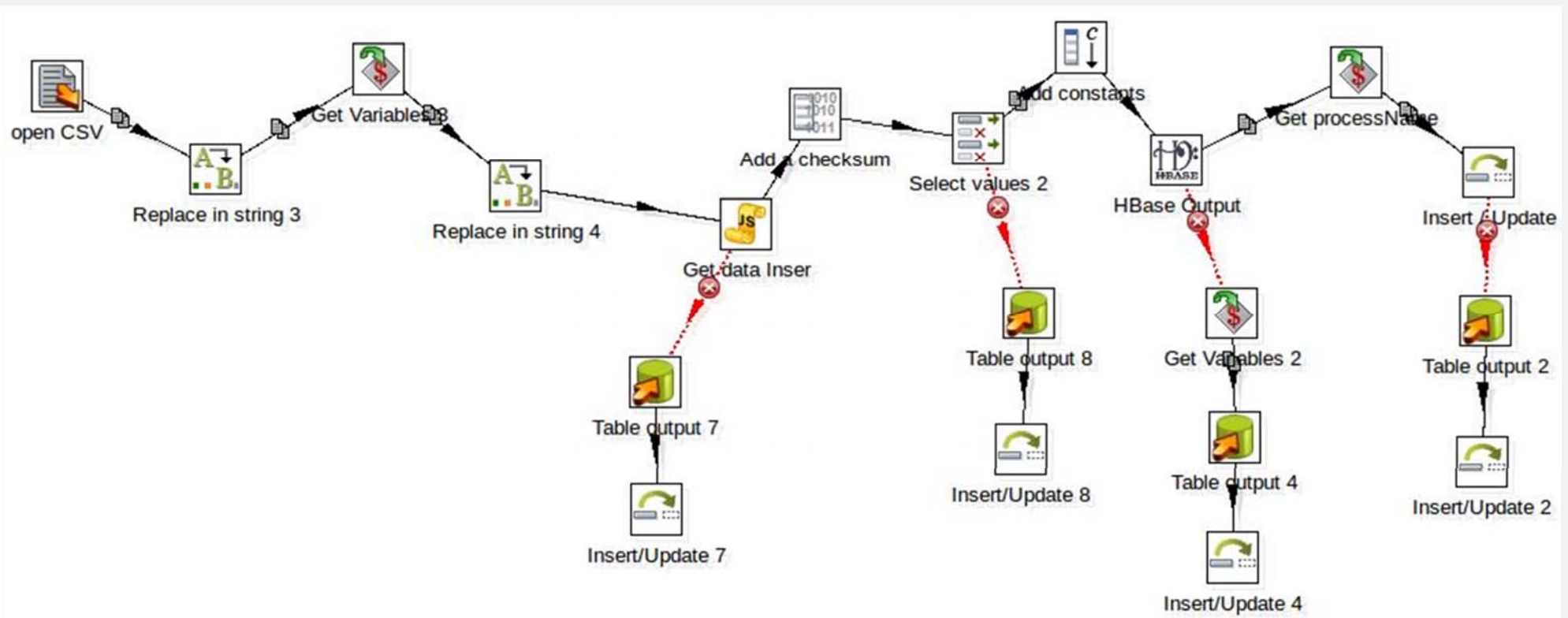


# Transformation Service Data

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

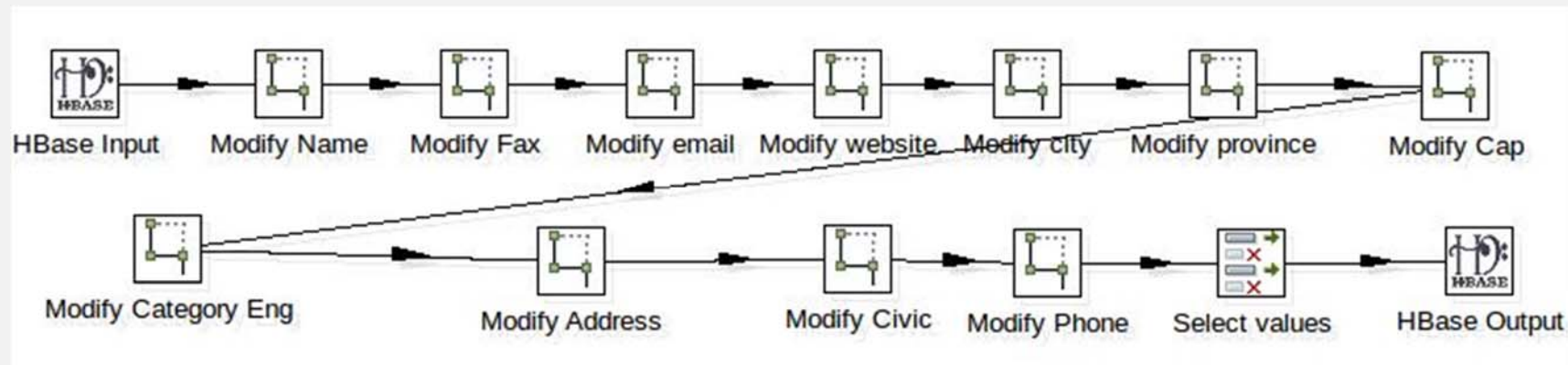
# Ingestion phase

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 the 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. For 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 re-use of the transformation.
- 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 into the fields that the sub-transformation accepts as input. A similar mapping happens when the processed rows are returned to the parent.

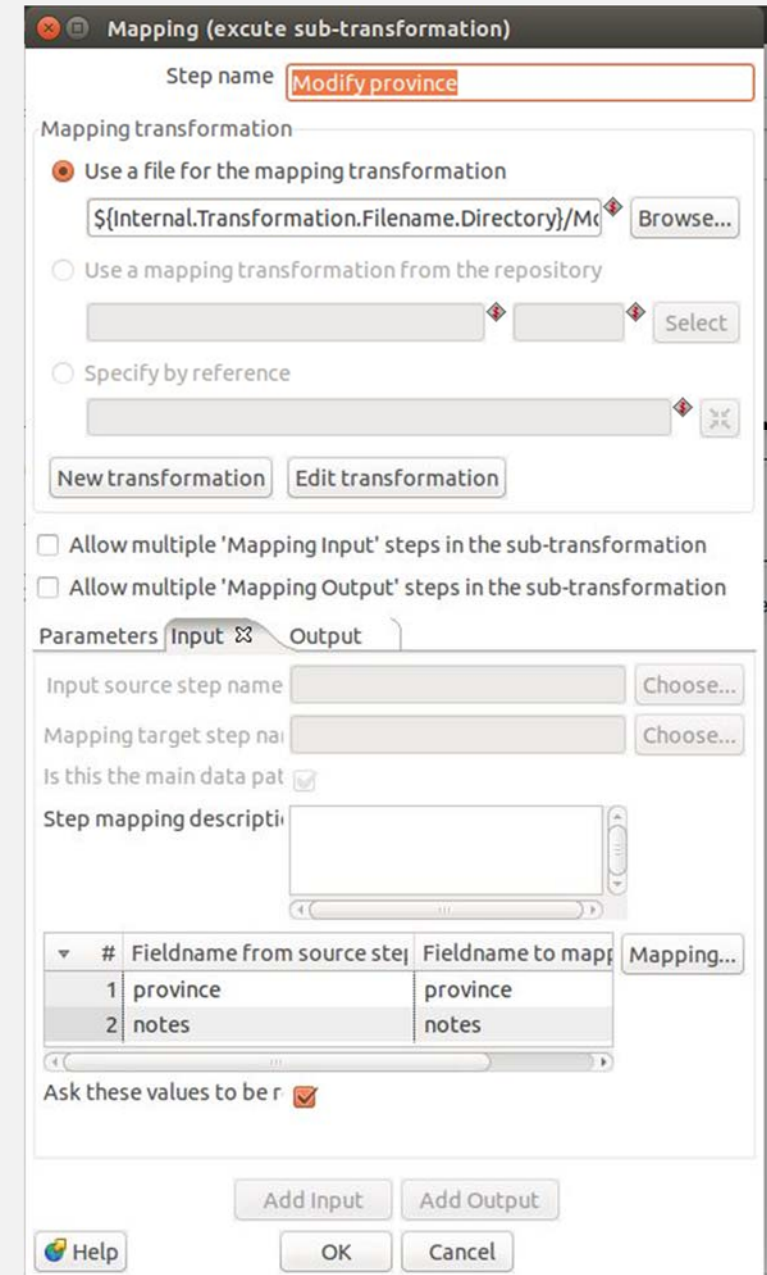




# QI phase

## Mapping

- To re-use a transformation you:
  - specify the sub-transformation to be executed;
  - 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.
- This can be seen like a function that returns output values calculated on specific input data.



Mapping (execute sub-transformation)

Step name

Mapping transformation

☒ Use a file for the mapping transformation

☐ Use a mapping transformation from the repository

☐ Specify by reference

☐ Allow multiple 'Mapping Input' steps in the sub-transformation  
☐ Allow multiple 'Mapping Output' steps in the sub-transformation

Parameters

Input source step name

Mapping target step name

Is this the main data path? ☒

Step mapping description

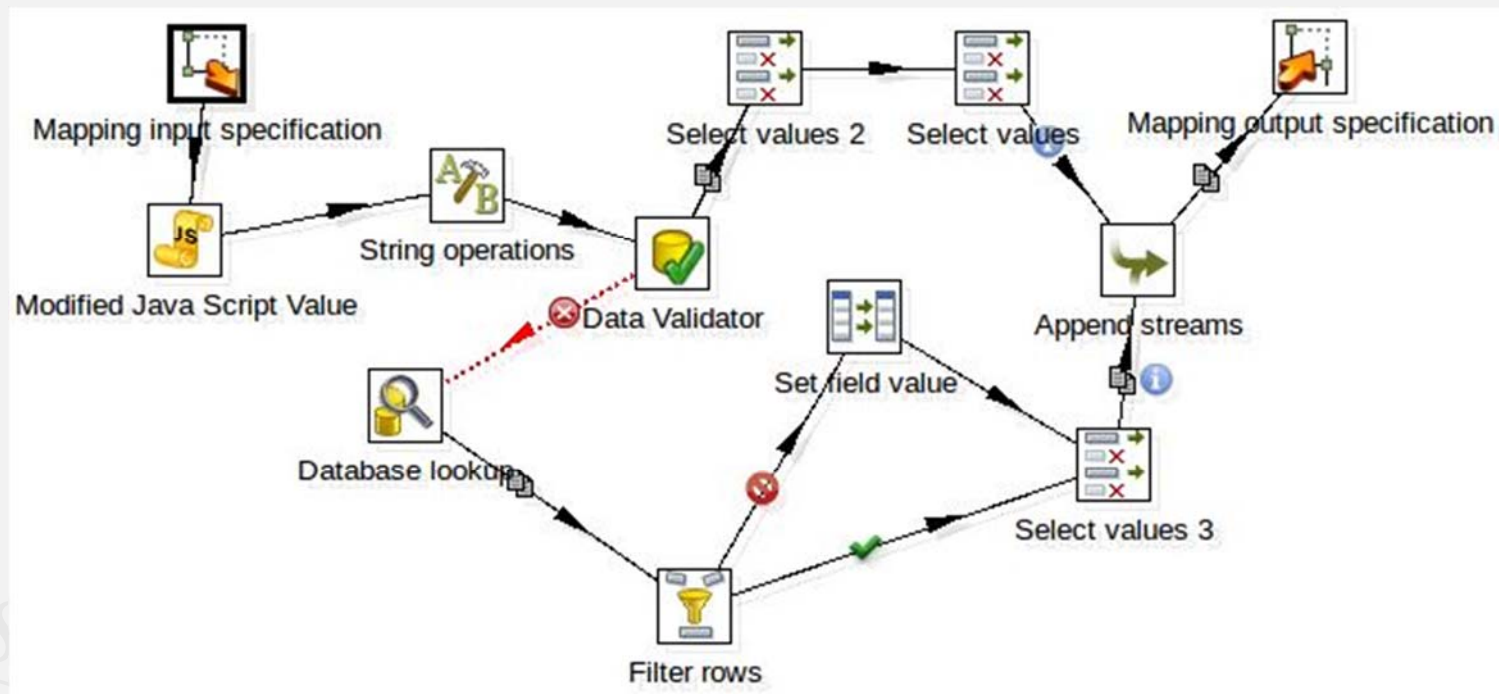
#	Fieldname from source step	Fieldname to map
1	province	province
2	notes	notes

Ask these values to be r ☒

# QI phase

## Mapping

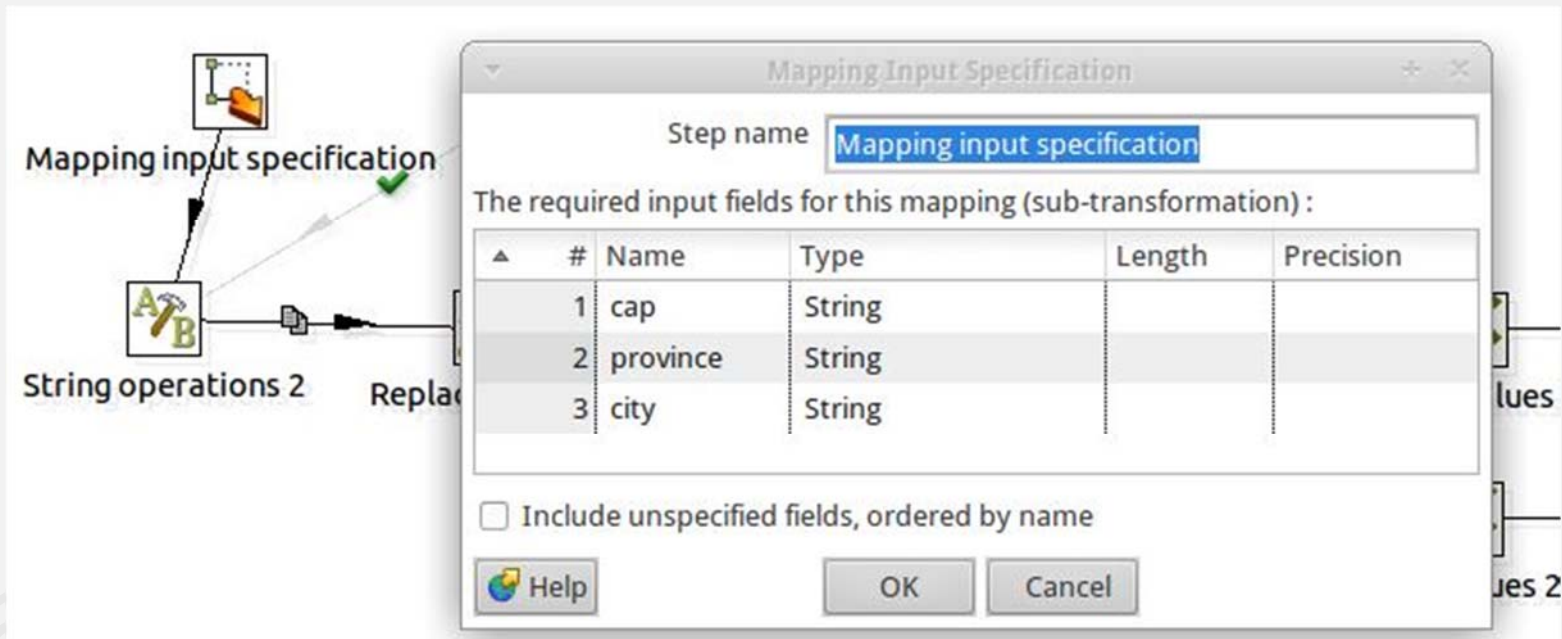
- The sub-transformation called by the mapping is a transformation just like any other with a couple of key differences.



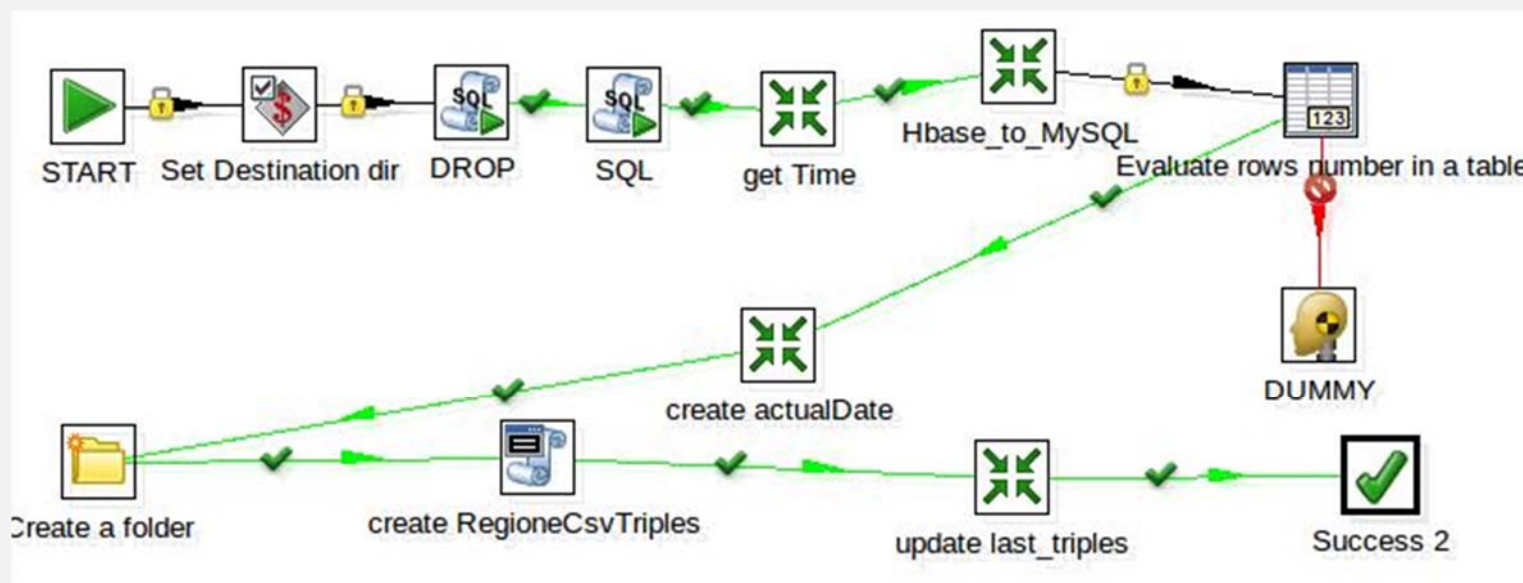
# QI phase

Every sub-transformation called by the mapping requires these additional steps:

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



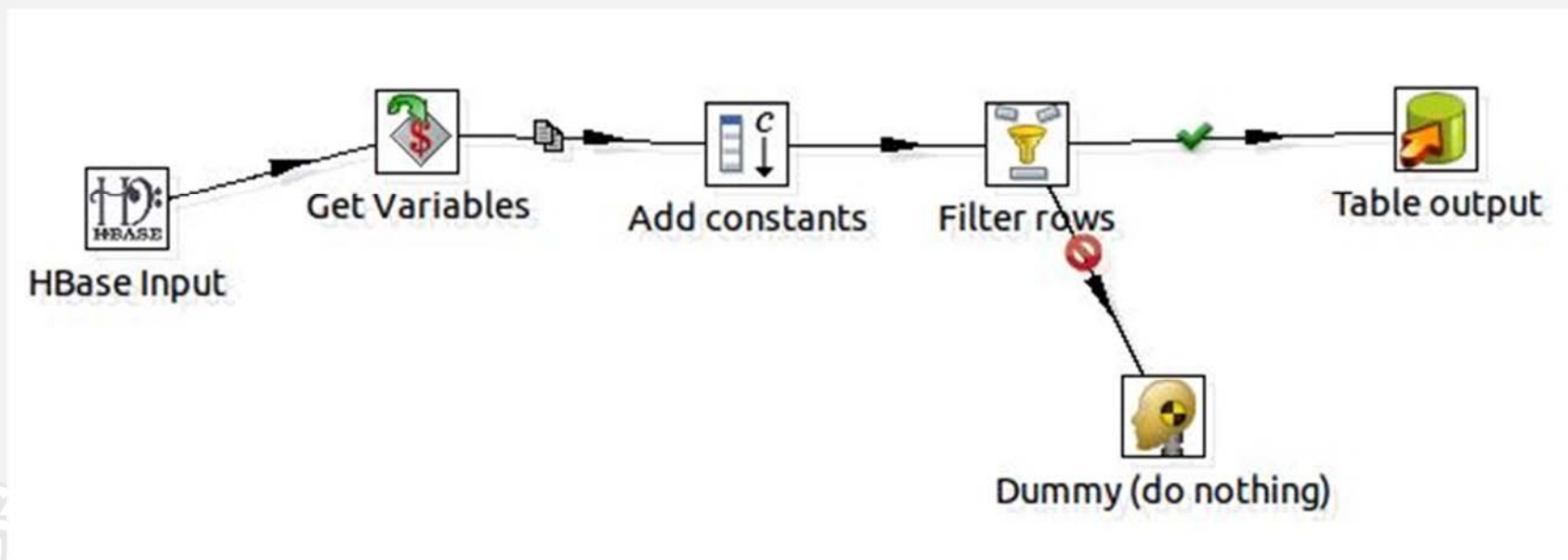
# RDF Triples generation phase



The triples are generated with **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.

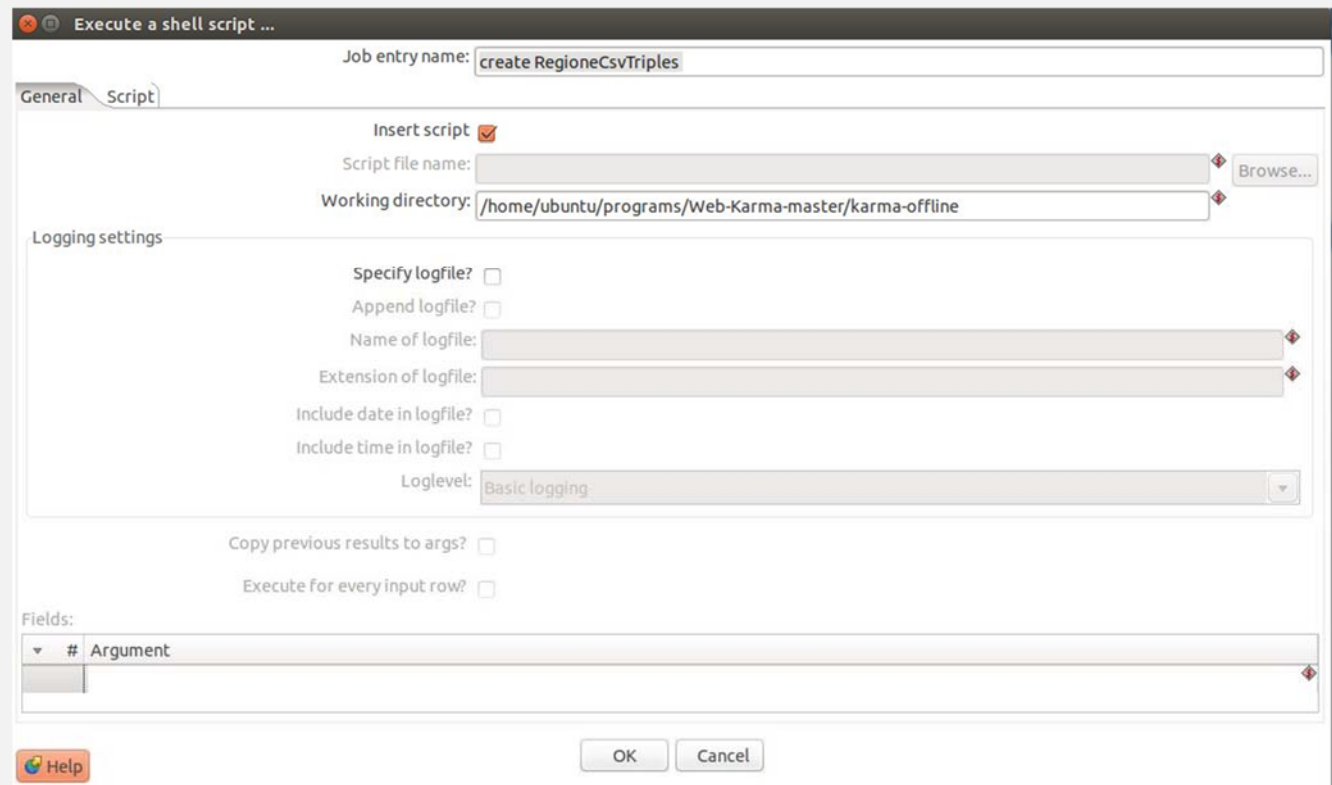




# RDF Triples generation phase

## 2. Creating RDF triples from data loaded previously through a specific script.

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

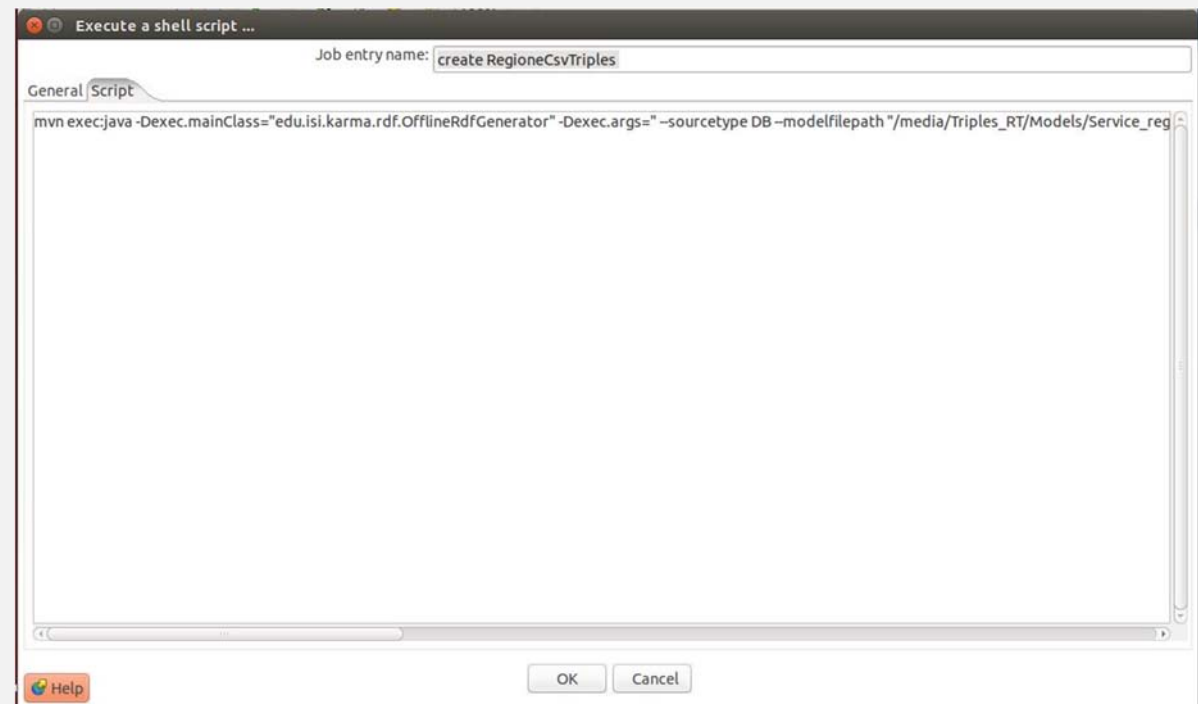
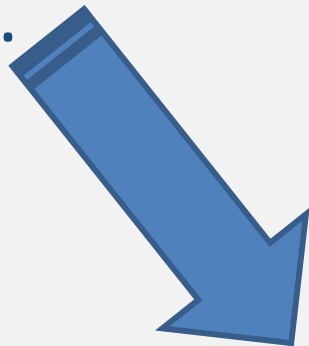


- You can check the option “Insert Script” if you want to execute the script in the Script tab instead of executing the one contained in *Script file name* field.

# 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 (.ttl), the database table (where you get source data) and the connection parameters to database;
  - the mapping model must be created previously with Karma.
- In output you specify the file name (.n3) where the RDF triples will be stored.

# Transformation Parking

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



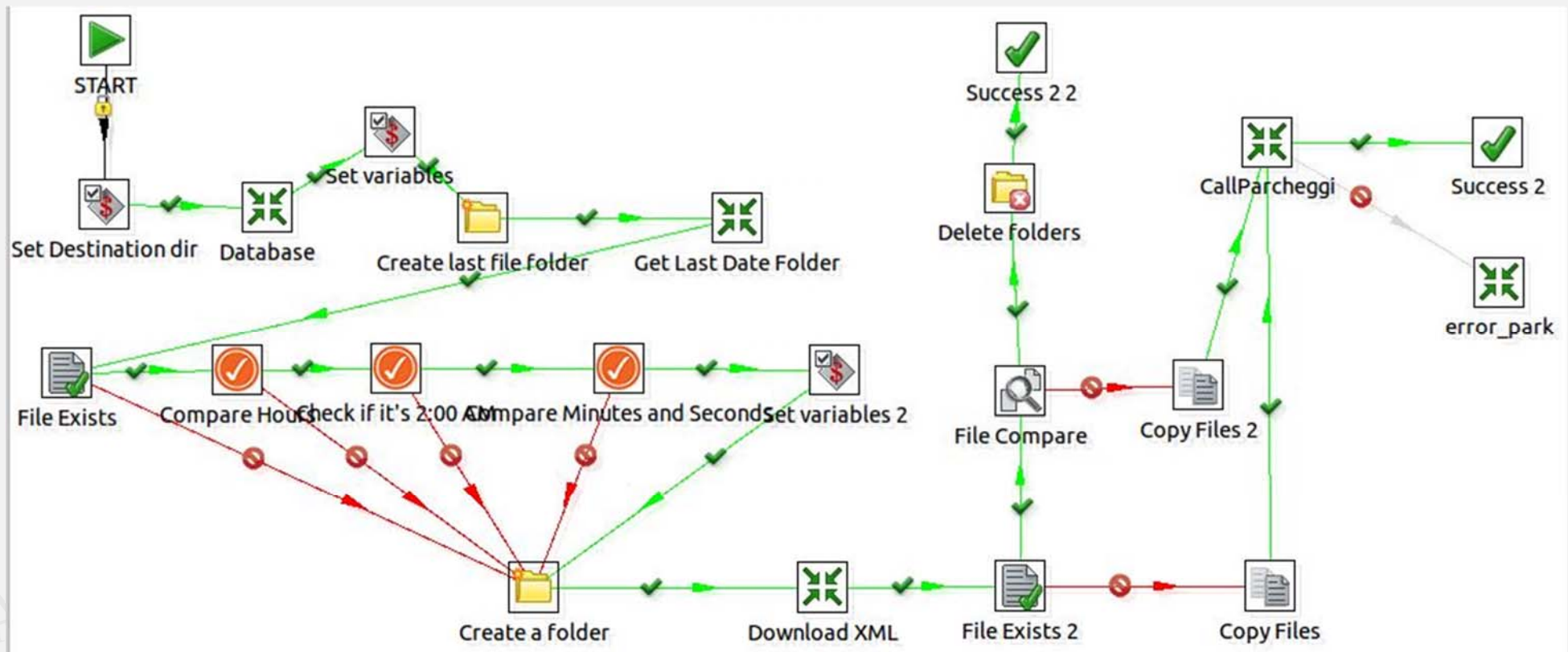
The screenshot shows the SIM Consultazione web interface. It features a navigation menu on the left with 'Classi Dati' expanded, showing 'TPL', 'Infrastrutture di trasporto', and 'Tempo reale'. The main content area is titled 'Consultazione Tempo reale' and contains a table with the following data:

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

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

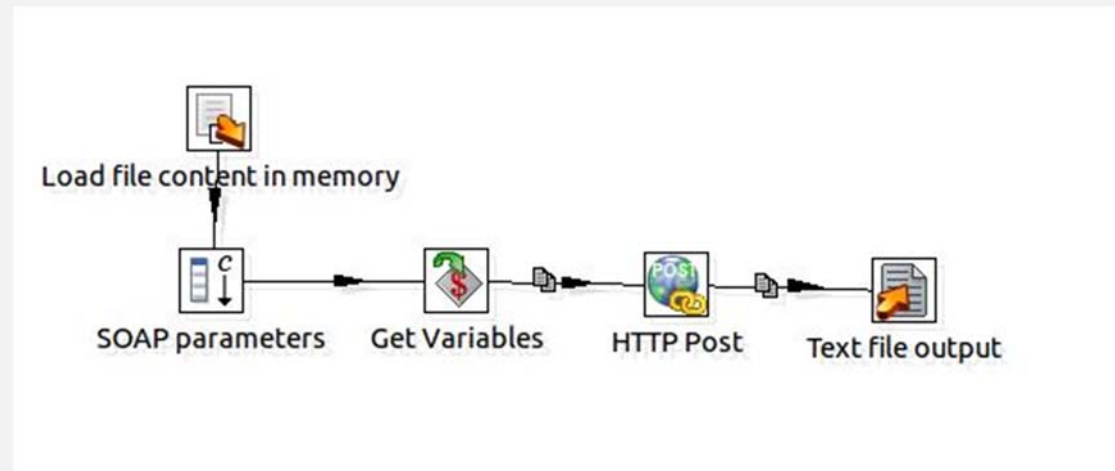
# Ingestion phase

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 catalog parameter to identify a sensors group.
4. Invocation of the web service with ***HTTP Post step***;
  - downloaded data are saved in a xml file (*Text file output step*)
5. Storing data on HBase databse (*CallParcheggi transformation*).



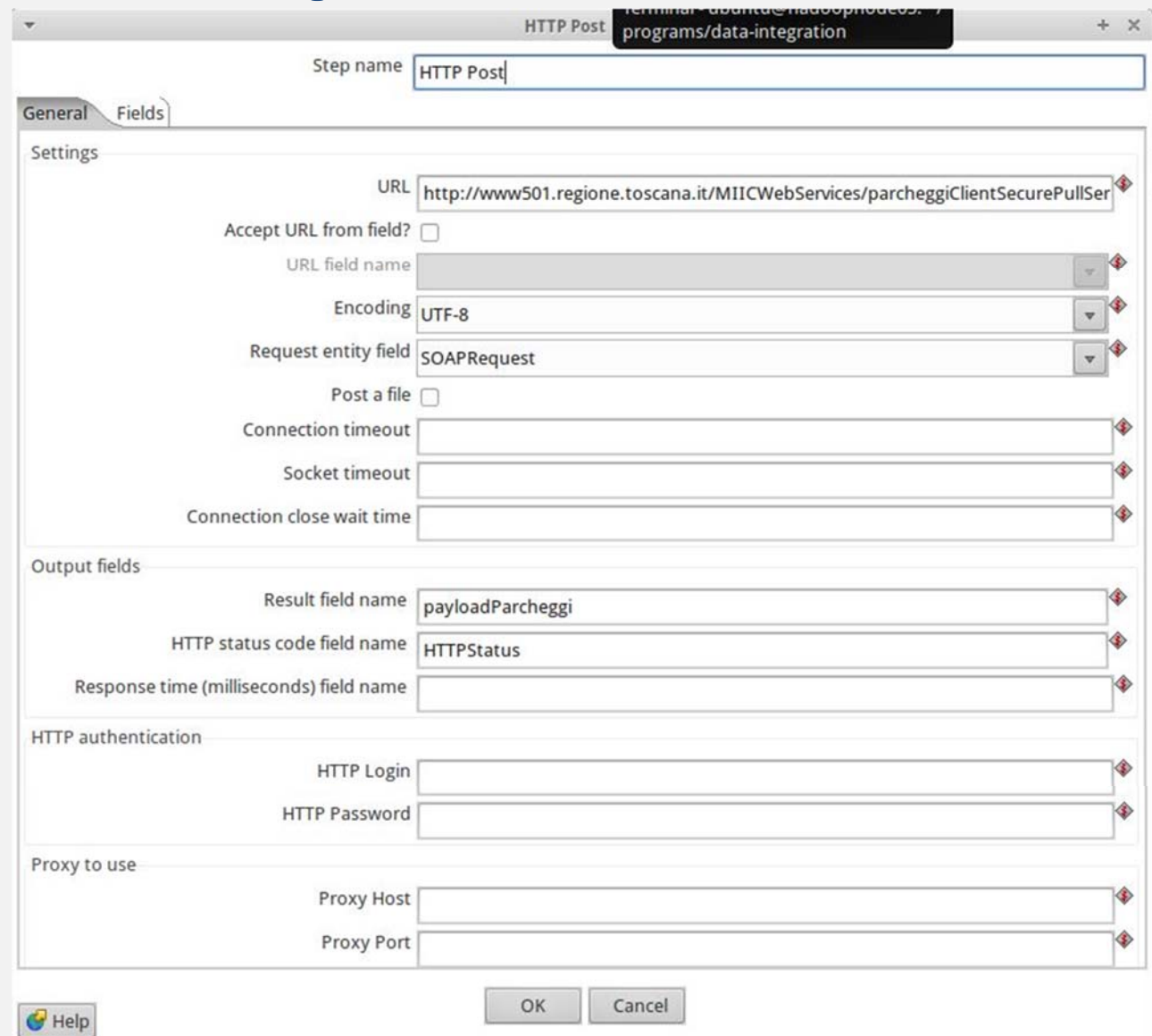
#	Alias	Key	Column family	Column name
1	FinalKey	Y		
2	actualDate	N	Family1	actualDate
3	carParkIdentity	N	Family1	carParkIdentity
4	carParkOccupancy	N	Family1	carParkOccupancy
5	carParkStatus	N	Family1	carParkStatus
6	catalog	N	Family1	catalog
7	exitRate	N	Family1	exitRate
8	fillRate	N	Family1	fillRate
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



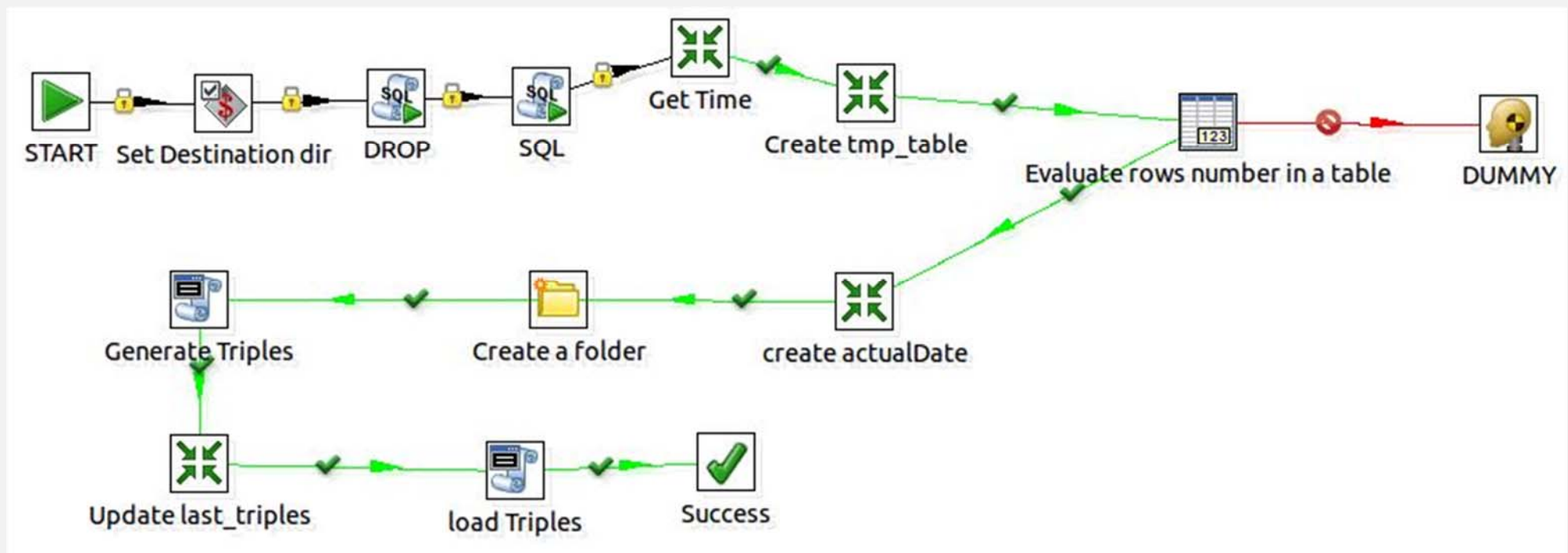
# Ingestion phase

## HTTP Post

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

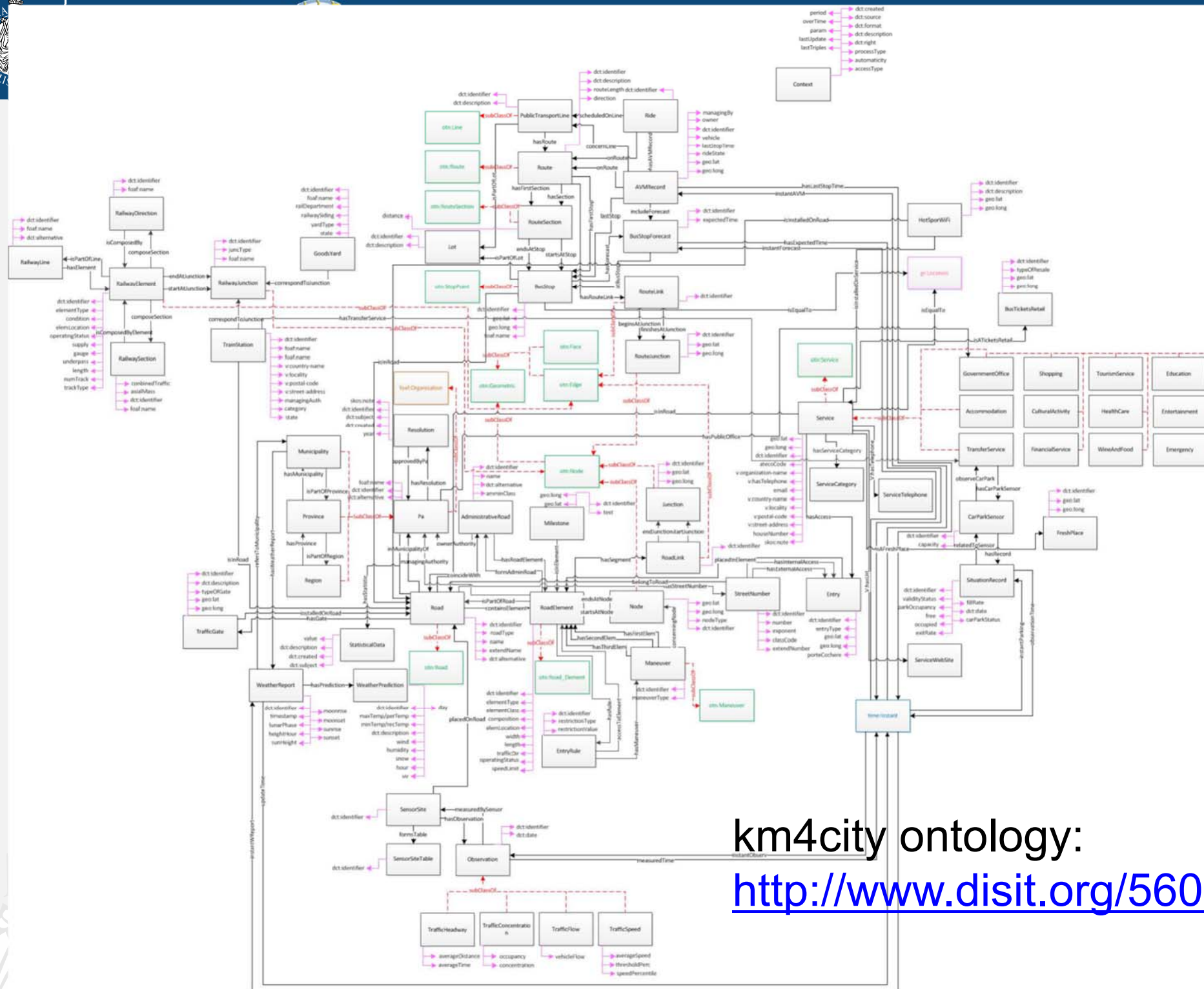


# RDF Triples generation phase



The RDF triples are generated with **km4city** ontology (<http://www.disit.org/5606>) and then loaded on Virtuoso 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**).
- A scheduler is used to manage the periodic execution of ingestion and triple generation processes;
  - this tool throws the processes with a 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 (Example)

	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



5

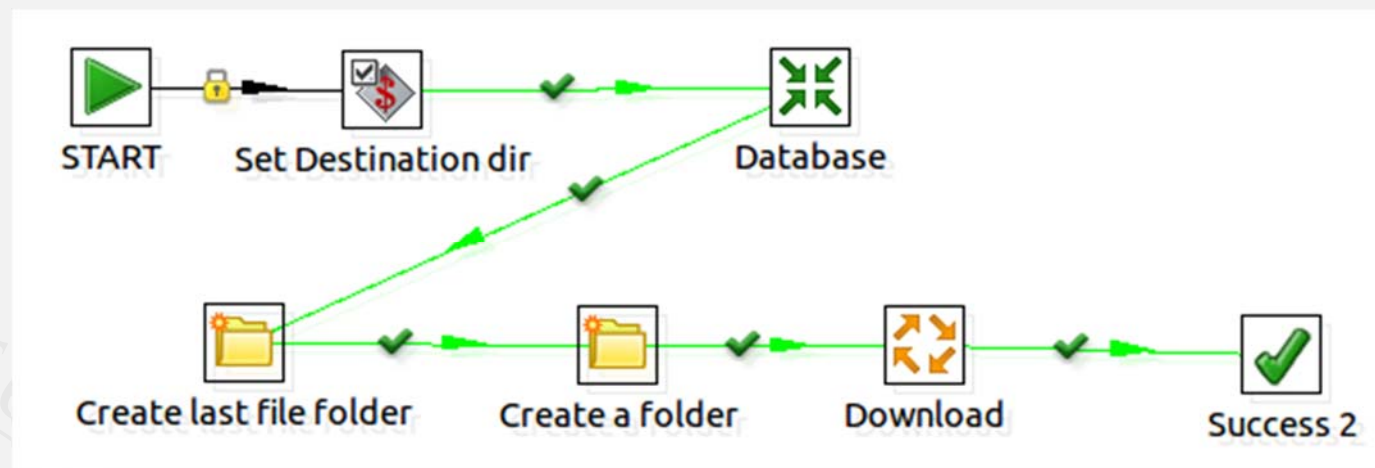
# Data Ingestion phase (1)





# 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 the data in specific HBase table.
  - The acquired dataset is “Strutture ricettive” in CSV format.
  - All the information characterizing the process are stored into a specific MySQL table (*process manager 2*).



# Phase 1: Set storage root folder

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



Set variables...

Job entry name: **Set Destination dir**

Properties file

Name of properties file

Variable scope: Valid in the Java Virtual Machine

Settings

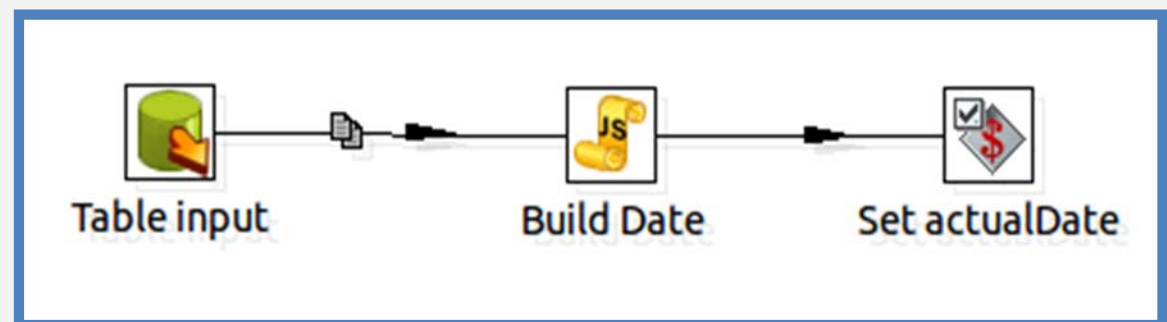
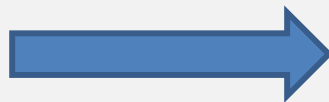
Variable substitution? ☒

Variables :

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

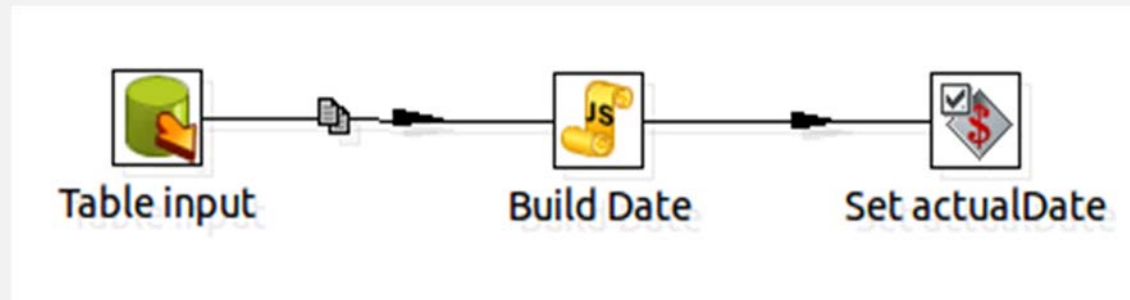
## Phase 2: Creating storage folder

- **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 retrieves some information characterizing the process and calculates the current date. From this date the variables to build the remaining part of path in which the downloaded file will be stored are extracted. The path must be **/Sources/Categoria/NomeProcesso/Anno\_mese/Giorno/Ora/MinutiSecondi/NomeProcesso.xxx**.

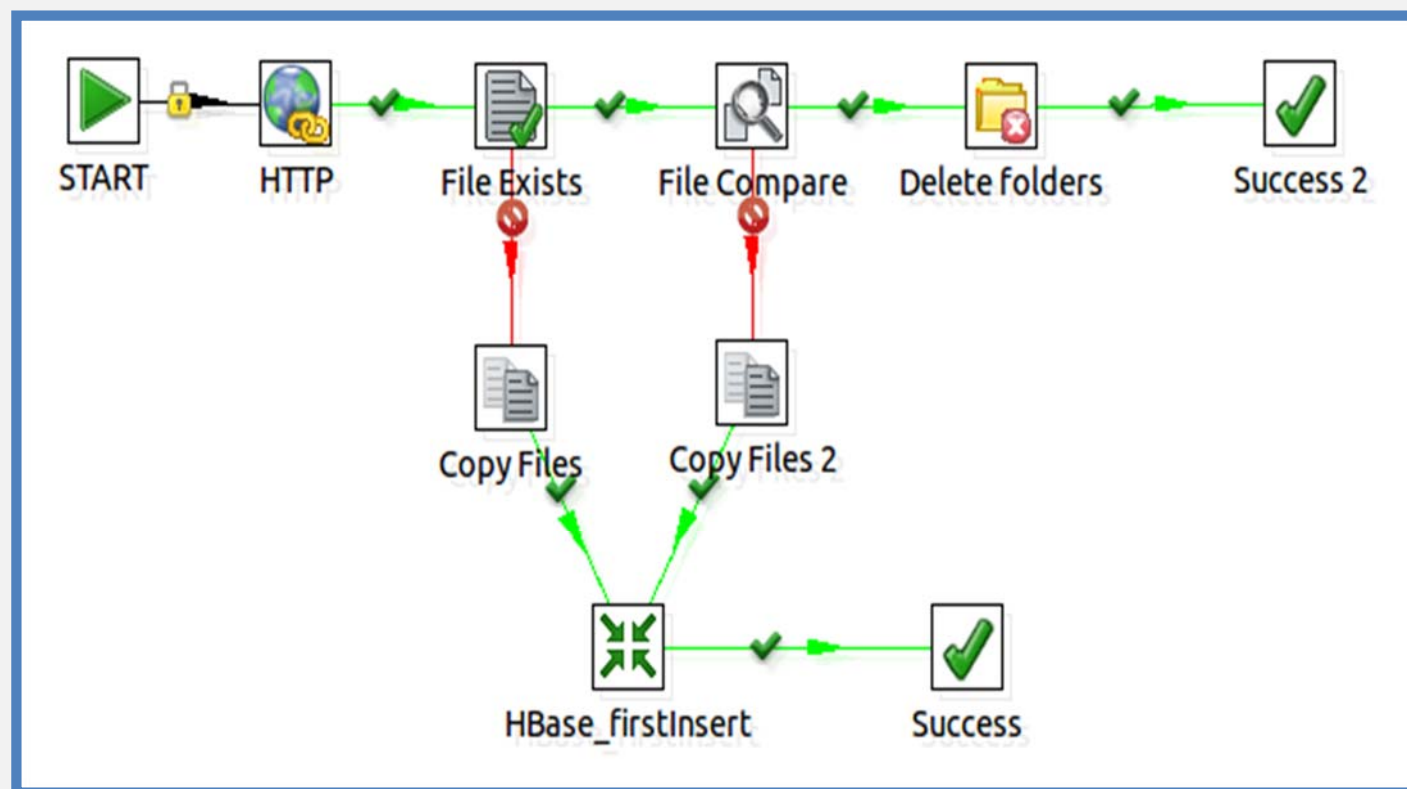
# Phase 2: Creating storage folder

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



# Phase 3: Download/Store dataset

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



# Phase 3: Download dataset

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



URL:

result row? ☐

contains URL

Username:

Password:

r for upload:

Proxy port:

xy for hosts:

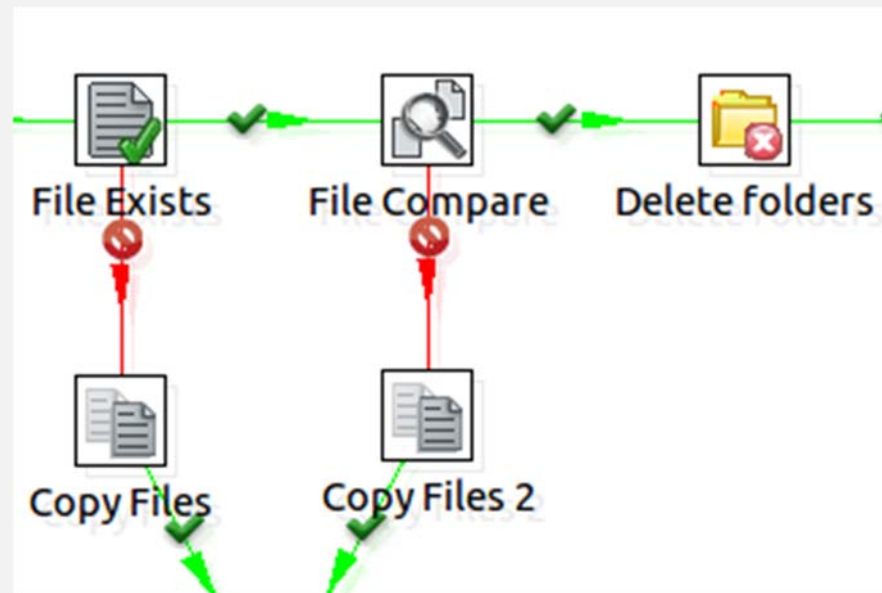
Upload file:

Target file:



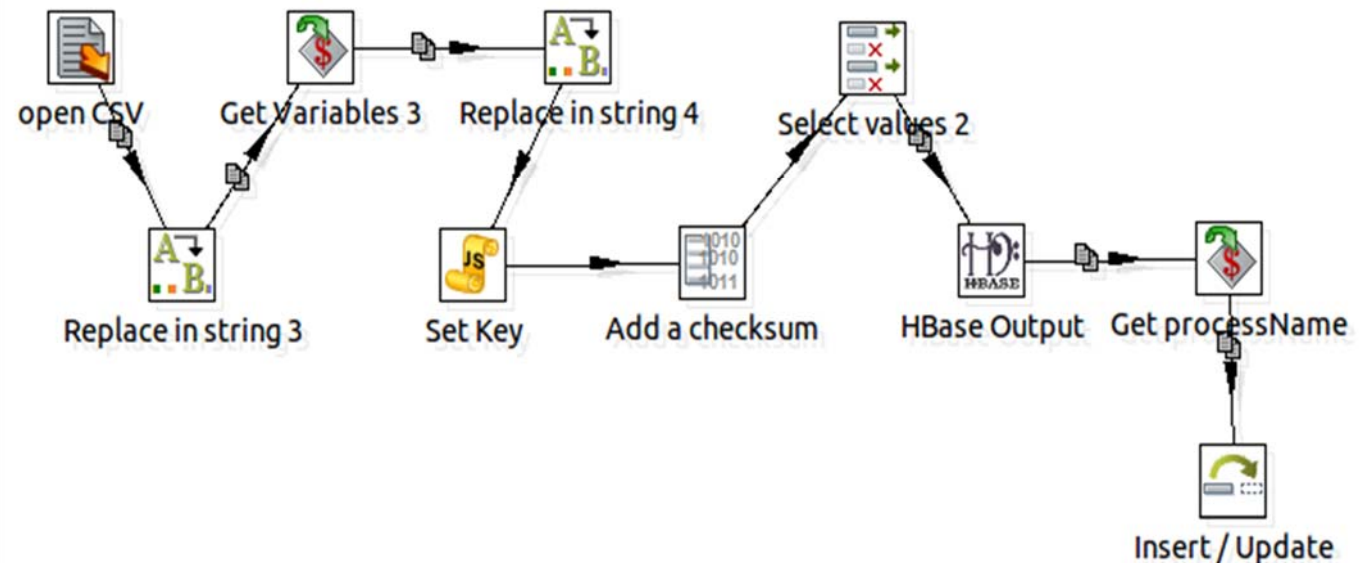
## Phase 3: Store dataset

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



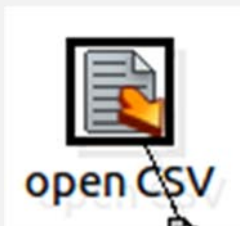
# Phase 4: Load data into HBase

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



# Phase 4: Load data into HBase

- **PDI step: Text file input**
- Select the source file you just downloaded (Strutture ricettive.csv), choose the separator type of the CSV file and select the fields to be imported.

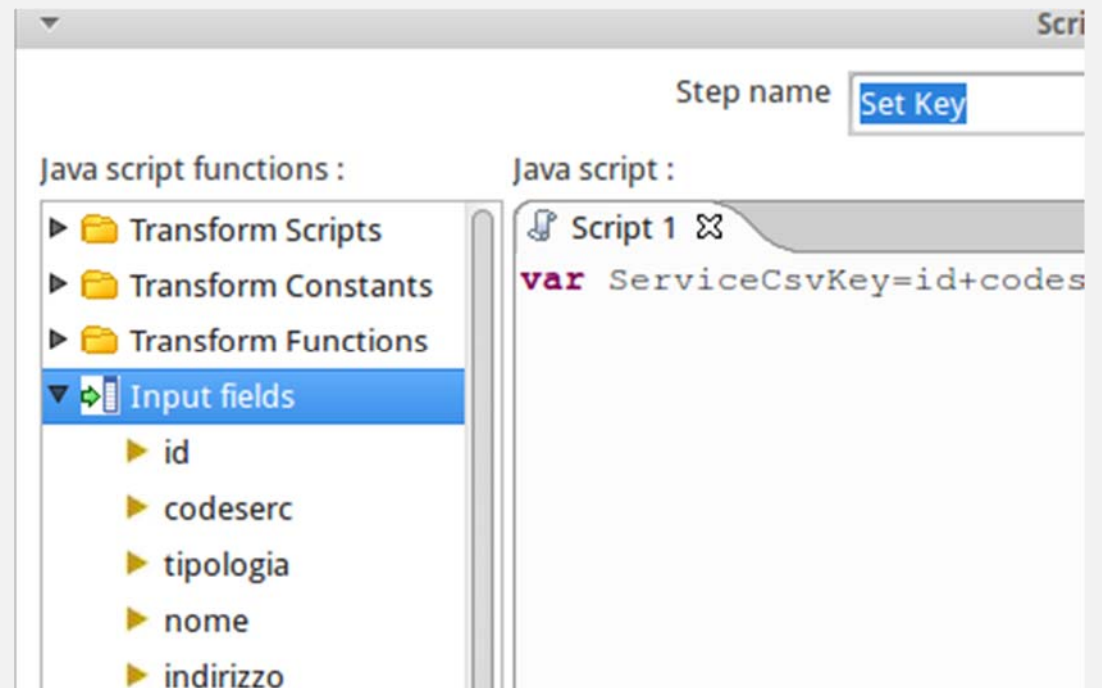


Step name

	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			

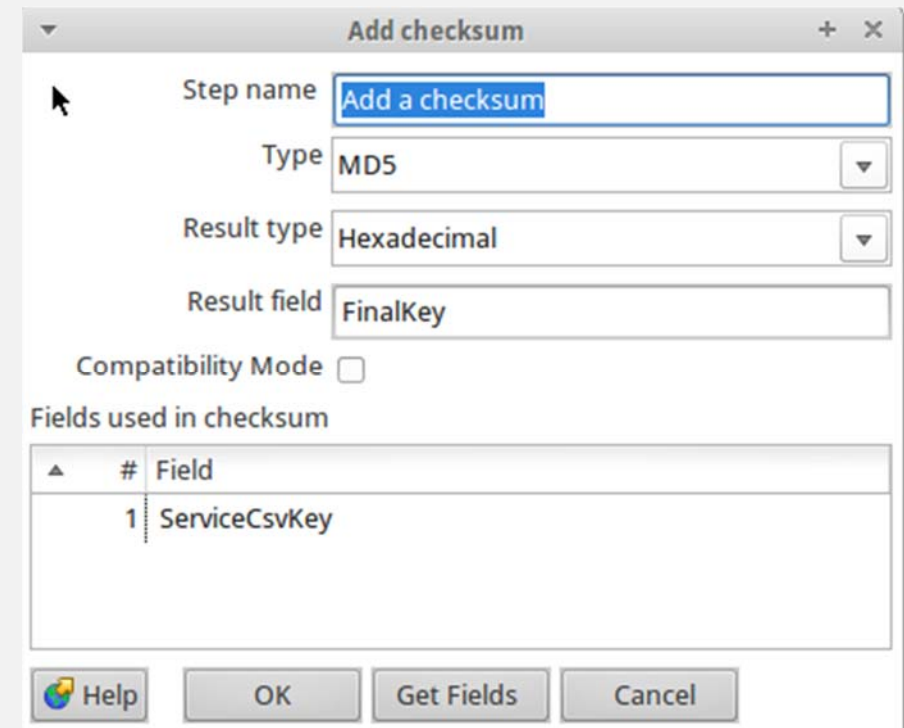
## Phase 4.1: Defines a key

- **PDI step: Modified Java Script value**
- Define a variable by concatenating the input fields coming into the step. This variable will be used for define the storage key field.



# Phase 4.1: Defines a key

- **PDI step: Add a checksum**
- Choose the algorithm (MD5, CRC32) for encoding the storage key field. You must also define the new name in output.

The dialog box 'Add checksum' contains the following fields and options:

- Step name:** Add a checksum
- Type:** MD5
- Result type:** Hexadecimal
- Result field:** FinalKey
- Compatibility Mode:** ☐
- Fields used in checksum:**

#	Field
1	ServiceCsvKey

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

## 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 values 2**

Select & Alter Remove Meta-data

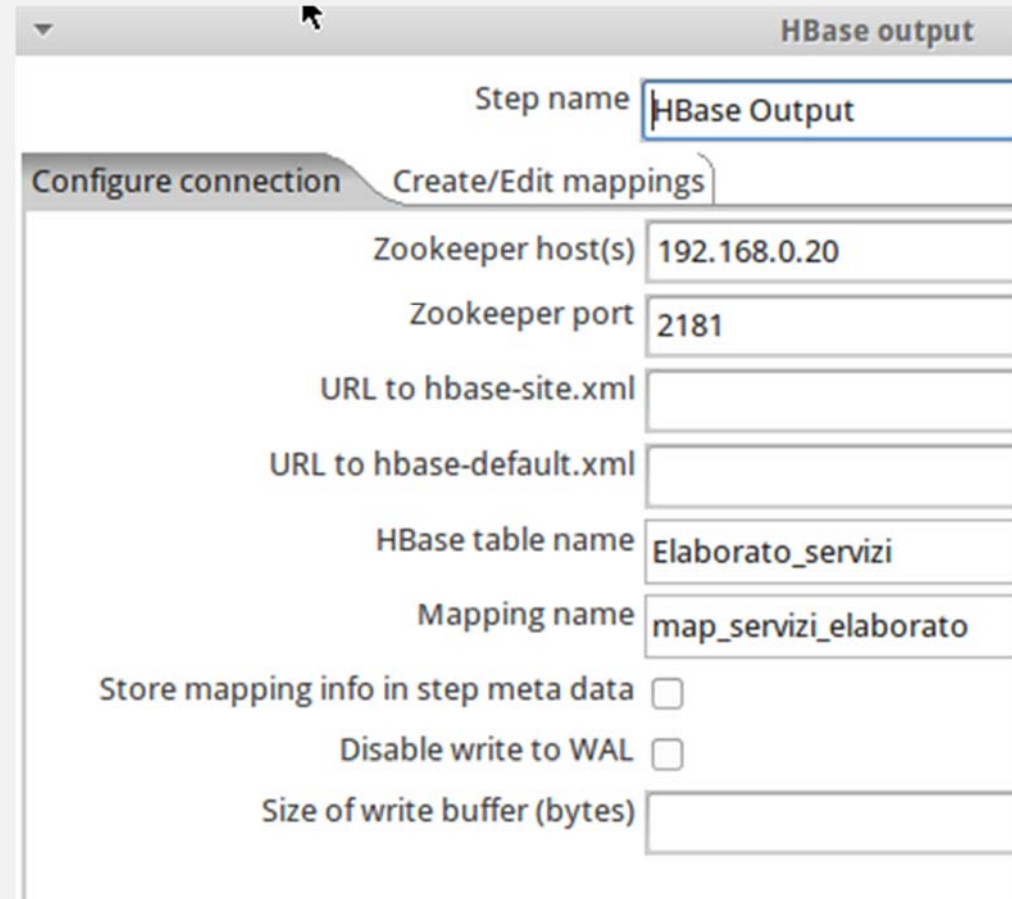
Fields :

▲	#	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 (IP address of the machine that hosts the HBase database) and the port (2181).
- Specify the HBase storage table.
- Define a mapping of the input fields on the specific HBase table.






# Phase 4.3: Update MySQL

- **PDI step: Insert / Update**
  1. create a connection to MySql database 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 be updated (in this case the *last update* field).



Insert / Update

Step name Insert / Update

Connection conn2\_SiiMobility@192.168.0.20

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 f
	1	process	=	process	

Update fields:

▲	#	Table field	Stream field	Update
	1	last_update	actualDate	Y

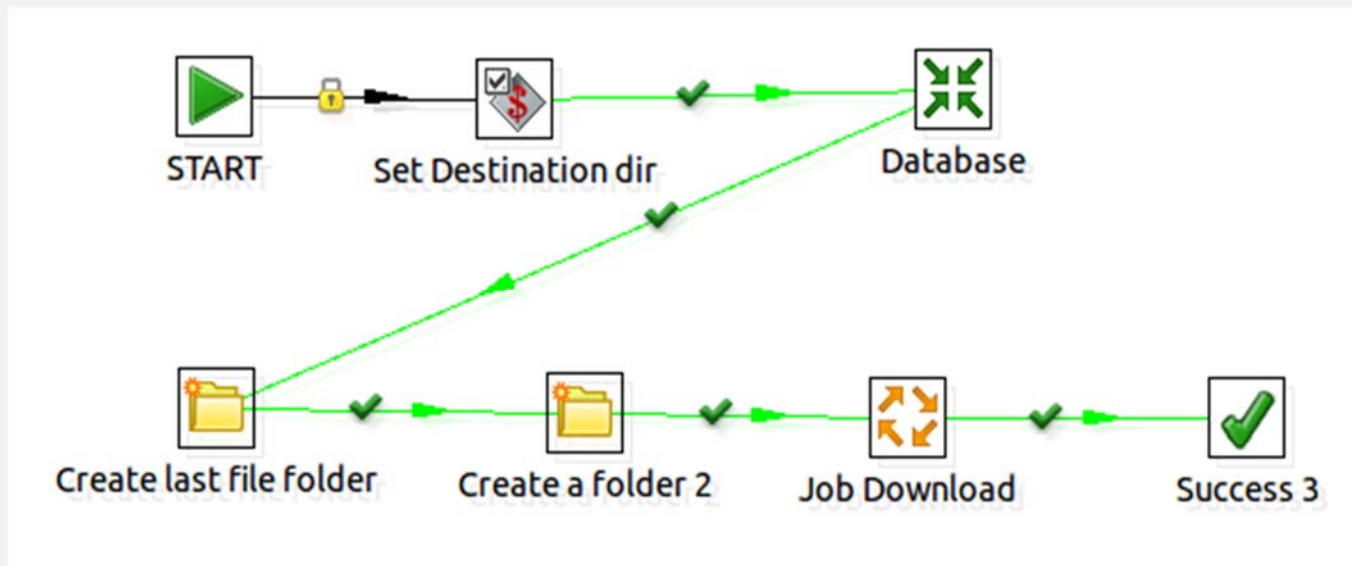
# Job Data Ingestion KMZ (1/6)

- 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.
  - All the information characterizing the process are stored into a specific MySQL table (*process manager 2*).



# Job Data Ingestion KMZ (2/6)

- The Job is similar to the previous one, but differs in some points.



# Download.kjb (3/6)

- In *HTTP post* step the *URL* field is setted with the `${PARAM}` variable that contains the value retrieved from MySQL database (*process manager 2* table) . This value is the web address for download the source files.



Transfere

Name of job entry: HTTP

General Headers

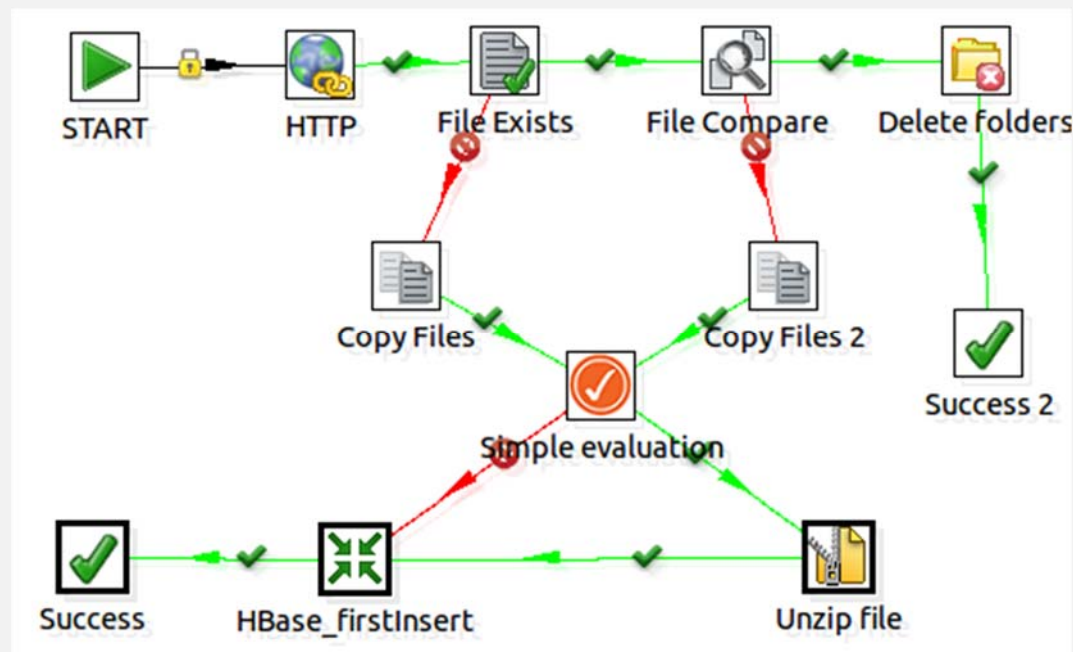
URL: \${PARAM}

Run for every result row? ☐

Input field which contains URL

# Download.kjb (4/6)

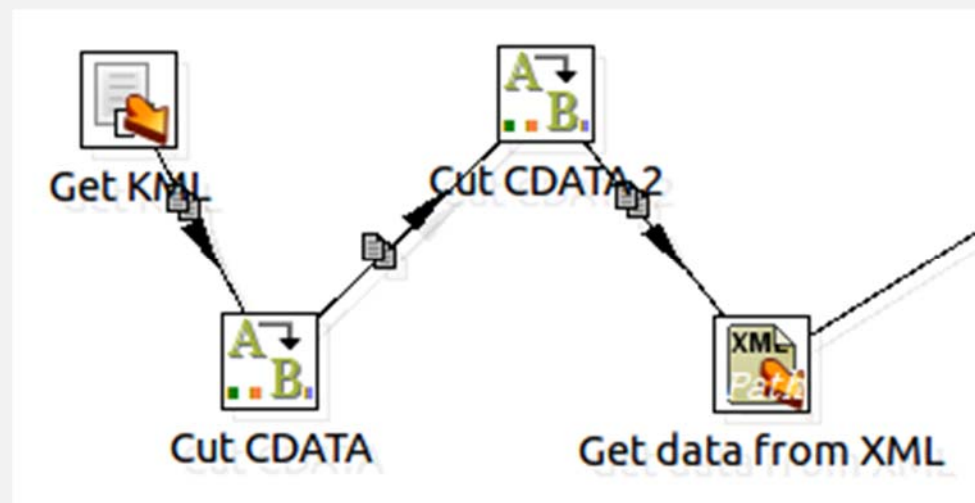
- The new step (*simple evaluation*) checks the format of the downloaded dataset. For KMZ files the execution flow is routed to the *Unzip file* step that extracts the KML file and stores it in the following folder:  
 **$\text{\$DestinationDir}/\text{\$processName}/1\text{Last\_file}$ .**



# HBase\_firstinsert (5/6)

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

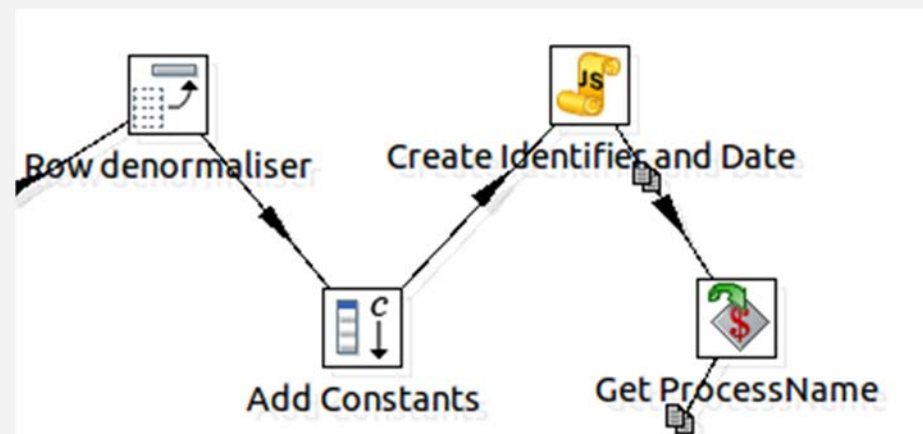
After some modifications, the entire content is passed to the *Get data from XML* step and the tag, span and coordinates fields are extracted through the use of XPath .





# HBase\_firstinsert (6/6)

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



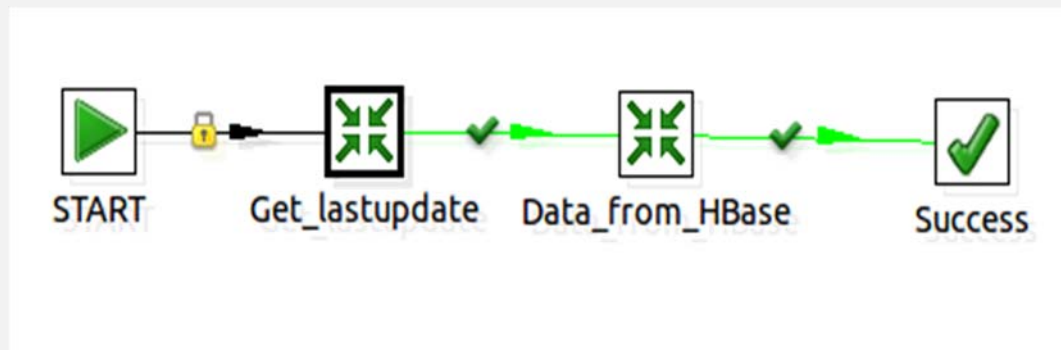


5

# Quality Improvement phase (2)

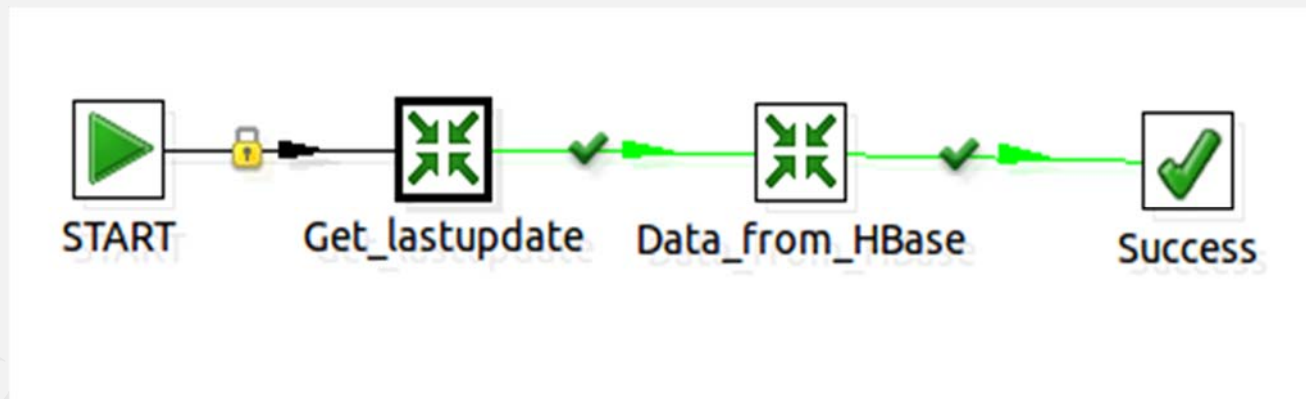
# Job Quality Improvement CSV

- This job reads data from HBase table created in Data Ingestion phase, improves the quality of data and re-loads data into HBase database (in new table).
- All the information characterizing the process are stored into a specific MySQL table (process manager 2).



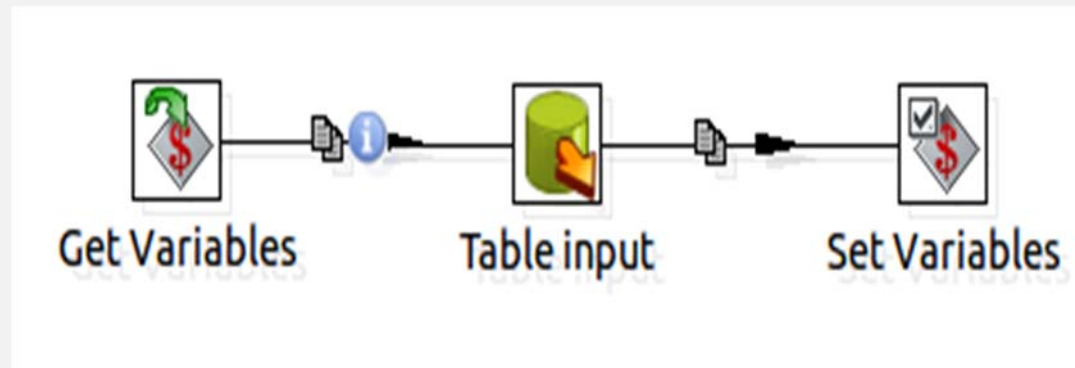
# Job Quality Improvement CSV

- This job invokes two transformations to:
  - get the update date of dataset (*Get\_lastupdate* step);
  - apply the quality improvement (QI) to fields stored in HBase table. Each field has its own QI transformation (*Data\_from\_Hbase* step).

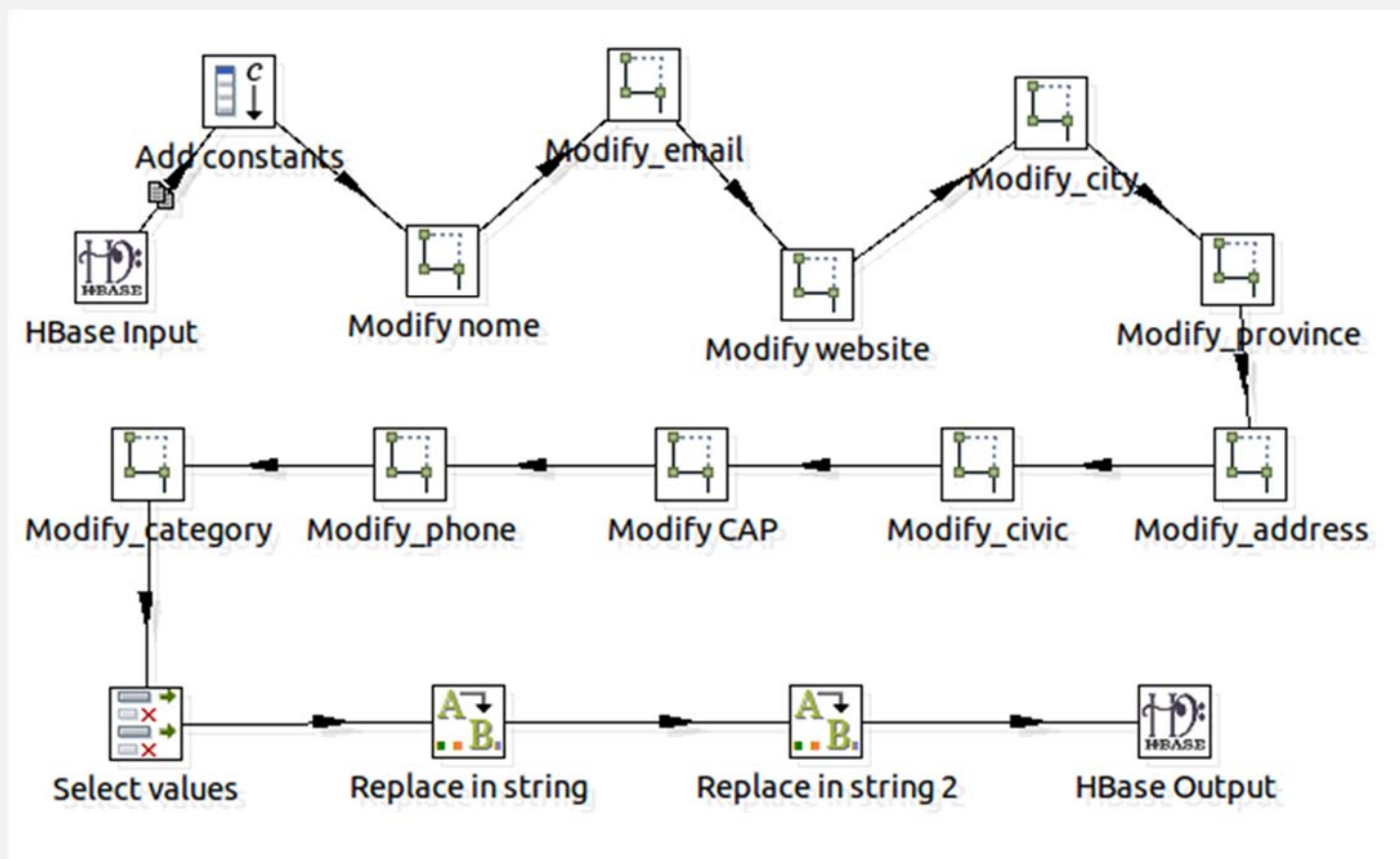


# Phase 1: Get dataset update data

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



# Phase 2: Data Quality Improvement



Each field has its own QI transformation (*mapping* step)

# Phase 2.1: Reading HBase table

- **PDI step: HBase input**
- Retrieve the data from HBase table specifying the Zookeeper host (IP address), 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

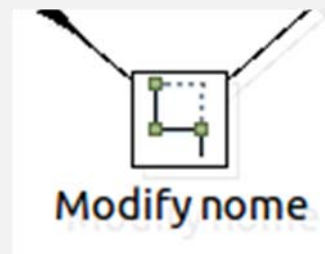
Fields :

▲	#	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. You must:
  - specify the QI transformation to be executed;
  - specify the input fields that are required by QI transformation;
  - specify the output fields that are required by QI transformation.



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

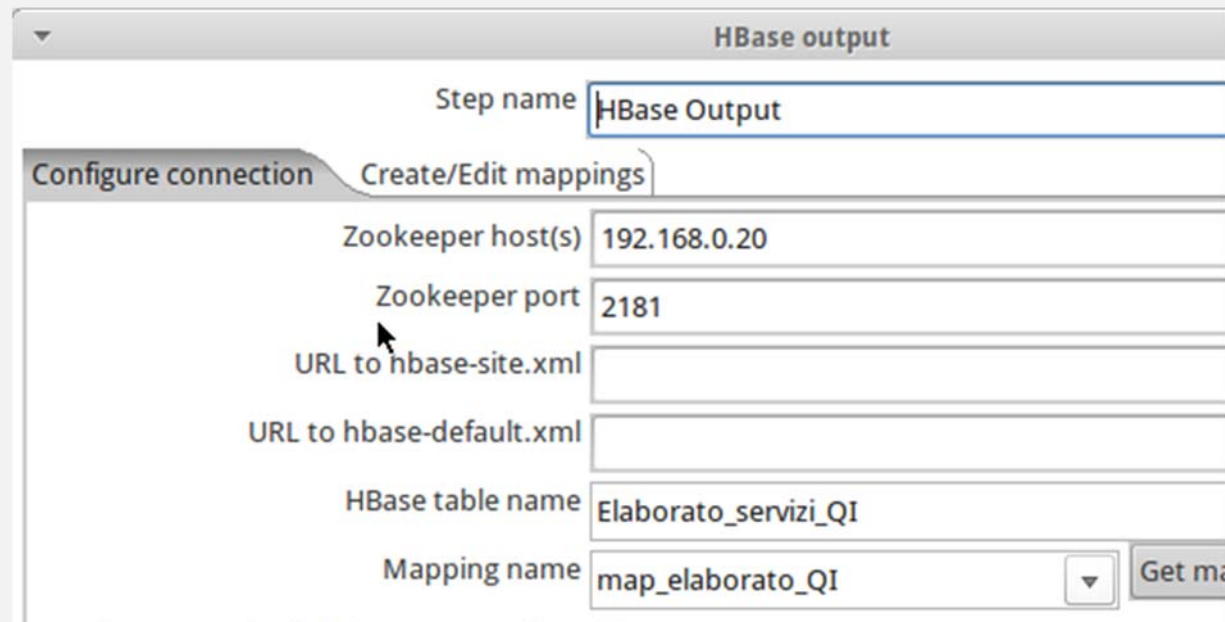
Input ☒ Output ☐ Parameters ☐

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

☒ Ask these values to be renamed back on output?

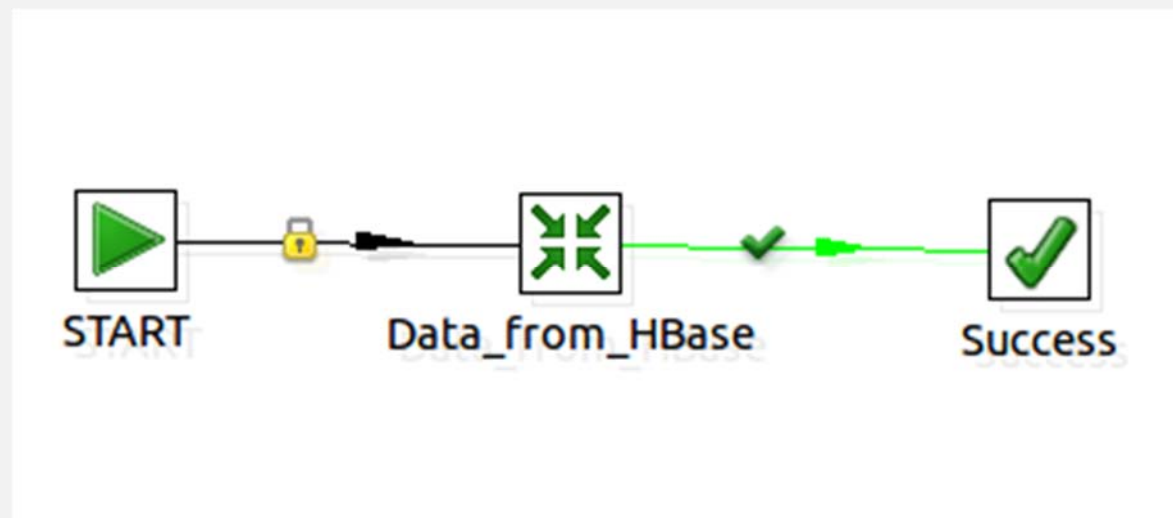
# Phase 3: Load data into HBase

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



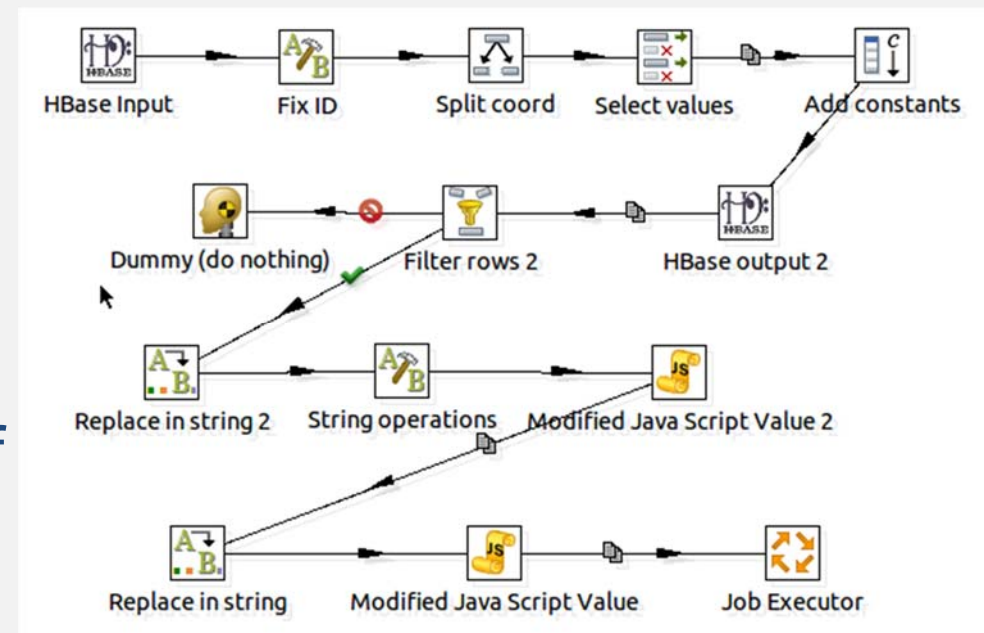
# Job Quality Improvement KMZ (1/3)

- 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 the *Data\_from\_HBase* transformation.



# Job Quality Improvement KMZ (2/3)

- *Data\_from\_HBase* transformation is used to:
  - apply the quality improvement (QI) to the fields stored in Hbase;
  - in this case the dataset contains geotagged information (latitude and longitude), so a reconciliation operation is made in order to retrieve the toponimo code of the street relating to the service.



# Reconciliation operation (3/3)

- Goal: associate with each service contained in the data set its code name, which identifies a specific road within the road graph provided by the Tuscany Region.
- To achieve this, a specific MySQL table (*tbl\_toponimo*) containing all the toponimo codes is used. This table is also enriched with the geographic coordinates of a point corresponding to the approximate center of each road.

5

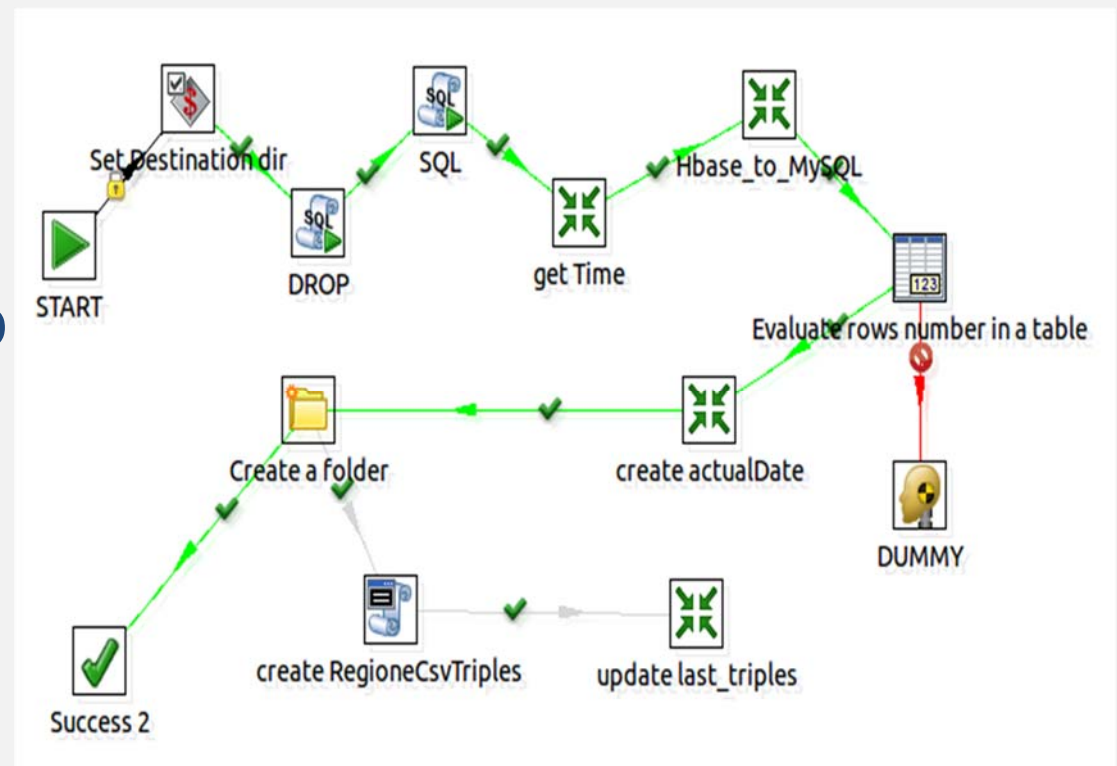
# Triplification phase (3)





# 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 then stored in file in n3 format.
- All the information characterizing the process are stored into a specific MySQL table (*process manager 2*).





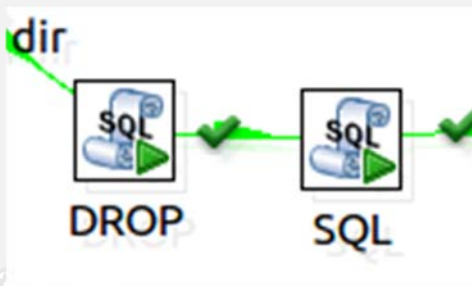
# Phase 1: Set storage root folder

- **PDI step: Set Destination dir**
- Set the variable that contains 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. The MySQL table is first deleted and then recreated if it already exists.



Job entry name:

Connection:

SQL from file: ☐

SQL filename:

Send SQL as single statement? ☐

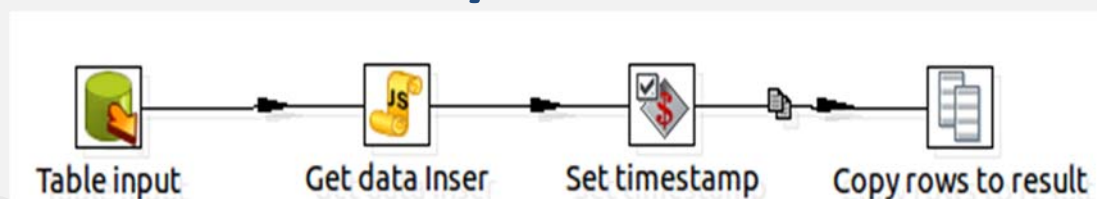
Use variable substitution? ☒

SQL Script:

```
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,
  ...
)
```

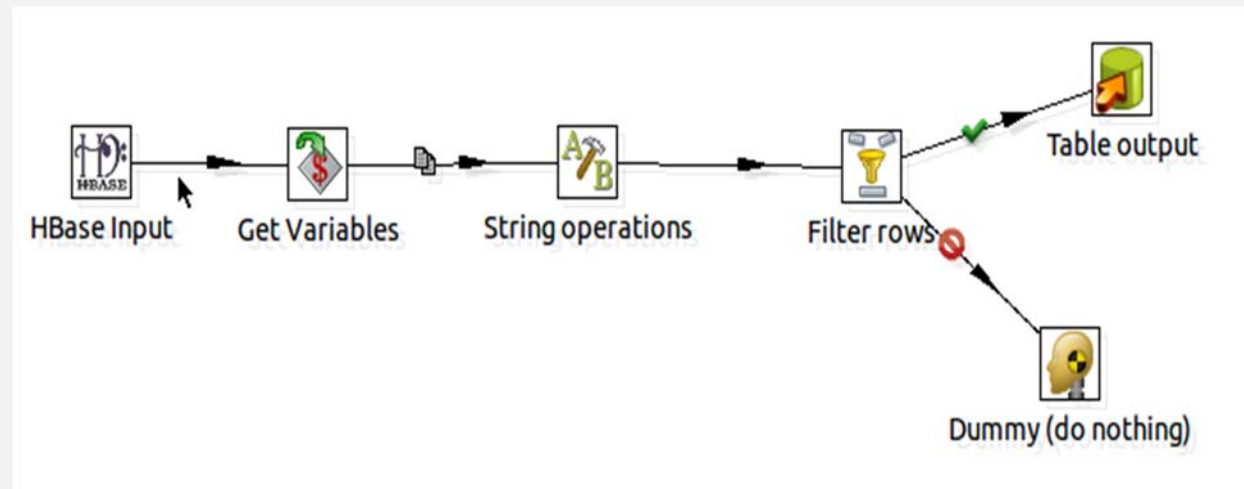
## Phase 3: Get last triple field

- **PDI steps: Table input, Modified Java Script value, Set variables**
- This transformation is invoked to get the last generation date of the triples from MySQL database and calculates the current date. From this date the variables to build the remaining part of path in which the triples file generated will be stored. The path must be **/Triples/Categoria/NomeProcesso/Anno\_mese/Giorno/Ora/MinutiSecondi/NomeProcesso.n3** .



# 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 the HBase database, filtering on the basis on process name and last triple field (retrieved previously). These data are then stored in MySQL table created previously.



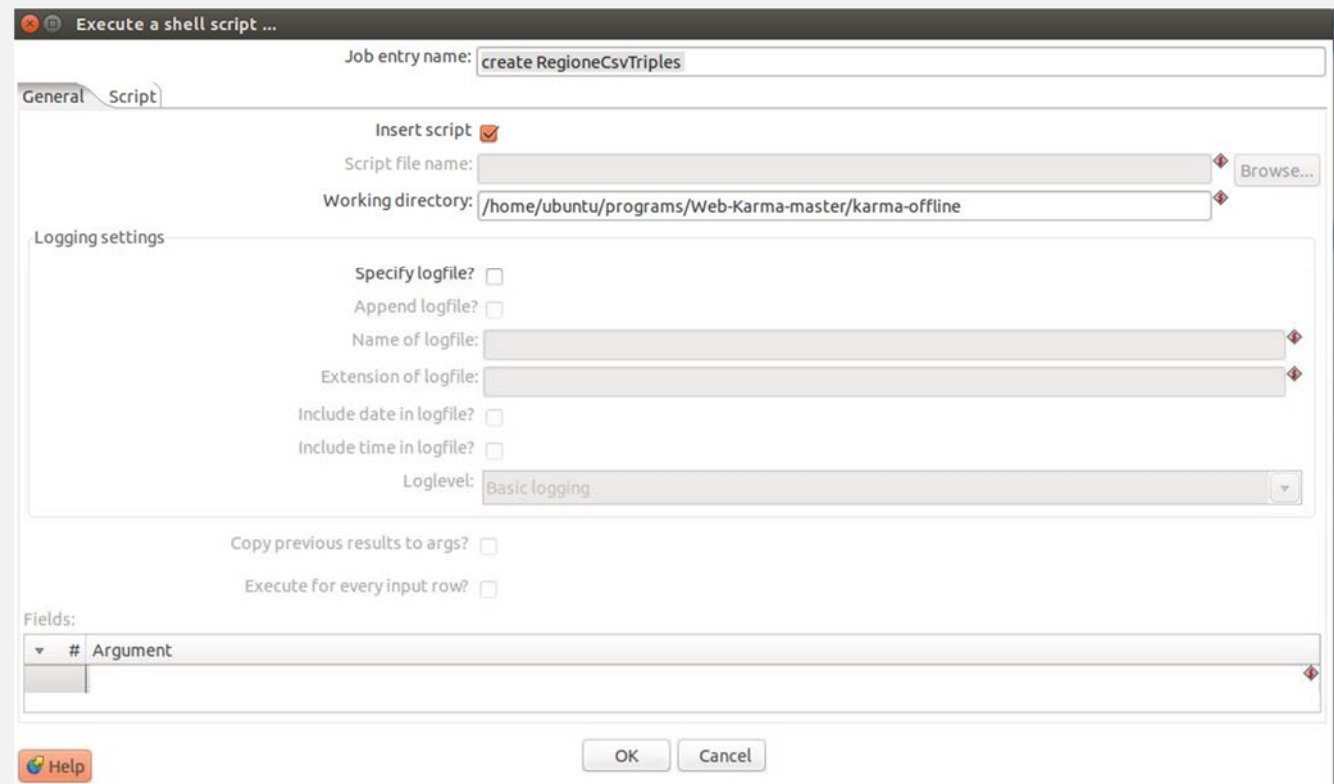
# Phase 5: Creating storage folder

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



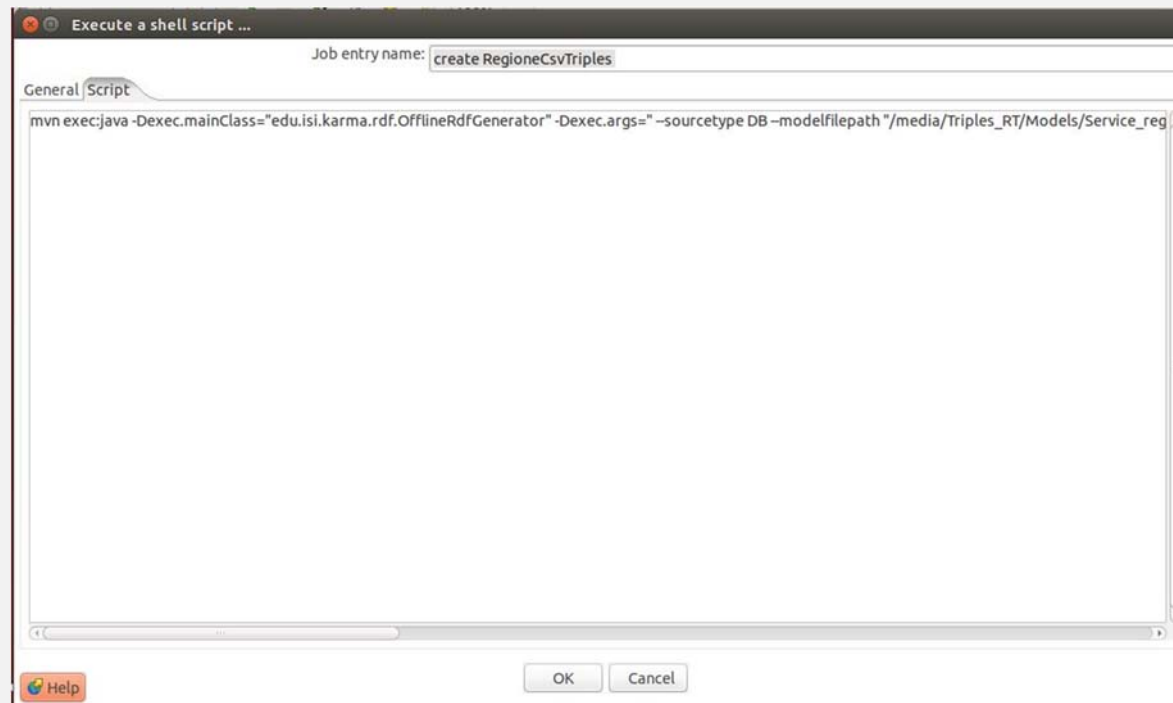
# Phase 6: RDF Triples generation

- PDI step: Execute a shell script Step

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 in the *Script* tab the specific command to create the RDF triples



# 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 MySQL database;
- the mapping model must be created previously with Karma.
- In output you specify the file name (.n3) where the RDF triples will be stored.

# Reference

**LINK PAGINA WEB**

**<http://www.disit.org/6690>**



# Smart City: data ingestion and mining

Corso del DISIT lab

**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>

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