Manuale Utente

December 21, 2017

Abstract

In questo manuale verranno esposte tecniche e linee guida per la realizzazione di un buon processo di acquisizione dati per arricchire la Knowledge bade che sta alla base di diversi progetti nazionali (SiiMobility) e europei (Resolute, Replicate, Select4cities). Gli strumenti utilizzati sono: Penthao Spoon, Karma, Apache Phoenix, database MySQL e Hadoop HBase. L'obiettivo principale è quello di esporre un esempio guidato. L'utilizzatore è esortato a seguire (dove possibile) le indicazioni fornite e ad interagire con i tutor per qualsiasi dubbio o domanda. Gli stessi tutor saranno responsabili della revisione e della validazione dei processi realizzati dall' utente.

Contatti

Ing. Michela Paolucci: michela.paolucci@unifi.it
Dr. Marco Faustino: marco.faustino@stud.unifi.it
Dr. Gabriele Xavier Giannini: garbiele.giannini2@stud.unifi.it
Dr. Federico Bambagioni: federico.bambagioni@stud.unifi.it

1 Introduzione: processi ETL (Extact, Transform, Load)

Il processo di acquisizione dati si articola in più fasi. Una fase preliminare prevede lo studio del dataset assegnato. È necessario **conoscere** il dato per procedere al meglio alle fasi successive. Generalmente si individuano due tipologie di dataset:

- ***statico***: si tratta di informazioni relative al tipo di servizio individuato, che presumibilmente non variano nel tempo o che comunque hanno delle variazioni sporadiche. Ad esempio la posizione geografica di un edificio, l'identificativo, l'indirizzo, etc.
- ***real-time*** *(o periodico/dinamico)*: si tratta di informazioni variabili nel tempo (in brevi intervalli) che richiedono dunque una frequenza di acquisizione più alta e hanno solitamente un corrispettivo statico

Esempio: Dataset che comprende informazioni sullo stato del flusso del traffico segnalato da vari sensori che si trovano nella città di Firenze.

In questo caso si hanno sia dati statici che dati dinamici:

- dati statici: posizione del sensore (coordinate) e relative specifiche (id, tipo di misurazioni, etc.)
- dato dinamico (ciò che cambia nel tempo): numero di macchine/moto/etc. che passano in quel tratto di strada, etc.

Si noti che il dato dinamico in questo caso è collegato a quello statico: è necessario sapere quale sensore di Firenze ha misurato il passaggio di N macchine in un dato tempo T.

L'elaborazione di queste due tipologie di dataset richiede procedure di tipo diverso. È necessario creare un ETL per i dati statici e un ETL differente per acquisire ed elaborare i dati dinamici:

- Ogni ETL prevede l'inserimento finale dei dati su tabelle HBase, con il supporto di informazioni che vengono manutenute in tabelle MySQL
- È necessario, come già anticipato, mantenere la connessione tra dati statici e relativi dati dinamici (se presenti). Il collegamento avviene tramite la presenza di una colonna comune alle tabelle HBase di uscita dei due ETL (statico e dinamico) che contiene l'identificativo (di un sensore, di un ospedale, altro). Tramite questa connessione sarà possibile collegare i dati dinamici

all'informazione statica. In Figura 1 si osserva un esempio.

stationID	longitude	latitude	COD_TOP	city	country-name	street	civicNumber
eCharging_15EP22T2AA1S000066	11.220752	43.793507	RT04801706780TO	FIRENZE	FI	PIAZZA DELL'ELBA	4

Figura 1a): riga di una tabella HBase relativa a dati statici

```
eCharging_15EP22T2AA1S000066                    column=0:STATIONSTATE, timestamp=1510938384371, value=ACTIVE
```

Figura 1b): Esempio di tabelle HBase di un dato dinamico,
collegato al dato statico di 1a)

Possono esistere anche:

- dataset SOLO statici: punti di interesse (POI, Point of Interest) di una città (chiese, musei, etc.)
- dataset SOLO dinamici: segnalazione di terremoti (sono real time e l'epicentro del terremoto è sempre diverso)

2.1 Input e output di un ETL

Gli ETL sono processi di elaborazione dei dati, partono quindi dall'ingestion (Extraction) per poi proseguire con la rielaborazione del dato che comprende fasi di qualificazione e aggregazione dei dati (Transforming) e il caricamento del dato su store (Load).

Si è scelto di memorizzare:

- il dato finale:
  - in uno store NoSQL (HBase)
  - e in triple rdf (per aggregare i dati tra sé e connetterli alla multi-ontologia KM4City)
- i parametri di ingresso degli ETL in una tabella MySQL. Ad esempio: indirizzo del web server da cui scaricare il dataaset, licenza d'uso associata al dataset scaricato, descrizione del dataset e/o del data provider, directory in cui saranno salvate le triple rdf (file.n3), categoria semantica di KM4City associata al dato (classe), etc. La tabella è la 'process_manager2', ed è uguale per tutti gli ETL. Le informazioni che devono essere inserite in questa tabella sono spiegate nel dettaglio nel paragrafo 4.

2 Uso della macchina virtuale degli strumenti per gestire gli ETL

Accesso alla macchina virtuale
User: ubuntu
Password: ubuntu

Strumenti da usare e relativi comandi:

- Penthao Kettle (in particolare si usa il tool grafico Spoon)
  - *Link utili:*
    - www.pentaho.com/
    - https://wiki.pentaho.com/display/EAI/Pentaho+Data+Integration+Steps
    -
  - *Versione:* Pentaho Data Integration (PDI) ver. 7.0
  - *Comandi*:
    - Per avviare Spoon, lanciare il comando seguente da una qualsiasi directory della macchina virtuale: spoon.sh
- Mysql:
  - Link utili:
    - http://www.mysql.com

- Comandi:
- Usare browser: PhpmyAdmin interface http://127.0.0.1/phpmyadmin/ con:
  - Username: testuser
  - Password: testpw

- HBase:
  - *Link utili:*
    - http://hbase.apache.org/
  - *Versione*: Apache HBase ver. 1.2.5 (Database NoSQL), in uso come stand alone
  - *Comandi:*
    - Per lanciare HBase (da qualsiasi directory): "start-hbase.sh"
    - Per arrestare HBase (da qualsiasi directory): "stop-hbase.sh"
    - Per verificare l'esecuzione: “jps”
    - Per verificare l'esecuzione da interfaccia web, accedere alla pagina web: http://localhost:16010/master.jsp
- Phoenix,
  - *Link utili*:
    - https://phoenix.apache.org/language/datatypes.html
    - http://www.hadooptpoint.org/filters-in-hbase-shell/
    - https://forcedotcom.github.io/phoenix/
  - Comandi:
    - Per creare una tabella tramite Phoenix da riga di comando:
      - andare nella directory seguente: "cd /home/ubuntu/Desktop/Utils/ create_table.sql"
      - lanciare il comando per aprire il file "nano create_table.sql"
      - lanciare il comando per eseguire la query contenuta nel file: "psql.py localhost create_table.sql"
    - Per gestire le tabelle da riga di comando: (comando per mettere la FAMILY????)
      - sqlline.py 127.0.0.1:2181:/hbase
      - !tables
      - drop TABLE test_hbase;
      - select serviceID from Electric_vehicle_charging;
- Karma data integration:
  - *Link utili*:
    - http://usc-isi-i2.github.io/karma/
  - *Versione*: 2.024
  - *Comandi*:
    - Per avviare Karma: lanciare "mvn -Djetty.port=9999 jetty:run" dalla cartella "cd programs/Web-Karma-master/karma-web"
  - *Accesso tramite web*:

        - http://localhost:9999

3 Regole generali di Nomenclatura

Ogni ETL ha una serie di regole di Nomenclatura da rispettare. Nel caso più complesso, in cui un dataset abbia sia la parte statica che quella realtime (o periodica/dinamica), le regole sono le seguenti:

- Cartella principale: nomeDataset_formatoOrigineDati
    - Cartella **'Static'** (contiene l'ETL relativo ai dati statici)
        - Cartella **'Ingestion'** (obbligatoria)
            - **'Main.kjb'**
            - N_files.ktr (un numero non definito di file ktr ovvero di trasformazioni di Pentaho)
            - M_files.kjb (un numero non definito di file kjb, ovvero di job di Pentaho)
        - Cartella **'QualityImprovement'** (opzionale)
            - **Main.kjb**
            - N_files.ktr (un numero non definito di file ktr ovvero di trasformazioni di Pentaho)
            - M_files.kjb (un numero non definito di file kjb, ovvero di job di Pentaho)
        - Cartella **'Triplification'** (obbligatoria)
            - **Main.kjb**
            - N_files.ktr (un numero non definito di file ktr ovvero di trasformazioni di Pentaho)
            - M_files.kjb (un numero non definito di file kjb, ovvero di job di Pentaho)
    - Cartella **'Realtime'** (contiene l'ETL relativo ai dati realtime o periodici)
        - Cartella **'Ingestion'** (obbligatoria)
            - **Main.kjb**
            - N_files.ktr (un numero non definito di file ktr ovvero di trasformazioni di Pentaho)
            - M_files.kjb (un numero non definito di file kjb, ovvero di job di Pentaho)
        - Cartella **'Triplification'** (opzionale**, serve SOLO in casi particolari, se richiesto**)

Esempio:

- PisaBikeSharing_kmz
    - Static
        - Ingestion
            - Main.kjb
            - getAPI.ktr
            - Ingestion.kjb
            - Database.ktr

        - …
    - QualityImprovement
        - Main.kjb
        - Qi.ktr
        - ….
    - Triplification
        - Main.kjb
        - …
- Realtime
    - Ingestion

4 Acquisizione Dati Statici

Il processo di acquisizione per questo tipo di dati comprende tre fasi:

- *Ingestion*: il dato viene scaricato ad esempio da un web server o da un file e caricato su HBase.
    - Si noti che è necessario seguire delle regole di nomenclatura per memorizzare il dato prima nel file system e poi su HBase. La memorizzazione su file system serve solo come controllo ulteriore, quella su HBase consente la vera e propria gestione dei dati.
    - *Esempio di memorizzazione dei dataset (così come sono messi a disposizione dai providers) nel file system, ovvero il percorso:*
        - Sources/Servizi/PisaBikeSharing_kmz/2017/12/19/15/3938 (per un dettaglio maggiore si vedano i capitoli 5 e 6)
    - *Esempio di nome di Tabella in uscita su HBase*: "**PisaBikeSharing**" con family 'Family1'
- *Quality Improvement*: si effettuano modifiche sul dato (in ingresso i dati appena inseriti nella tabella HBase e in uscita una nuova tabella HBase con i dati qualificati, ad esempio le strade tutte con caratteri maiuscoli aggiunte di coordinate se mancanti, correzione url/numerici civici/numeri di telefono o altro)
    - *Esempio di nome di Tabella HBase*: "**PisaBikeSharing_QI**" con family 'Family1'
- *Triplification*: si produce l'output sotto forma di triple in un file.n3, facendo riferimento alla ontologia km4city
    - *È necessario seguire delle regole di nomenclatura per memorizzare le triple su file system.*

ETL dati statici:

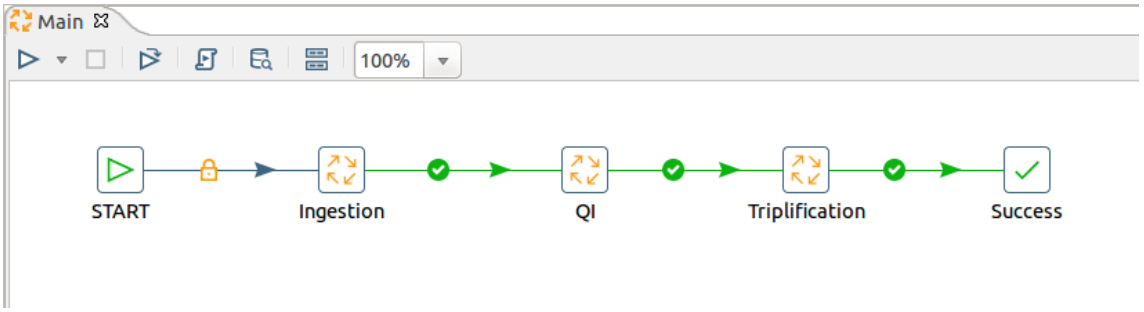

Figura 2: Pipeline del job Main.kjb

In Figura 2 è possibile osservare la struttura interna del file Main.kjb che corrisponde ad un job di Spoon. I job possono contenere al loro interno, altri job, trasformazioni o blocchi semplici. Ad esempio, il job dell'Ingestion contiene:

- Blocco di 'start' (per dare inizio al processo ETL)
- Tre trasformazioni (che corrispondono alle diverse fasi di manipolazione del dato e che sono descritte in dettaglio nei sotto paragrafi seguenti):
    - Ingestion
    - QI
    - Triplification
- Blocco 'Success' (per concludere il processo ETL)

Tale struttura è invariata per tutti gli ETL e dovrà essere mantenuta rigorosamente (questo facilita anche il riuso degli ETL).

La prima operazione da eseguire PRIMA di poter iniziare a lavorare sull'ETL, consiste nel definire i dati di input, ovvero nell'aggiornare la tabella MySQL chiamata '***process_manager2'*** del database *Elaborato_Sis_Distr*. Tale tabella contiene una riga per ogni ETL.

process	Resource	Category	Format	Access	Real_time	Source	param	last_update
Electric_vehicle_charging_kmz_ST	Colonnine per la ricarica dei veicoli elettrici	Servizi	kmz	HTTP	no	Opendata comune Firenze - http://opendata.c...	http://datigis.comune.fi.it/kml/Colonnin...	2017-09-29T17:01:51.000+02:00

Figura 3: Una riga estratta dalla tabella *Elaborato_Sis_Distr.process_manager2*

Come è possibile osservare in Figura 3, la chiave della tabella è la colonna '*process'*. Il valore assegnato a tale colonna non è casuale, deve esporre alcune informazioni del processo stesso quali: un identificativo del dato, il formato del file e il tipo del processo. In questo esempio vediamo "*Electric_vehicle_charging_kmz_ST"*.

Altri campi di fondamentale importanza sono:

- *Resource*: breve descrizione del dato
- *Category*: directory principale (dove verrà scaricato il dato e dove sarà salvato il file.n3, Servizi, TPL, etc.)
- *Format*: formato del file (kmz, csv, xml, json, etc.)
- *Access*: protocollo mediante il quale si ottiene il file (es: ftp, http, etc.)
- *Real_time*: tipo del processo
- *Source*: sorgente del file
- *Param*: percorso al file (es: http://dati.toscana.it/dataset/rt-oraritb/resource/ee55333c-fe53-4599-981d-389b13f28bb1)

Per quanto riguarda il job "*Main.kjb"* è necessario settare un parametro *processName* all'interno di esso, parametro che è impostato come parametro di input. Il suo valore dovrà essere identico a quello della chiave della riga inserita nel database (tabella 'process_manager2', colonna process). Vedi Figura 4.

Figura 4: Proprietà (dati di input) del job Main.kjb

Si procede con la descrizione delle singole trasformazioni che compongono l'ETL statico: **Ingestion, QI, triplification.**

6.1 Ingestion

In questa trasformazione si procede ne modo seguente: i) acquisizione del dataset dal provider (ad esempio scaricando un file dal web, tramite ftp, facendo crawling, etc.) e copia di esso in in file che viene memorizzato nel file system; ii) lettura dei dati dal file appena memorizzato; iii) estrazione dei dati di rilievo; iv) memorizzazione dei dati estratti su una tabella HBase. In Figura 5 è possibile visualizzare la pipeline del job *Main_Ingestion.kjb*, che a sua volta contiene altri step (o blocchi), job, trasformazioni.

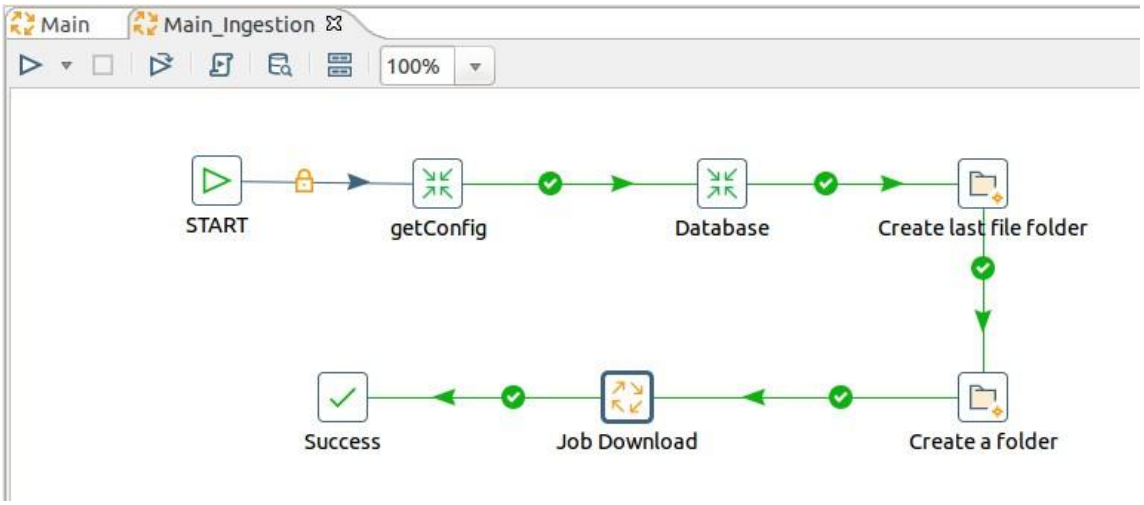

Figura 5: Pipeline del job *Main_Ingestion.kjb*

Lo step *getConfig* si limita a caricare un file di configurazione e a settare delle variabili globali. Tali variabili permettono di utilizzare percorsi relativi anziché assoluti. Si veda Figura 6. Il config.csv si trova nella cartella "/home/ubuntu/Desktop/Trasformazioni/".

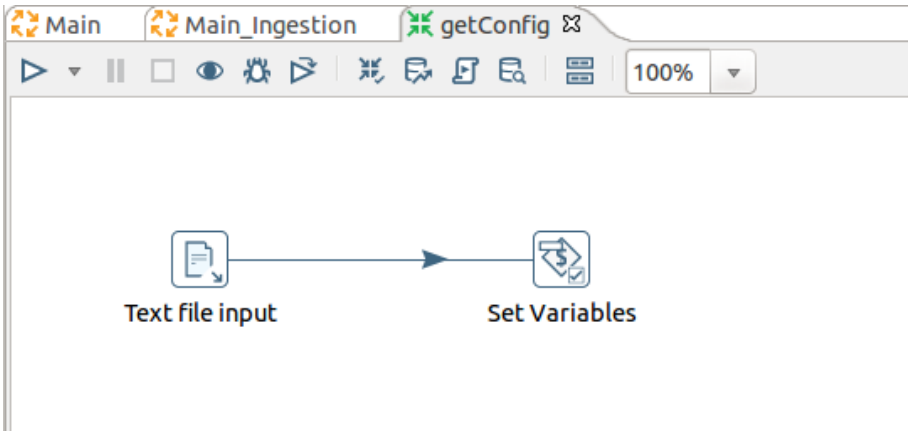

Figura 6: Pipeline della trasformazione *getConfig.ktr*

Lo step *Database* (sempre facendo riferimento alla figura. 5) è di tipo Transformation Executor e richiama la trasformazione Database.ktr che ha la struttura riportata in Figura 7.

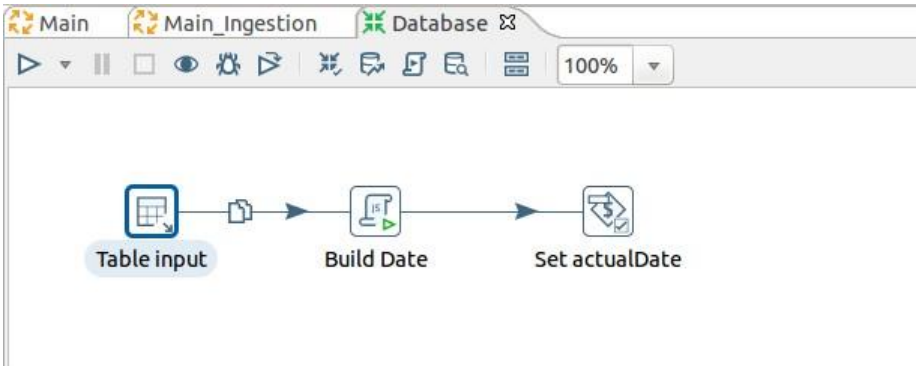

Figure 7: Pipeline della trasformazione *Database.ktr*

Lo step *Table input* (sempre facendo riferimento alla figura. 5), recupera dalla tabella MySQL *process_manager2* tutti i campi relativi al processo in esecuzione tramite una query SQL.

Lo step *Build Date* (di fig.5) è un blocco in cui è possibile scrivere del codice Javascript. Nello specifico a partire dalla data attuale, vengono ricavati i campi per costruire il path in cui memorizzare il file scaricato. Successivamente nello step *Set actualDate* i campi definiti in precedenza sono trasformati in variabili.

Nei passi *Create last file folder* e *Create a folder* (Vedi Figura 5) vengono create le cartelle che ospiteranno il file sorgente e la cartella '1LastFile', in cui sarà memorizzata una copia della versione più aggiornata del file scaricato.

Infine in Figura 8 si osserva il *Job Download*, (sempre a partire dalla fig.5).

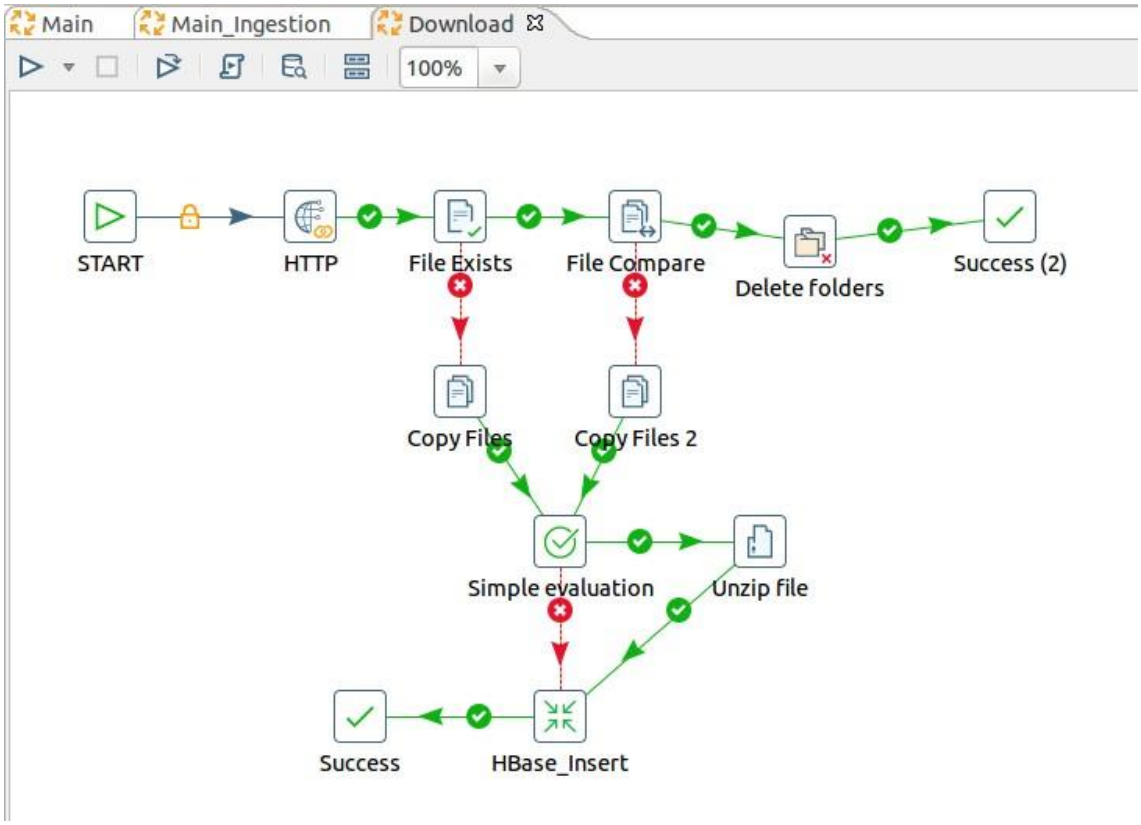

Figure 8: Pipeline del job *Download.kjb*

Nello step *HTTP* vengono impostati i campi: 'URL', come indirizzo da cui scaricare il file di interesse; 'Target', come percorso dove salvare tale file (ovviamente i percorsi sono relativi, ovvero vengono utilizzate le variabili settate in precedenza).
Dopodiché viene effettuato un controllo: se il file sorgente contenente i dati appena scaricati esiste già all'interno della cartella ***1LastFile,*** viene effettuato un confronto con quello appena scaricato (ovvero si confrontano i dati attuali con quelli scaricati in precedenza, nel caso in cui l'ETL sia già stato fatto partire). Se i due file sono identici, le cartelle appena create vengono eliminate e il job termina in *Success 2*, questo significa che i dati messi a disposizione dal Data provider NON sono variati e che siamo già in possesso dei dati aggiornati (quindi NON c'è bisogno di analizzarli nuovamente). In caso contrario la copia all'interno della cartella 1LastFile viene aggiornata con i dati nuovi. Nell'esempio si procede unzippando il file e inserendo il dato in una tabella *HBase*.
In Figura 9 si osserva la pipeline della trasformazione. Il dato in questo caso viene letto da un file.kml, viene "ripulito" e infine è inserito in HBase tramite lo step *HBase Output*. Per semplicità tale tabella dovrà essere chiamata *Electric_vehicle_charging_ST*.

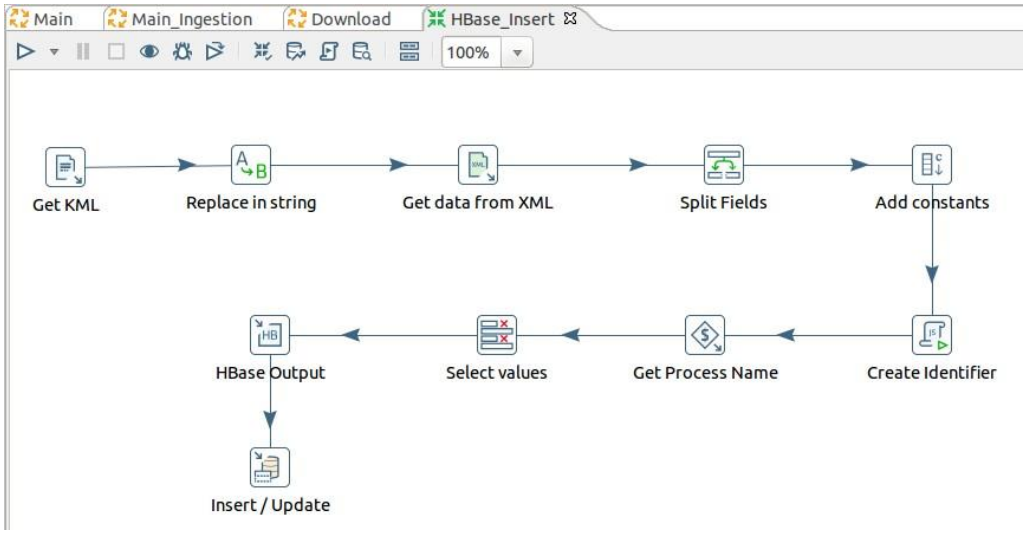

Figura 9: Pipeline della trasformazione *HBase_Insert.ktr*.

6.2 Quality Improvement

La fase di Quality Improvement ha come obiettivo quello di migliorare la qualità dei dati acquisiti nella fase di ingestion, operando opportune modifiche sullo specifico dato in esame e memorizzandolo in una nuova tabella HBase. La struttura del Job *Main_QI* è riportata in Figura 10.

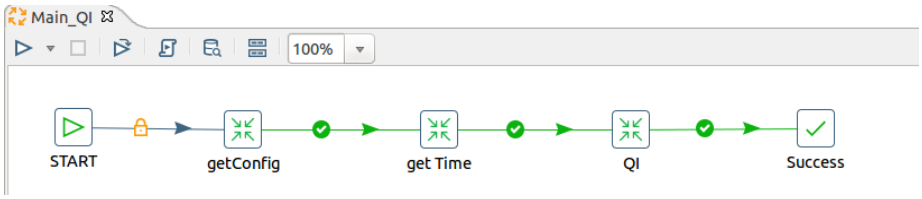

Figure 10: Pipeline del job *Main_QI.kjb*

La trasformazione *QI.ktr* ha la responsabilità di aggiungere fondamentali proprietà al dato. Tramite una query SPARQL vengono estratte informazioni come: coordinate (latitudine e longitudine), via, numero civico, città, cap, ma soprattutto il toponimo. Si veda Figura 11.

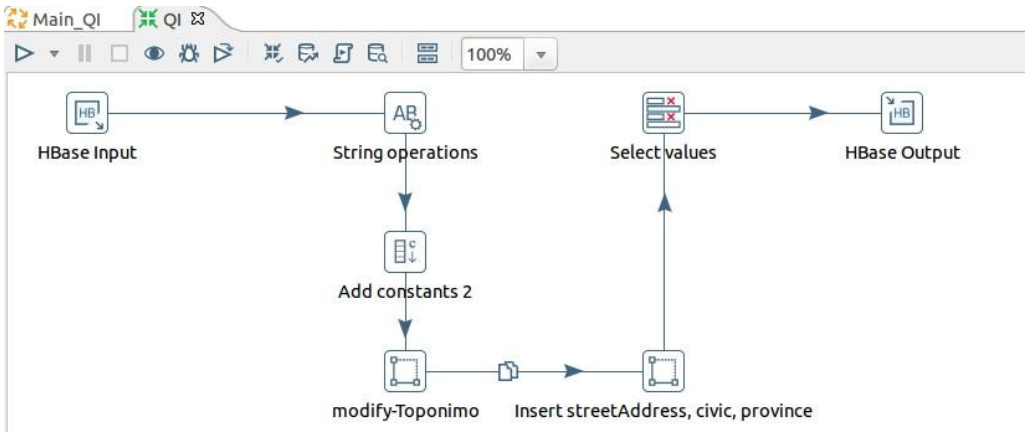

Figura 11: Pipeline della trasformazione *QI.ktr*

Il dato viene letto dalla tabella *Electric_vehicle_charging_ST* e viene completato con le informazioni mancanti. Attraverso lo step *Select Value* si effettua invece una pulizia del dataset ovvero si selezionano soltanto i campi che risultano rilevanti per il sistema. Quindi il dato è pronto per essere salvato su una nuova tabella HBase (es: *Electric_vehicle_charging_ST_QI*, sempre con family pari a 'Family1').

6.3 Triplification

La fase di Triplification ha l'obiettivo di generare un file.n3 contenente un insieme di triple RDF partendo dai dati acquisiti e processati nelle due fasi precedenti. Ciascuna informazione collezionata viene collegata alla ontologia KM4City, tramite il tool Karma.
Karma prevede:

- La creazione di un modello che faccia il mapping dei tipi di dati analizzati su una o più ontologie (nel nostro caso nella multi-ontologia KM4City): il mapping si effettua incrociando le ontologie (in forma di file contenenti le triple) con i dati contenuti in una tabella di un database MySQL (quindi per generare il modello Karma è necessario trasferire temporaneamente i dati da HBase a MySQL).
- L'applicazione del modello ai dati tramite il tool kitchen

La struttura di questo processo ETL è rappresentata in Figura 12.

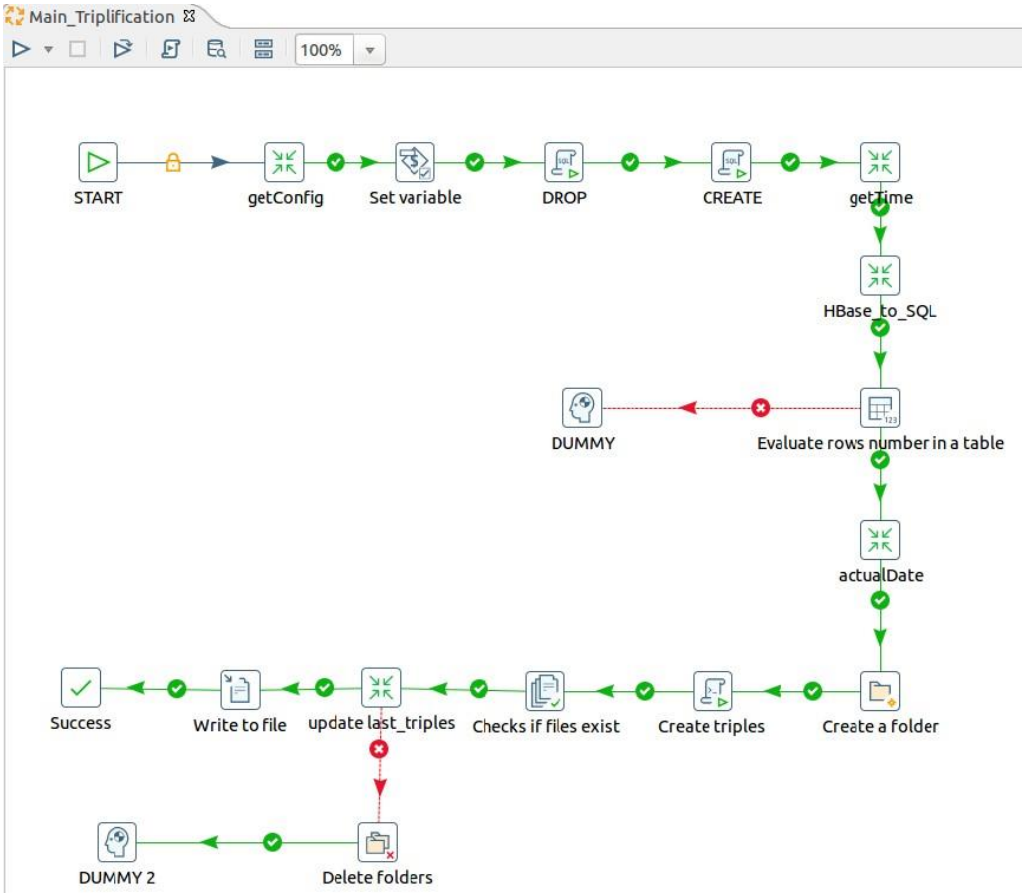

Figura 12: Pipeline del job *Main_Triplification.kjb*.

Lo step *Set variable* crea la variabile *MODELPATH* che mantiene il percorso al modello. Il modello dovrà essere salvato in una cartella chiamata *Model* all'interno della cartella *Triplification*. Vedi Figura 13.

	Variable name	Value
1	MODELPATH	/home/ubuntu/Desktop/Trasformazioni/Electric_vehicle_charging_kmz/Static/Triplification/Model

Figura 13: *Set variable*

Negli step successivi l'informazione contenuta nella tabella HBase *Electric_vehicle_charging_ST_QI*, viene copiata temporaneamente in un'altra tabella MySQL *Electric_vehicle_charging_kmz_ST*. Questa operazione è necessaria perché per realizzare le triple rdf aderenti alla ontologia KM4City, si è scelto di usare il tool Karma. Lo step Create a folder crea la cartella che conterrà il file.n3 di triple. A livello di nomenclatura è necessario seguire le seguenti linee guida:

- la cartella contenente il file.n3 avrà come nome il solito del processo e come cartella 'antenato' la categoria del processo contenuta nella colonna 'Category' della tabella MySQL 'process_manager2' e in mezzo le cartelle che tengono nota della data in cui sono state fatte le triple (Categoria/anno/mese/giorno/ora/secondi/Nomeprocesso).

- Esempio: TPL/Bus_ataflinea/2017/12/19/17/0001

Lo step *Create triples* realizza le triple come si evince dal nome mentre i successivi effettuano alcuni controlli e validazioni.

5 Acquisizione dati RealTime (periodici o dinamici)

Il processo di acquisizione comprende una unica fase durante la quale il dato viene scaricato, processato e inserito in una tabella HBase. In Figura 14 è possibile osservare la struttura interna del file Main.kjb, che dovrà essere mantenuta rigorosamente.

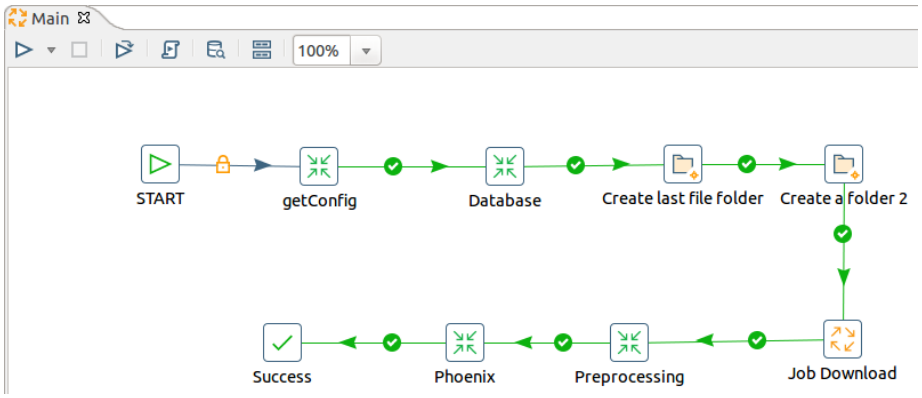

Figura 14: Pipeline del job Main.kjb

Come nel caso statico, la prima operazione da eseguire consiste nell'aggiornare la tabella MySQL '*process _manager2'* del database *Elaborato_Sis_Distr*. Vedi Figura 15.

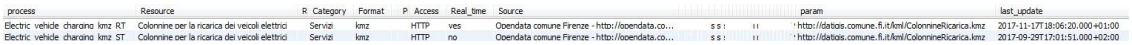

Figura 15: Alcune righe estratte dalla tabella *Elaborato_Sis_Distr.process_manager2*

Anche per questo ETL, è necessario settare il parametro *processName* all'interno del job. Il suo valore dovrà essere identico a quello della chiave della riga inserita nel database ovvero *Electric_vehicle_charging_kmz_RT*. I passi successivi sono pressoché identici e possiedono le stesse responsabilità. In Figura 16 è riportata la struttura del job *Download.kjb*.

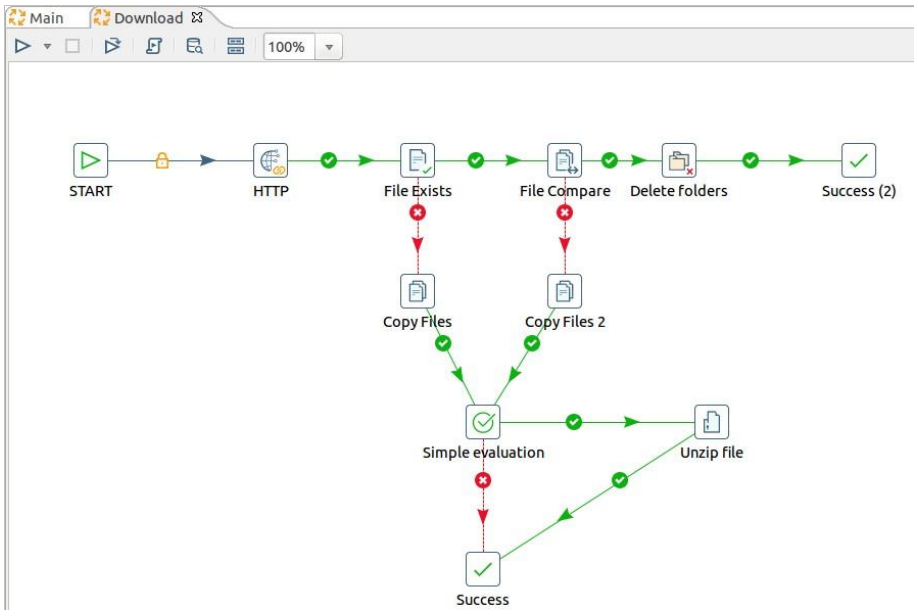

Figura 16: Pipeline del job *Download.kjb*

Lo step *Preprocessing* in questo esempio ha soltanto il compito di selezionare l'informazione RealTime e di settare un identificativo per ogni riga del dataset. Tale identificativo è identico a quello utilizzato durante l'acquisizione della parte statica. **Questo campo è fondamentale perché collega il dato statico e quello**

RealTime. Si veda Figura 17: l'output è copiato nello stream.

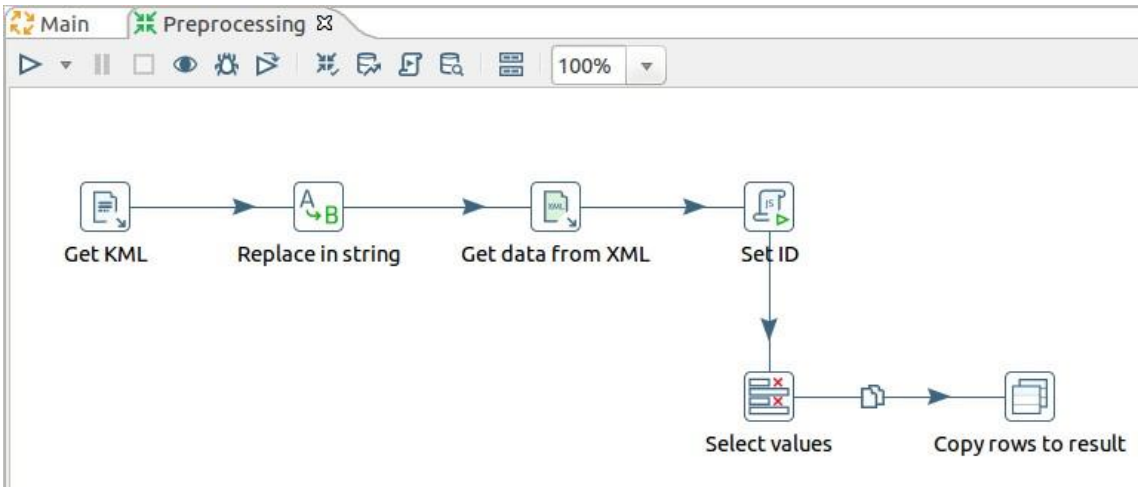

Figura 17: Pipeline della trasformazione *Preprocessing.ktr*

Dallo stream il dato viene letto e attraverso lo step *Phoenix insert* viene inserito (mediante la funzione upsert) in una tabella HBase. Tale tabella deve essere creata da riga di comando. In figura 19 si può osservare un esempio.

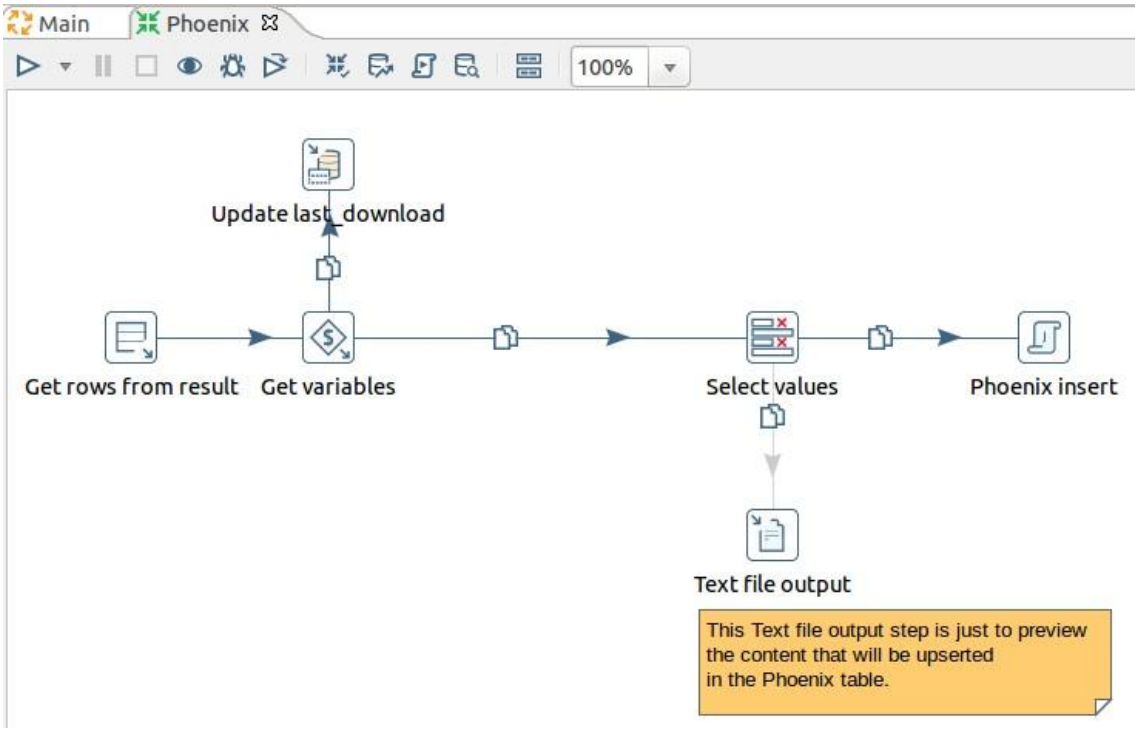

Figura 18: Pipeline della trasformazione *phoenix.ktr*

```
CREATE TABLE if not exists Electric_vehicle_charging (
serviceID       VARCHAR,
actualDate      Date,
stationState    VARCHAR,
chargingState   VARCHAR,
constraint pk primary key (serviceID) )
```

Figure 19: Creare la tabella su Hbase tramite Phoenix