Distributed Systems and Internet Technologies Lab
Distributed Data Intelligence and Technologies Lab
Department of Information Engineering (DINFO)
University of Florence
http://www.disit.dinfo.unifi.it

Programmazione ETL per data warehouse

Version 1.0, data 19/04/2015

http://www.disit.org

Lo scopo di questa guida e' quello di insegnare a realizzare / programmare processi ETL (Extraction - Transformation - Loading) per il data warehouse. http://en.wikipedia.org/wiki/Data_warehouse. Il contesto in cui viene introdotta questa tecnologia e' quello delle Smart City, pertanto tutti gli esempi saranno collegati alla catena di processo del data warehouse per le Smart City dove la parte di database e' un noSQL RDF store. La descrizione generale la potete trovare:

- **Slide 2014-2015 Programmazione ETL per DataWarehouse (Parte 8)**: from open data to triples, OD 2 RDF, OD and PD, static and Dynamic OD, Problemi architetturali, programmazione ETL, esempi concreti, massive data mining and crawling, quality improvement, geolocalization, triplification, reasoning and rendering, example of km4city data ingestion.
- P. Bellini, M. Benigni, R. Billero, P. Nesi and N. Rauch, "**Km4City Ontology Bulding vs Data Harvesting and Cleaning for Smart-city Services**", International Journal of Visual Language and Computing, Elsevier, http://dx.doi.org/10.1016/j.jvlc.2014.10.023 http://www.disit.org/6573
- Link alla pagina web di riferimento: http://www.disit.org/6690

Guida alla realizzazione di un Processo ETL

Questi realizzino l'acquisizione di nuovi OpenData da diverse sorgenti e li trasformino in Triple RDF, per poterli integrare all'interno di un sistema già esistente. A tal proposito questi processi ETL devono basarsi sull'architettura illustrata in Figura 1 che si compone di tre specifiche fasi:

- Fase 1 -> ***Data Ingestion***
- Fase 2 -> ***Quality Improvement***
- Fase 3 -> ***Triplification***

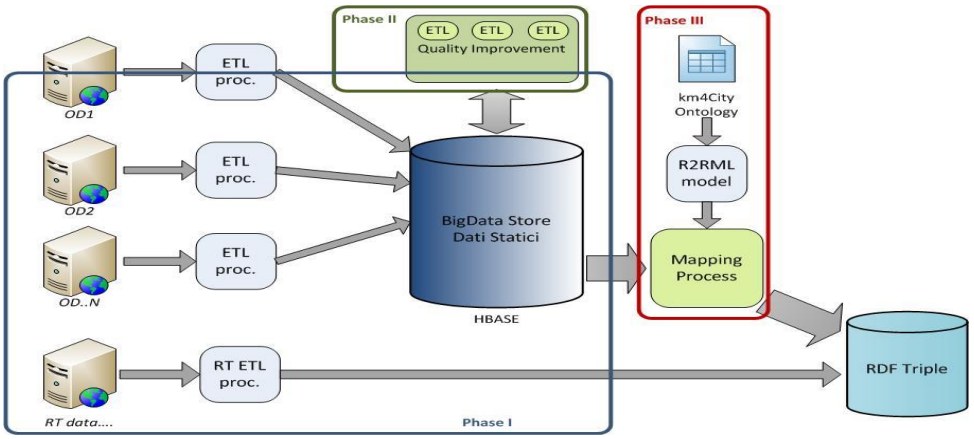

Figura 1- Architettura della fasi di un processo ETL

Ad ogni dataset da elaborare è associato uno specifico processo. I dati di input, quelli di output e le trasformazioni che realizzano le varie elaborazioni devono essere organizzati secondo la seguente struttura:

- Input **-> File sorgenti**
  /Sources/Categoria/NomeProcesso/Anno_mese/Giorno/Ora/MinutiSecondi/file.xxx

- Elaborazione **-> Processo ETL**
  /Trasformazioni/NomeTrasformazione/Ingestion/...
  /Trasformazioni/NomeTrasformazione/QualityImprovement/...
  /Trasformazioni/NomeTrasformazione/Triplification/...

- Output **-> Triple**
  /Triples/Categoria/NomeProcesso/Anno_mese/Giorno/Ora/MinutiSecondi/file.n3

Una volta scaricato uno specifico dataset (fase di Ingestion) è opportuno realizzare un'analisi preliminare con l'obiettivo di identificare i campi di interesse, le differenti forme in cui essi si presentano e se necessitano di operazioni che ne migliorino la qualità (fase di QI). A tal proposito può essere utile utilizzare *datacleaner* (http://datacleaner.org/), uno strumento di supporto per il profiling dei dati e l'individuazione di particolari pattern.

Esempio A

Di seguito viene descritto in modo dettagliato il processo ETL inerente l'acquisizione del dataset "*Strutture_ricettive*" disponibile sul sito dati.toscana.it in formato csv

(http://dati.toscana.it/dataset/ceb33e9c-7c80-478a-a3be-2f3700a64906/resource/5e8ec560-cbe6-4630-b191-e274218c183c/download/strutturericettive20141012.csv).

1. Data Ingestion

La fase di data ingestion prevede l'acquisizione del file sorgente dal portale Open Data della regione toscana, la sua memorizzazione in una specifica cartella e il caricamento dei dati sulla relativa tabella HBase. Il punto di partenza di questo processo è il Job ***Main*** la cui struttura è riportata in Figura 2.

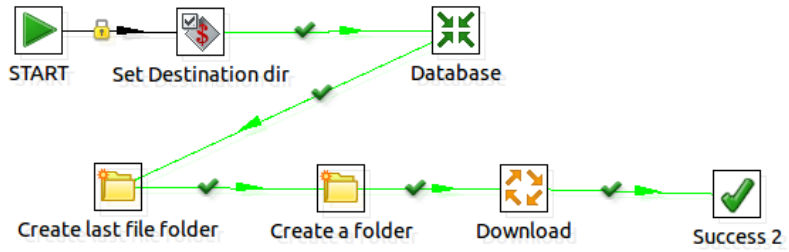

Figura 2 - Job Main (fase Data Ingestion)

Nelle impostazioni generali del Job deve essere definito il parametro *processName,* il cui valore verrà specificato al momento dell'esecuzione dell'intero processo.

Lo step *Set Destination dir* è un blocco di tipo *Set variables* in cui viene settata la variabile **DestinationDir** che indica la radice del percorso il cui sarà memorizzato il file sorgente; il valore di questa variabile dovrà essere **/Sources/Categoria.**

Lo step *Database* è di tipo *Transformation Executor* e richiama la trasformazione Database.ktr che ha la struttura riportata in Figura 3.

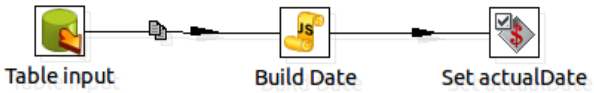

Figura 3 - Trasformazione Database (fase Data Ingestion)

Lo step *Table input* recupera dalla tabella MySQL process_manager2 tutti i campi relativi al processo in esecuzione tramite una query SQL.

Lo step *Build Date* è un blocco in cui è possibile scrivere del codice Javascript. Nello specifico in questo step, 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 e, all'uscita da questo step, 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.

Lo step *Download* è un blocco di tipo *Job* in cui è richiamato il job Download.kjb la cui struttura è riportata in Figura 4.

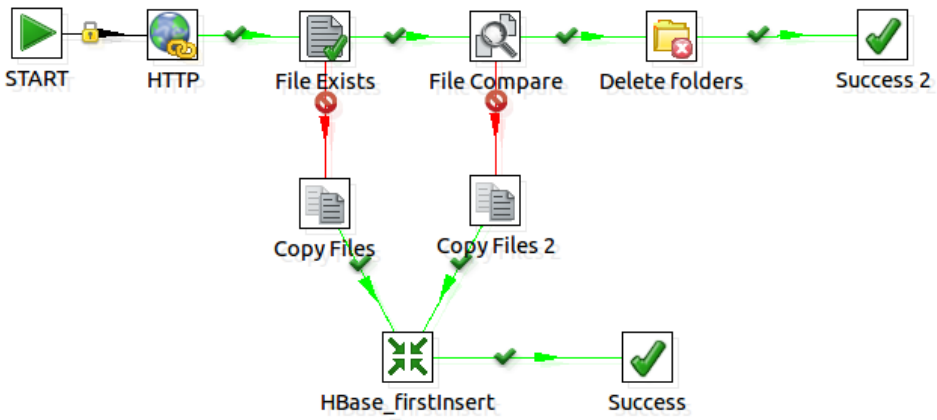

Figura 4 - Job Download (fase Data Ingestion)

Nello step *HTTP* vengono definiti l'**URL** da cui scaricare il file di interesse e il **Target file** in cui viene specificato il nome del file e il path in cui memorizzarlo. Successivamente viene fatto un controllo sull'esistenza del file sorgente all'interno della cartella 1LastFile e se presente viene fatto un confronto con il file appena scaricato. Se i due file sono identici le cartelle appena create vengono eliminate e il job termina (*Success 2*), altrimenti viene aggiornata la copia all'interno della cartella 1LastFile e si richiama la trasformazione *Servizi_csv_ING* nello step *HBase_firstInsert*.

La trasformazione *Servizi_csv_ING* ha la seguente struttura.

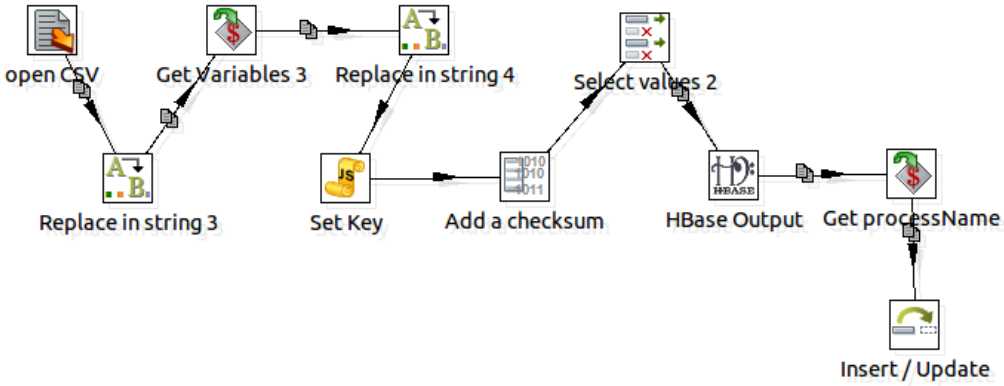

Figura 5 - Trasformazione Servizi_csv_ING (fase Data Ingestion)

Inizialmente nello step *open CSV,* di tipo *Text file input,* viene selezionato il file csv da caricare specificando nella tab Content il formato e il separatore da utilizzare, e nella tab Fields tutti i campi da importare. Nei vari step *Replace in string* viene modificato il contenuto di alcuni campi di tipo string (ad esempio si effettua la rimozione del carattere \ all'interno del campo name).

Nello step *Get Variables 3* vengono definiti i campi *process* - *actualDate* - *timestamp* a partire dalle rispettive variabili. Viene poi eseguito lo step *Set Key* (Javascript) in cui viene definito il campo ServiceCsvKey ottenuto come concatenazione di tutti i campi estratti dal file sorgente; questo campo viene poi cifrato in MD5 nello step *Add checksum* e rinominato FinalKey.

Dopo aver selezionato i campi di interesse, mediante lo step Select values 2, viene eseguito lo step HBase Output per la loro effettiva memorizzazione nel NoSQL DB HBase. Questo step prevede che vengano

impostati i parametri *Zookeeper host* (in locale 127.0.0.1) e la porta Zookeeper port (2181) nella tab Configure conection. Poichè HBase è un DB di tipo schema-less è necessario definire un mapping dei campi di input sulla specifica tabella in cui saranno memorizzati. Ciò si realizza nella tab Create/Edit mappings selezionando la tabella (creata in precedenza tramite h-rider o dalla shell di HBase) e i campi di interesse, e specificando un nome per il mapping creato.

Una volta recuperato il nome del processo (step *GetProcessName*), nello step *Insert/Update* viene aggiornato il campo *last_update* con la data di ultima modifica della riga nella tabella MySQL process_manager2 contenete le informazioni relative al processo in esame.

2. Quality Improvement

La fase di Quality Improvement ha come obiettivo quello di migliorare la qualità dei dati acquisiti nella fase di ingestion, operando delle opportune modifiche sullo specifico dato in esame e memorizzando il contenuto in una nuova tabella HBase. La struttura del Job iniziale, *Data_QI*, è riportata in Figura 6.

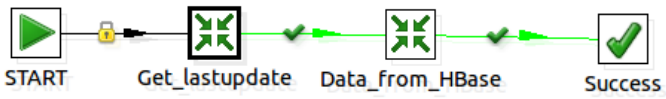

Figura 6 - Job Data_QI (fase QI)

Lo step *Get_lastupdate* richiama l'omonima trasformazione in cui, a partire dal *processName* passato come parametro, si recupera dalla tabella MySQL process_manager2 il valore dell'ultima data di aggiornamento (last_update) che viene poi trasformata in una variabile in modo da poter essere utilizzata anche nelle altre trasformazioni che compongono il job. La struttura di questa trasformazione è la seguente.

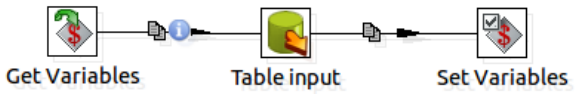

Figura 7 - Trasformazione Get_lastupdate (fase QI)

Successivamente all'interno dello step *Data_from_HBase* viene eseguita la trasformazione *Servizi_csv_QI.ktr* la cui struttura è riportata in Figura 8. In questa trasformazione per modificare i diversi campi si utilizza uno step particolare, il *Simple Mapping (sub-transformation)* , che consiste nel richiamare una sotto-trasformazione specificando ogni volta il mapping tra i campi del flusso in input/output e quelli definiti all'interno della sotto-trasformazione stessa. Si è deciso di adottare questa soluzione i un'ottica di riusabilità; in questo modo, infatti, è possibile adattare la stessa sotto-trasformazione ai diversi dataset da elaborare (es. Strutture_ricettive, Banche, Enogastronomia, Eventi...).

Lo step *HBase input* recupera i dati caricati in HBase nella tabella creata in fase di ingestion. I settaggi sono analoghi a quelli fatti in precedenza, non è necessario però creare un nuovo mapping ma è sufficiente utilizzare quello già esistente (legato alla specifica tabella). E' inoltre possibile operare un filtraggio sui dati da caricare utilizzando la tab *Filter result set*; nell'esempio in questione è stato applicato un filtraggio sulla

base nel nome del processo in esame. Successivamente nello step *Add costants* sono stati definiti due nuovi campi (*notes* e *categoryEng*) che non erano presenti nel file sorgente, ma che sarebbero stati utilizzati nel resto della trasformazione aumentando il contenuto informativo.

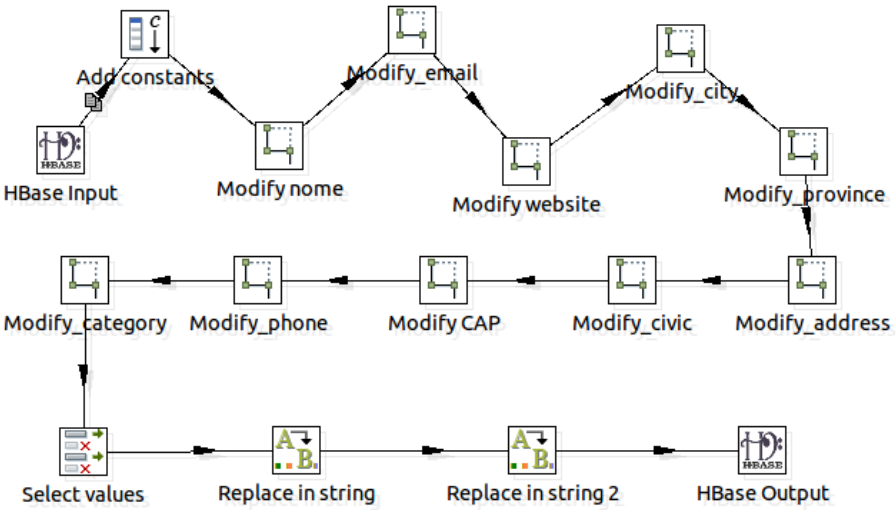

Figura 8 - Trasformazione Servizi_csv_QI (fase QI)

Le modifiche vere e proprie impattano i seguenti campi:

- Nome: Viene effettuato un trim sia in testa che in coda alla stringa.

- Email: Viene controllata la correttezza del formato dell'indirizzo/i email presenti. Le email non corrette o gli eventuali indirizzi aggiuntivi vengono spostati nel campo *notes*.

- Website: Tramite l'uso di espressioni regolari, viene controllato che la URL in ingresso sia conforme al pattern di un indirizzo web e se compare più di un indirizzo. I website non corretti o quelli aggiuntivi vengono spostati nel campo *notes.*

- City: Viene effettuato un trim preliminare sia in testa che in coda alla stringa, e successivamente vengono rimossi eventuali caratteri numerici o caratteri speciali (\ - _) che talvolta possono comparire all'interno del nome della campo city.

- Province: Inizialmente il campo *province* viene convertito tutto in UpperCase, e viene poi verificato che sia effettivamente composto da due soli caratteri. Se ciò non è vero si ricorre all'uso del campo *cityUpper* (campo city convertito in maiuscolo) e tramite lo step *Database lookup* si ricava la sigla della provincia dalla tabella MySQL *MappingCity* in cui sono è presente l'elenco di coppie Provincia-Comune della Regione Toscana.

- Address e Civic: La trasformazione relativa agli indirizzi è divisa in 2 parti. La prima parte (*Modify_address*) si occupa della divisione dell'indirizzo dal numero civico (eventualmente seguito da testo aggiuntivo); la seconda parte (Modify_civic) si occupa della separazione del numero civico "principale" (il primo) dagli altri numeri civici e dal testo aggiuntivo che vengono memorizzati nel campo *notes.*

- CAP: In questa trasformazione inizialmente si controlla se il campo CAP è presente o meno indirizzando a seconda dei casi, tramite lo step *Filter rows*, il flusso dei dati su due diverse diramazioni.

a) **CAP pieno** - Si verifica che il CAP presente sia composto da 5 cifre, e nel caso in cui ciò non fosse verificato viene resettato.
b) **CAP vuoto** (cioè non presente nel dataset in input) - In questo caso si utilizzano la tabella MySQL *MappingCity* e i campi del flusso *city e province* per recuperare il CAP corrispondente; se però la città di riferimento è Firenze - Pisa - Livorno allora si usa anche il campo address, poichè per queste città esiste più di un CAP.

- Phone: Inizialmente vengono rimossi i vari prefissi internazionali, poi nel caso in cui compaia più di un numero telefonico tutti quelli addizionali vengono separati dal primo e spostati nel campo notes.

- Category: Questa trasformazione a partire dal valore del campo *category* (se presente) utilizza lo step *Database lookup* per recuperare il valore dell'equivalente categoria in inglese (*CategoryEng*) dalla tabella MySQL *ServiceCategory*. Inoltre solo per la generica *category* ***Servizi*** il flusso della trasformazione viene deviato in modo tale da eseguire del codice Javascript (step *Modified Java Script Value*) che si occupa di determinare la specifica *CategoryEng* sfruttando il campo nome.

Alla fine di tutte le sotto-trasformazioni, i dati di interesse opportunamente selezionati e rinominati vengono caricati in una nuova tabella HBase (identificata dal suffisso QI) utilizzando lo step HBase Output come descritto in precedenza.

3. Triplification

La fase di Triplification ha l'obiettivo di generare un insieme di triple RDF partendo dai dati acquisiti e migliorati nelle due fasi precedenti e "mappandoli" su un modello costruito sulla base delle relazioni definite all'interno di una specifica ontologia di riferimento. La struttura di questo processo ETL è rappresentata in Figura 9.

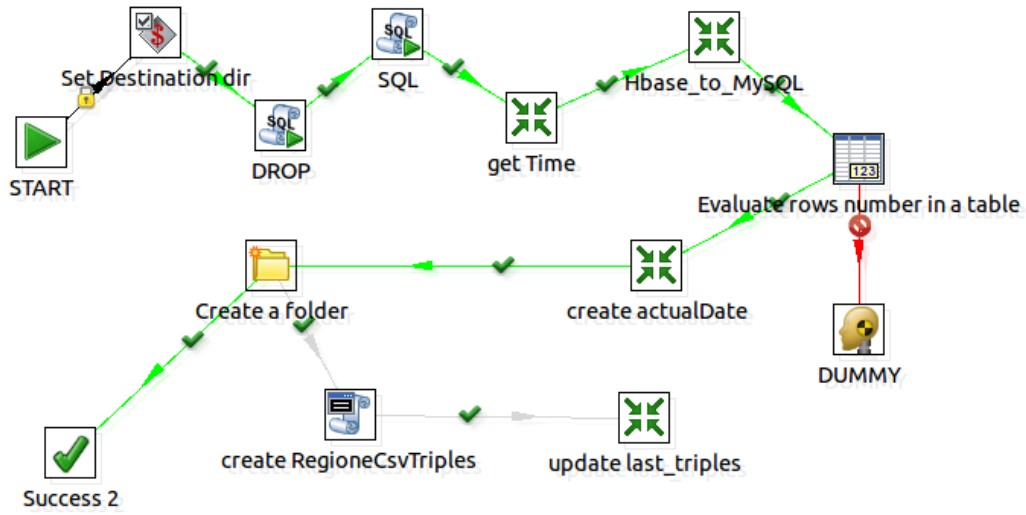

Figura 9 - Job Main_csv (fase Triplification)

Il Job *Main_csv* comincia con lo step Set Destination dir in cui viene settata la variabile relativa alla radice del percorso in cui saranno memorizzate le triple RDF generate.

Per la generazione delle triple (utilizzando il tool Karma) è necessario che i dati da mappare siano memorizzati all'interno di un classico database relazionale, a tal proposito è necessario copiare i dati presenti in HBase (tabella di QI) su una tabella MySQL temporanea. Attraverso lo step *Drop* si elemina l'eventuale tabella MySQL già presente e successivamente la si ricrea nello step SQL, definendo in modo opportuno i campi secondo il modello ontologico.

La trasformazione *getTime* richiamata all'interno dello step get Time è strutturata nel seguente modo.

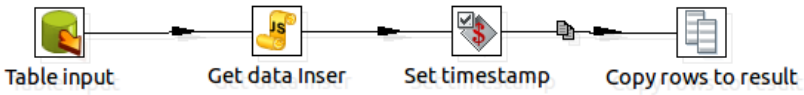

Figura 10 - Trasformazione getTime (fase Triplification)

Inizialmente sulla base del *processName* (passato come parametro) si ricava dalla tabella MySQL *process_manager2* il valore del campo *last_triples* che corrisponde alla data dell'ultima generazione delle triple RDF per quello specifico dataset. Questo valore viene poi convertito in un timestamp (t*imestampLT*).

Nello step *Hbase_to_MySQL* viene richiamata la trasformazione *HbaseToMysql.ktr* (Figura 11) che prima di tutto recupera i dati prodotti in output dalla fase QI e memorizzati in HBase filtrandoli opportunamente sulla base del nome del processo e del *timestampLT* recuperato in precedenza (blocco HBase Input). In questo modo saranno generate delle triple solo se i dati sono stati modificati/aggiornati nel tempo. I dati così recuperati vengono mappati e memorizzati sui rispettivi campi della tabella temporanea MySQL creata al passo precedente tramite lo step *Table Output*.

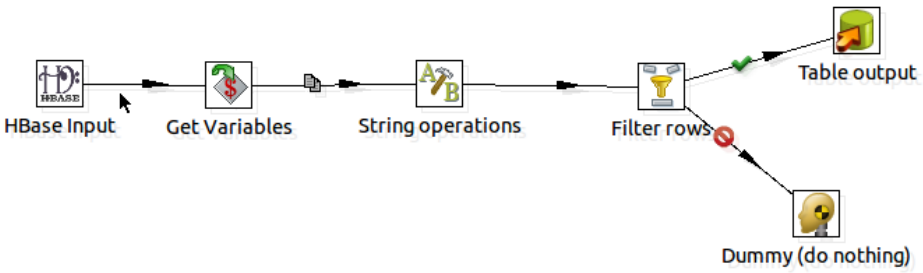

Figura 11 - Trasformazione Hbase_to_MySQL (fase Triplification)

Successivamente si effettua un controllo per verificare che la tabella MySQL non sia vuota, cioè che siano stati estratti dei nuovi dati da HBase (step *Evaluate rows number*). In caso affermativo viene attivata la trasformazione create *actualDate* in cui si ricava la data al momento dell'esecuzione che sarà utilizzata per costruire il path in cui saranno memorizzate le nuove triple RDF generate.

Il passo successivo è lo step di tipo ***Execute a shell script*** *create RegionCsvTriples* in cui viene specificato il comando mediante il quale verranno generate le triple RDF. Infine la trasformazione *update last_triples* si occuperà di aggiornare il campo last_triples della tabella MySQL *process_manager2* con il valore del campo *date_LT* che indica la data di ultima generazione delle triple (creato nello step create actualDate).

Esempio B

Di seguito verranno descritte in dettaglio le fasi di Ingestion (parzialmente analoga a quella dell'esempio precedente) e Quality Improvement del processo ETL inerente l'acquisizione del dataset "*Distributori_di_carburante* " disponibile sul sito opendata.comune.fi.it in formato kmz (http://datigis.comune.fi.it/kml/distributori.kmz); è omessa la fase relativa alla generazione delle triple poichè identica a quella illustrata nell'esempio A.

1. Data Ingestion

La fase di data ingestion prevede l'acquisizione del file sorgente dal portale Open Data del comune di Firenze, il suo salvataggio in una specifica cartella e il caricamento dei dati nella relativa tabella HBase. Il punto di partenza di questo processo è il Job ***Main*** la cui struttura è riportata in Figura 12.

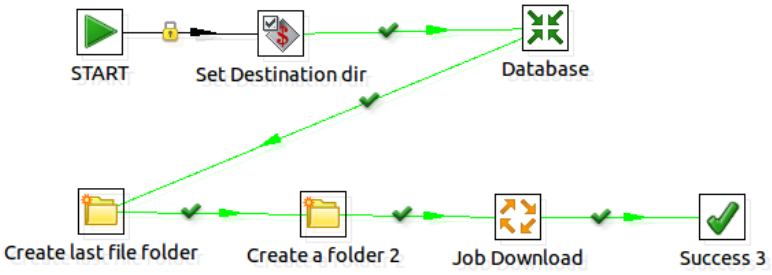

Figura 12 - Job Main (fase Data Ingestion)

Nelle impostazioni generali del Job deve essere definito il parametro *processName,* il cui valore verrà specificato al momento dell'esecuzione dell'intero processo.

Lo step *Set Destination dir* è un blocco di tipo *Set variables* in cui viene settata la variabile **DestinationDir** che indica la radice del path il cui sarà memorizzato il file sorgente; il valore di questa variabile dovrà essere **/Sources/Categoria.**

Lo step *Database* è di tipo *Transformation Executor* e richiama la trasformazione Database.ktr che ha la struttura riportata in Figura 13.

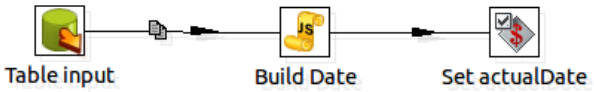

Figura 13 - Trasformazione Database (fase Data Ingestion)

Lo step *Table input* recupera dalla tabella MySQL process_manager2 tutti i campi relativi al processo in esecuzione tramite una query SQL.

Lo step *Build Date* è un blocco in cui è possibile scrivere del codice Javascript. Nello specifico in questo step, a partire dalla data attuale, vengono ricavati i campi per costruire il percorso in cui memorizzare il file scaricato. Successivamente nello step *Set actualDate* i campi definiti in precedenza sono trasformati in variabili e, all'uscita da questo step, 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.

Lo step *Job Download* è un blocco di tipo *Job* in cui è richiamato il job *Download.kjb* la cui struttura è riportata in Figura 14.

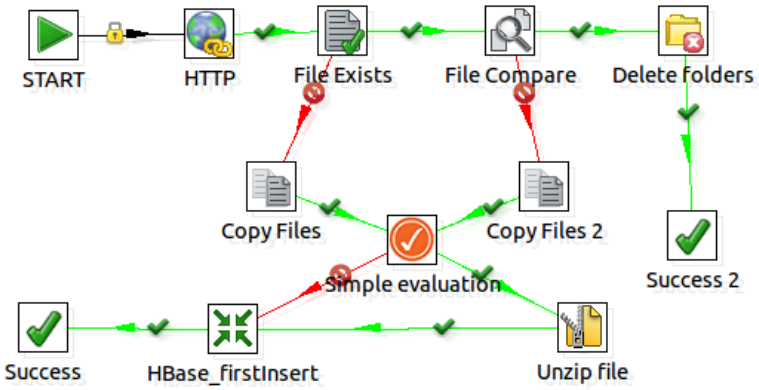

Figura 14 - Job Download (fase Data Ingestion)

Questo job si differenzia rispetto al Download.kjb dell'esempio A per i seguenti due aspetti.

- Nello step HTTP il campo URL viene impostato utilizzando la variabile ${PARAM} che contiene il valore recuperato dalla colonna param della tabella MySQL process_manager2. Tale valore rappresenta l'indirizzo web da cui scaricare il file sorgente.
- Nello step *simple evaluation* viene analizzato il tipo del dataset scaricato (utilizzando la variabile ${FORMAT} contenete il valore recuperato dalla colonna Format della tabella MySQL process_manager2). Nel caso di file *kmz* il flusso di esecuzione viene indirizzato sullo step *Unzip file* che estrae il file *kml* del dato in input e lo memorizza nella cartella *${DestinationDir}/${processName}/1Last_file.*

Successivamente nello step HBase_firstinsert viene richiamata la trasformazione *Distributori_di_carburante_kmz.ktr* che ha la seguente struttura.

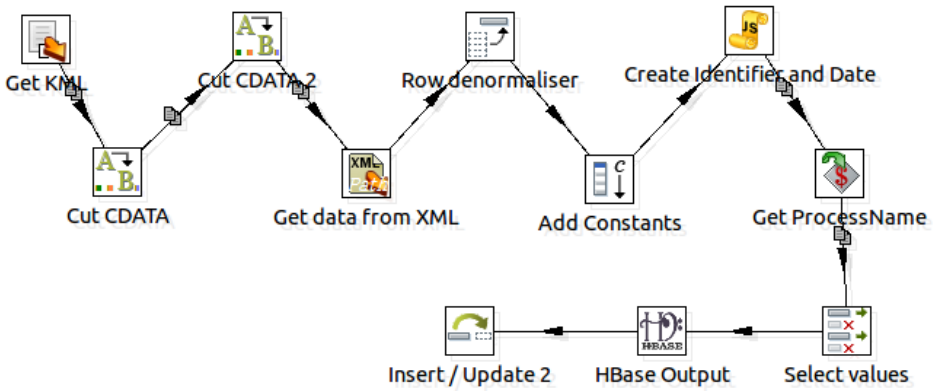

Figura 15 - Trasformazione Distributori di carburante (fase Data Ingestion)

Una volta estratto il file in formato kml è opportuno realizzare un'analisi preliminare (aprendolo in un editor di testo) in modo da individuare quali siano i campi di interesse che devono poi essere recuperati e memorizzati, un esempio è mostrato in Figura 16.

Figura 16 - Frammento file .kml

Nello step *Get KML* (di tipo *Load content file in memory*) viene recuperato l'intero contenuto del file sorgente scaricato, che sarà salvato all'interno del campo ***Field Content***. Il contenuto viene poi modificato negli step Cut CDATA e Cut CDATA_2 in cui vengono rimosse rispettivamente le stringhe **<![CDATA[** e **]]>**. Il nuovo campo in uscita (Mod-file1) viene passato allo step *Get data from XML,* in cui mediate l'uso di XPath vengo estratti i campi *tag, span e coordinates* come mostrato in Figura 17.

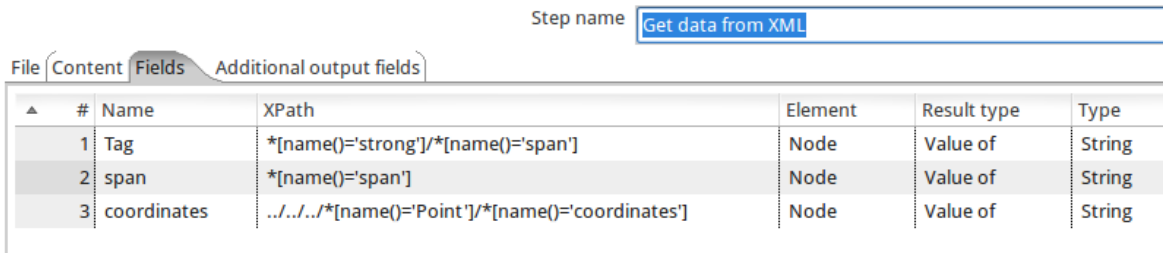

Figura 17 - Estrazione campi da file KML

Successivamente nello step *Row denormaliser* viene creato un flusso di righe, cioè per ogni istanza del campo coordinates si ottiene una riga caratterizzata dai campi ID - CODSTRADA - TOPONIMO (estratti dal campo *span* definito nello step precedente). Per ogni riga vengono poi definiti tre nuovi campi, *locality - country-name - sigla*, impostati con valori fissi. Nello step *Create Identifier and Date* vengono definiti i campi identifier (che sarà utilizzato come chiave nella memorizzazione in HBase), actualdate e timestamp (che identificano il riferimento temporale dell'acquisizione dei dati).

Infine, dopo aver selezionato i campi di interesse, si procede alla memorizzazione in HBase utilizzando lo step *HBase output* settato in maniera analoga a quanto descritto nell'esempio A (utilizzando ovviamente una nuova tabella); viene poi aggiornato il campo *last_update* nella tabella MySQL process_manager2.

2. Quality Improvement

La fase di Quality Improvement ha lo scopo di migliorare la qualità dei dati acquisiti nella fase di ingestion, operando delle opportune modifiche sullo specifico dato in esame e memorizzando poi il contenuto in una nuova tabella HBase. In questo caso trattandosi di dataset del comune di Firenze contenenti informazioni di georeferenzazione (Indirizzo e coordinate geografiche) si realizza una "riconciliazione preliminare", cioè utilizzando una tabella MySQL, contenente l'elenco delle strade della città, si ricava il *codice toponimo* relativo ad uno specifico indirizzo.

Il Job iniziale *Data_QI.kjb* richiama la trasformazione principale *Distributori_di_carburante_QI.ktr* la cui struttura è mostrata in Figura 18. Come si può osservare inizialmente vengono recuperati da HBase i campi memorizzati in fase di Ingestion e vengono separate le coordinate in modo da ottenere i valori di latitudine e longitudine (step *Split coord*). Nello step *Select values* vengono selezionati e rinominati gli effettivi campi di interesse, mentre in *Add costants* si definiscono i due nuovi campi ***address_syn*** e ***cod_toponimo*** che saranno utilizzati in seguito. Dopo queste operazioni preliminari si ha la memorizzazione in HBase nella nuova tabella nome_processo_QI.

Si passa poi al recupero del codice toponimo; questa operazione, ovviamente, verrà fatta solo per le righe aventi il campo *streetAddress* diverso da NULL (controllo realizzato nello step *Filter rows 2*), le altre termineranno l'esecuzione nel blocco *Dummy.*

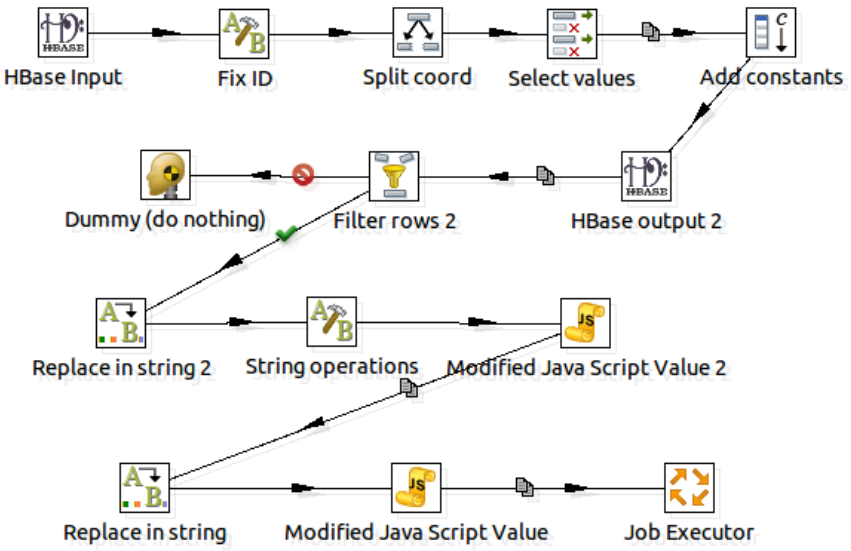

Figura 18 - Trasformazione Distributori_di_carburante_QI (Fase di QI)

Per ottenere il toponimo l'idea è quella di ricavare la parola (o le parole) composta dal maggior numero di caratteri presente all'interno del campo indirizzo. A tal proposito si realizza l'eliminazione di alcuni caratteri speciali, deii nomi propri di persona più comuni, e di alcune parole ben definite; questo insieme di operazioni è svolto nei due blocchi *Modified Java Script Value* e nei vari step che intervengono sulle stringhe (*Replace in string* e *String Operations*). L'eliminazione di alcuni nomi propri di persona serve per ridurre il numeri di risultati che saranno restituiti dalla query SQL eseguita nella fase successiva di questo processo ETL.

A questo punto ciascuna riga del flusso viene passata allo step Job Excecutor (che richiama il job *Job_DB_Like.kjb*) specificando un insieme di parametri come mostrato in Figura 19.

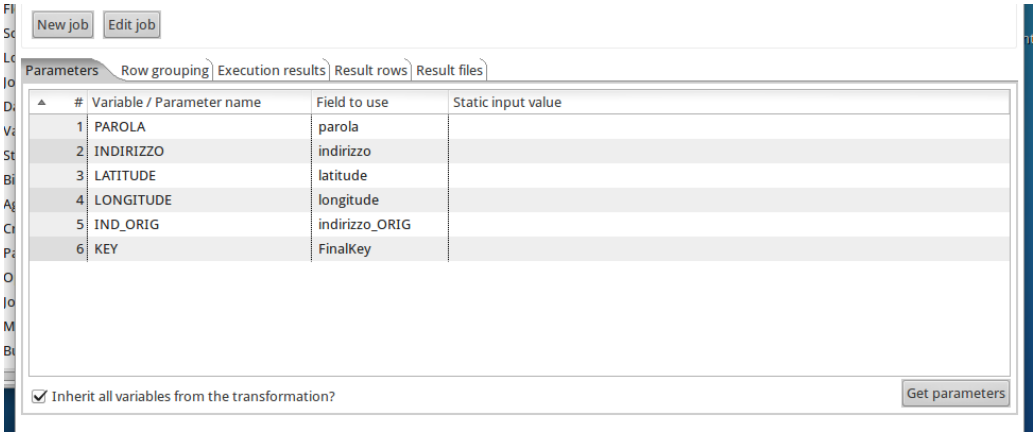

Figura 19 - Set di parametri passati a Job_DB_Like.kjb

All'interno del job *Job_DB_Like.kjb* viene semplicemente richiamata la trasformazione *DB_Toponimo_distributori.ktr*, la cui esecuzione dovrà essere ripetuta per ciascuna riga passata in input (ciò si realizza spuntando l'opzione *Execute for every input row* nella tab Advanced), la cui struttura è illustrata in Figura 20.

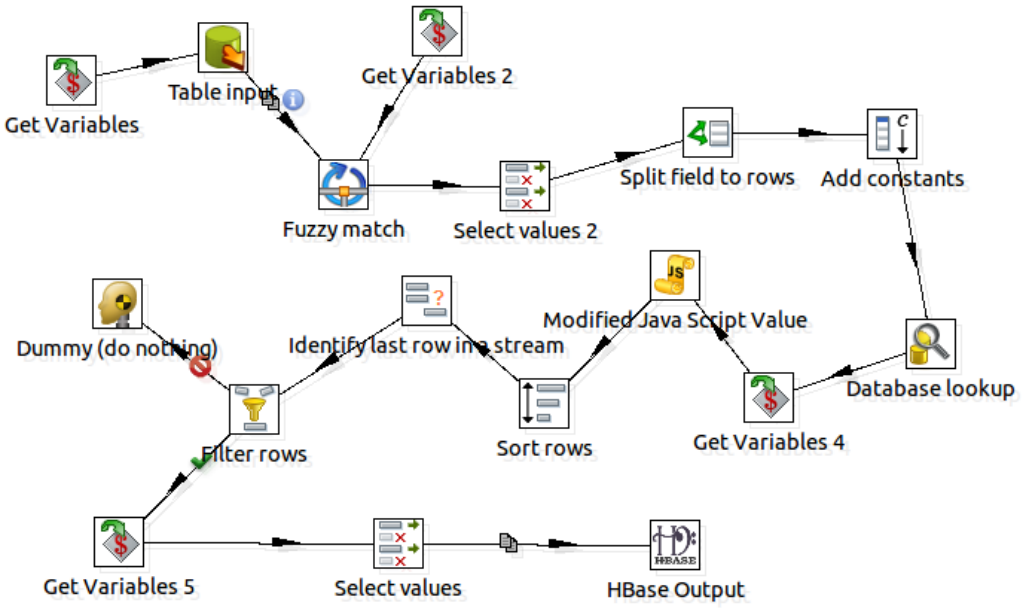

Figura 20 - Trasformazione DB_Toponimo_distributori.ktr (fase di QI)

Per ciascuna riga in input si realizza la seguente query (Figura 21) sulla tabella MySQL *tbl_toponimo_BIS* (step *Table input*).

```
SQL
SELECT EXT_NAME FROM tbl_toponimo_BIS where COD_COM = 'D612' and EXT_NAME LIKE '%${PAROLA}%';
```

Figura 21 - Query SQL per estrarre il campo EXT_NAME (fase QI)

In questo modo sulla base della variabile PAROLA individuata in precedenza si ricavano tutti gli indirizzi in cui essa è contenuta (campo EXT_NAME), la ricerca è limitata al comune di firenze indicando il codice comune D612.

Successivamente nello step *Fuzzy match* i risultati della query vengono confrontati con l'indirizzo in input (campo *ind* proveniente dallo step *Get Variables 2*) secondo il criterio di similarità *Pair letters Similarity*, e selezionati sulla base di uno specifico valore di soglia. Per ogni EXT_NAME che supera il valore di soglia verrà memorizzato all'interno del campo di output *match* (definito in questo stesso step) utilizzando la virgola come separatore; in questo modo si realizza un ulteriore filtraggio degli indirizzi da cui ricavare il codice toponimo cercato.

Successivamente dal campo *match* si ricava il campo *match2*, suddividendo il primo (utilizzando la virgola come separatore) in modo da ottenere una riga per ciascun indirizzo. Poi nello step *Database Lookup* per ciascun indirizzo selezionato si ricavano dalla tabella MySQL *tbl_toponimo_BIS* i campi COD_TOP, LAT e LON.

Nel successivo step *Modified Java Script Value* vengono utilizzate le coordinate geografiche per calcolare la distanza (campo *dist*) tra l'indirizzo presente nel dataset in ingresso e i match ricavati dalla tabella MySQL, i risultati vengono poi ordinati in modo discendente secondo il campo *dist,* e successivamente viene individuata l'ultima riga del flusso (step *Identify last row in stream*) mediante l'aggiunta di un campo booleano (result).

Infine dopo aver ridotto il flusso di dati all'ultima riga individuata (step *Filter row*), si selezionano (ed eventualmente rinominano) i campi *address_syn, codice_toponimo, streetAddress e FinalKey* che saranno utilizzati per aggiornare la tabella HBase nome_processo_QI, già creata e popolata in precedenza.