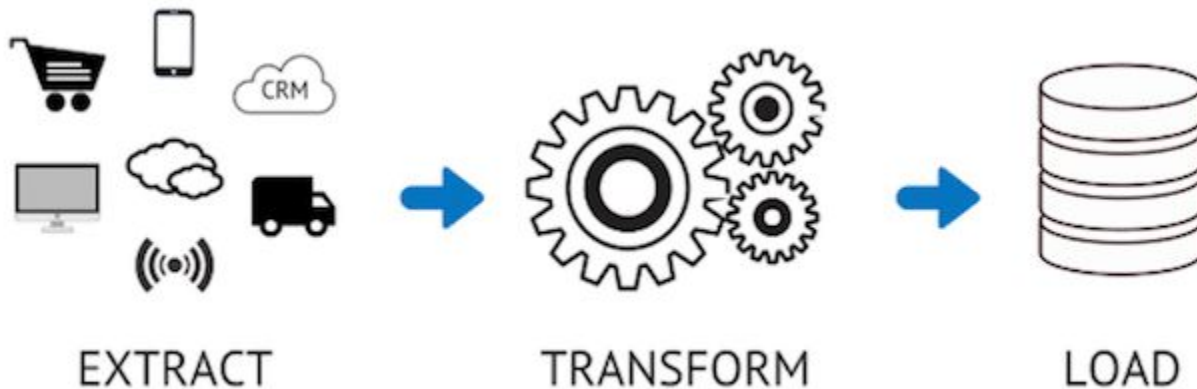


# ETL con Talend

Introduzione ad ETL e Talend Open Studio DI

# Cos'è ETL

**ETL** è l'acronimo per **Extract**, **Trasform** and **Load** e indica un processo di estrazione, trasformazione e caricamento dei dati in un sistema di sintesi.



I dati possono essere estratti da database relazionali, comuni file di testo o da altri sistemi informatici.

Durante la trasformazione dei dati avviene anche la fase di **Data Quality**, più complessa se si ha la necessità di estrarre dati da database disordinati e poco omogenei. Le tecniche di Data Quality permettono di pulire i dati disomogenei, eliminare duplicati e derivare informazioni calcolate; tutto ciò ha lo scopo di rendere conformi dati provenienti da sorgenti diverse e di renderli il più aderente possibile alla logica di business del sistema di analisi per cui vengono estratti.

# In quali ambiti vengono utilizzati i processi ETL

- Migrazione dei dati da un'applicazione a un'altra;
- Replica dei dati per l'esecuzione di backup o analisi della ridondanza;
- Processi operativi quali il trasferimento di dati da un sistema CRM in un ODS (Operational Data Store) per l'ottimizzazione e l'arricchimento, per poi restituire i dati al sistema CRM;
- Inserimento dei dati in un Data Warehouse per l'assimilazione, l'ordinamento e la trasformazione in business intelligence;
- Migrazione delle applicazioni locali in infrastrutture Cloud, Cloud ibride o multi-Cloud;
- Sincronizzazione di sistemi.

L'obiettivo di un processo ETL è ottenere dati puliti e accessibili da utilizzare per attività di analisi o processi di business.

I dati grezzi vengono inizialmente estratti da una vasta gamma di origini. Successivamente abbiamo la fase più critica del processo ETL e cioè la fase di trasformazione. Durante la trasformazione, i dati grezzi vengono opportunamente “convertiti” nei formati richiesti. Se i dati non sono puliti, applicare le regole aziendali di segnalazione diventa complicato. L'ultima fase del processo ETL prevede, in genere, il caricamento dei dati estratti e trasformati in una nuova destinazione. I dati possono essere caricati in un data warehouse in due diversi modi: tramite caricamento completo o incrementale.

# Talend Open Studio for Data Integration

Talend è una software-house statunitense che fornisce servizi e software di integrazione, gestione di dati e big data. Per la realizzazione di software ETL viene fornito il framework Talend Studio for Data Integration, un software basato su linguaggio Java, con lo scopo di fornire un'interfaccia grafica e un insieme di connettori utili allo sviluppo di flussi ETL.

Attraverso una semplice GUI, il programma produce automaticamente scripts in Java per il caricamento e la trasformazione dei dati. È possibile scegliere tra più di 900 componenti, connettersi a 40 database diversi, oltre a fogli Excel, file CSV, JSON, XML e tanti altri. Talend Data Integration permette di programmare i flussi ETL in modo visuale, tramite operazioni di drag&drop sugli elementi dalla palette alla griglia rappresentante il corpo del nostro programma, inoltre, permette la divisione di grandi flussi in unità più piccole, in modo da rendere il lavoro più semplice e scalabile.

# Scaricare TOS DI

Talend Open Studio for Data Integration può essere scaricato gratuitamente dal sito ufficiale nella versione “installabile” e “portable”.

La versione installabile è disponibile per i SO Windows e MacOS al seguente indirizzo:

<https://www.talend.com/it/products/data-integration/data-integration-open-studio/>



La versione portable (TOS\_DI-20200219\_1130-V7.3.1.zip), invece, la potete trovare al seguente indirizzo:

<https://www.talend.com/it/products/data-integration-manuals-release-notes/>

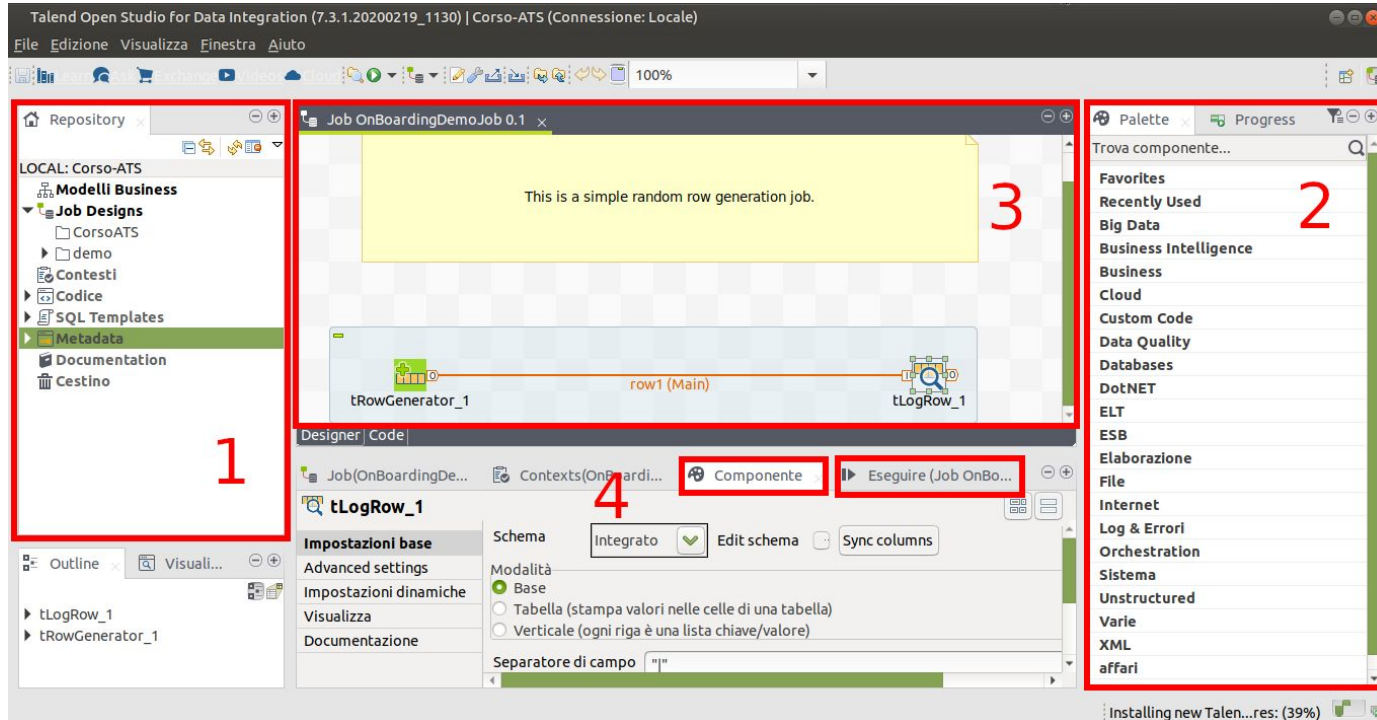
TOS_DI-20200219_1130-V7.3.1-osx-installer.dmg	7.3.1	26 febbraio 2020	Main	MAC	852MB	<a href="#">US and Europe</a>
TOS_DI-20200219_1130-V7.3.1.zip	7.3.1	26 febbraio 2020	Main	Unix_Linux Windows MAC	904MB	<a href="#">US and Europe</a>
TOS_DI-Win32-20200219_1130-V7.3.1.exe	7.3.1	26 febbraio 2020	Main	Windows	750MB	<a href="#">US and Europe</a>

Vi consiglio l'utilizzo dell'ultima versione stabile contrassegnata dal valore **Main** nella colonna **Tipo release**.

Per il download basta selezionare il link che trovate nell'ultima colonna presente nella tabella mostrata dall'immagine precedente (US and Europe).



# L'interfaccia di TSO DI



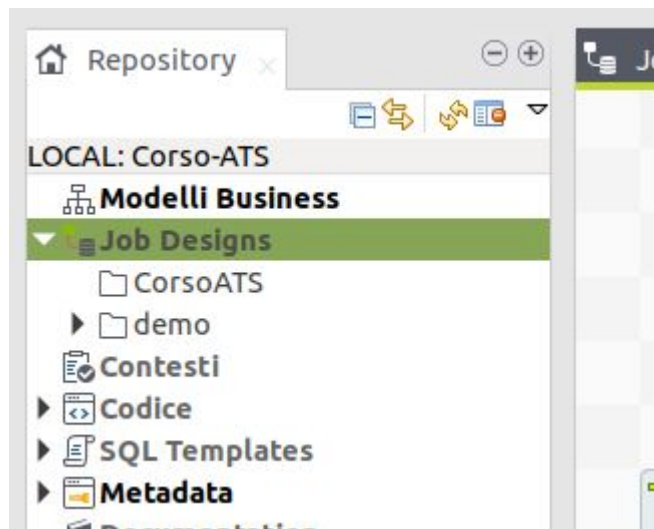
Talend Studio consente di creare programmi ETL mediante un'interfaccia utente grafica. Tali programmi sono chiamati job DI (Data Integration) o di integrazione dei dati.

L'interfaccia utente di Talend Studio include diversi pannelli chiamati viste

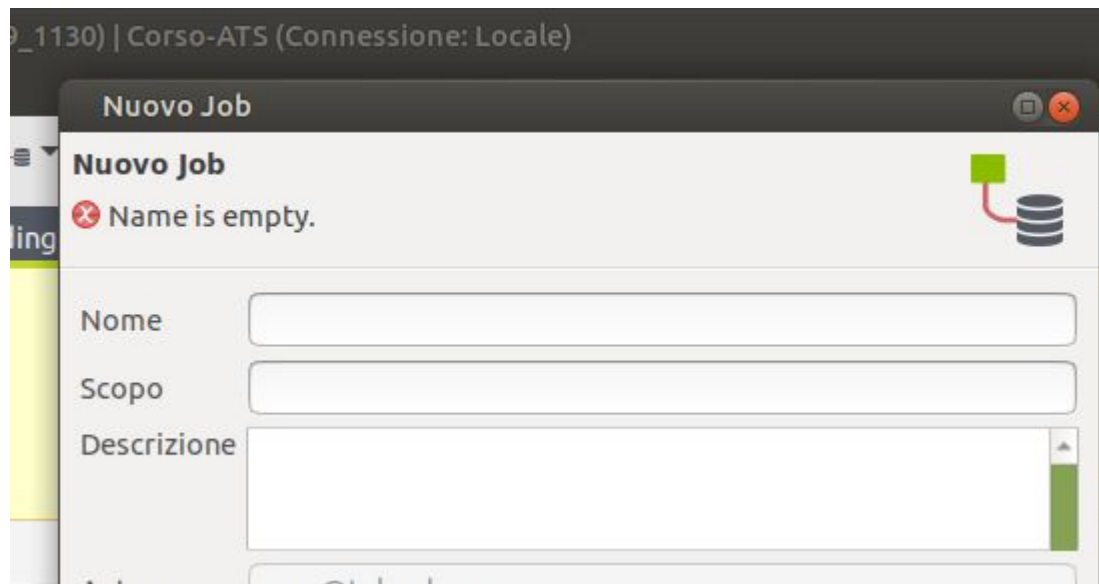
1. Nella vista Project Repository (Repository progetti) sono elencate tutte le voci di progetto, come job (programmi ETL Java), servizi, codice, metadati e documentazione dei progetti;
2. Nell'area Palette (Tavolozza) sono elencati tutti i componenti disponibili, organizzati in cartelle;
3. La vista Job Designer (Progettazione job) è la vista principale di Talend Studio, dove i vari componenti vengono utilizzati per creare job ETL;
4. Nell'area Component (Componente) sono visualizzati tutti i parametri necessari per configurare un componente. Le informazioni visualizzate in quest'area dipendono dalle selezioni effettuate in Job Designer (Progettazione job). Dalla vista Run (Esegui) è possibile attivare l'esecuzione di un job Talend e visualizzare

# Creazione del primo job

Project Repository (Repository progetti), fai clic con il pulsante destro del mouse su **Job Designs** (Progetti job).



Per aprire la procedura guidata di creazione di un nuovo job, fai clic su **Create Job** (Crea job standard).



The screenshot shows a software window titled "Nuovo Job" (New Job) with a dark header bar. Below the header, the title "Nuovo Job" is repeated. A red error icon and the message "Name is empty." are displayed. To the right of the error message is a small icon of a database cylinder with a red line connecting it to a green square. Below the error message are three input fields: "Nome" (Name), "Scopo" (Purpose), and "Descrizione" (Description). The "Nome" field is currently empty. At the bottom of the window, a partially visible email address "xxx@tld.com" is shown.

Nel campo Name (Nome) della procedura guidata, inserisci il nome del job, in questo caso **testJob**.

Nel campo Purpose (Scopo), inserisci *Visualizzare un messaggio*.

Nel campo Description (Descrizione), digita *Questo tutorial impiega un componente per visualizzare una casella di testo con un messaggio personalizzato*.

Per chiudere la procedura guidata e creare il job, fai clic su **Finish** (Fine).

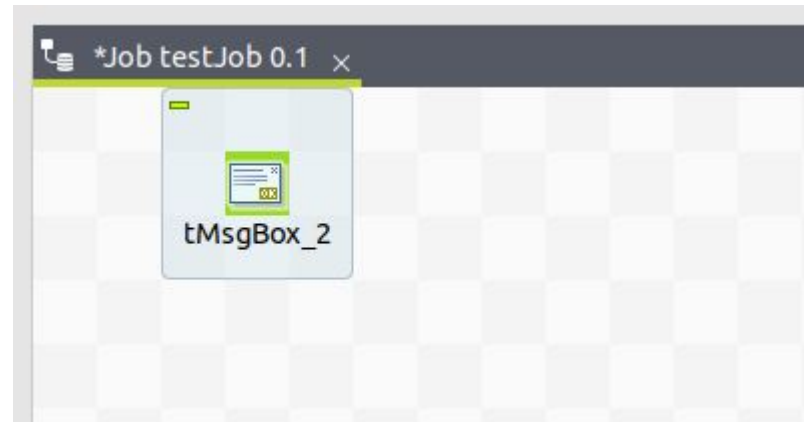
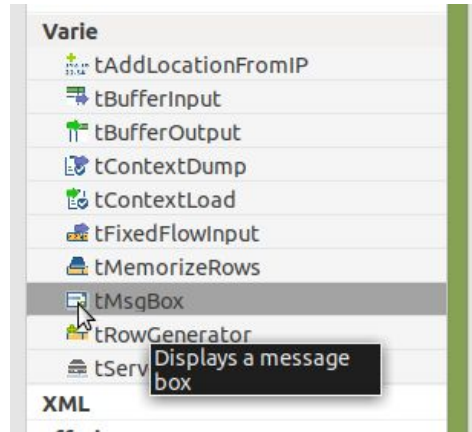
Viene aperto un nuovo job vuoto in **Job Designer** (Progettazione job).

I job DI impiegano componenti. Talend Studio offre una libreria completa di oltre 800 componenti per l'integrazione dei dati.

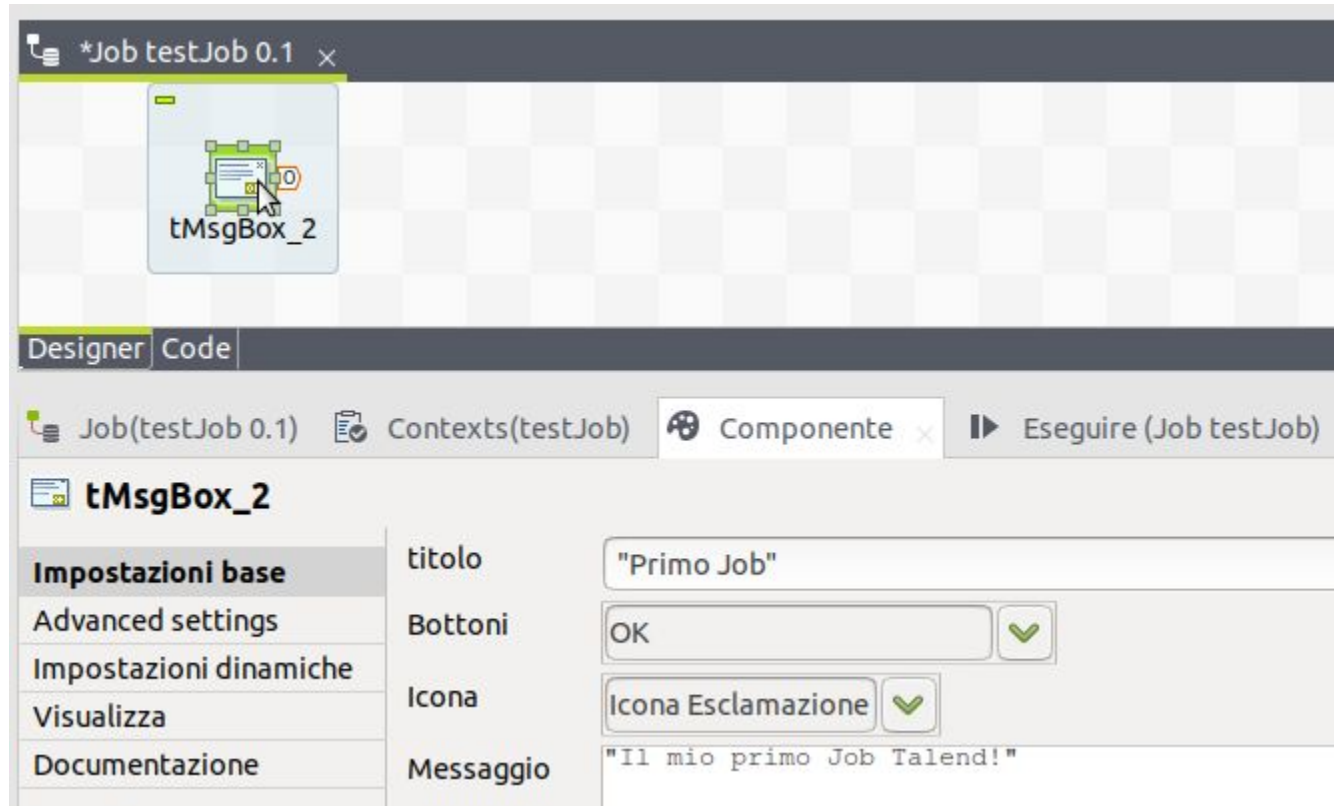


A questo punto possiamo provare ad aggiungere un componente al nostro Job. Come anticipato in TSO la creazione di un Job avviene tramite l'interfaccia visuale, è possibile aggiungere i nostri componenti all'interno del Job con una semplice operazione di drag and drop dall'area palette alla vista Job Designer.

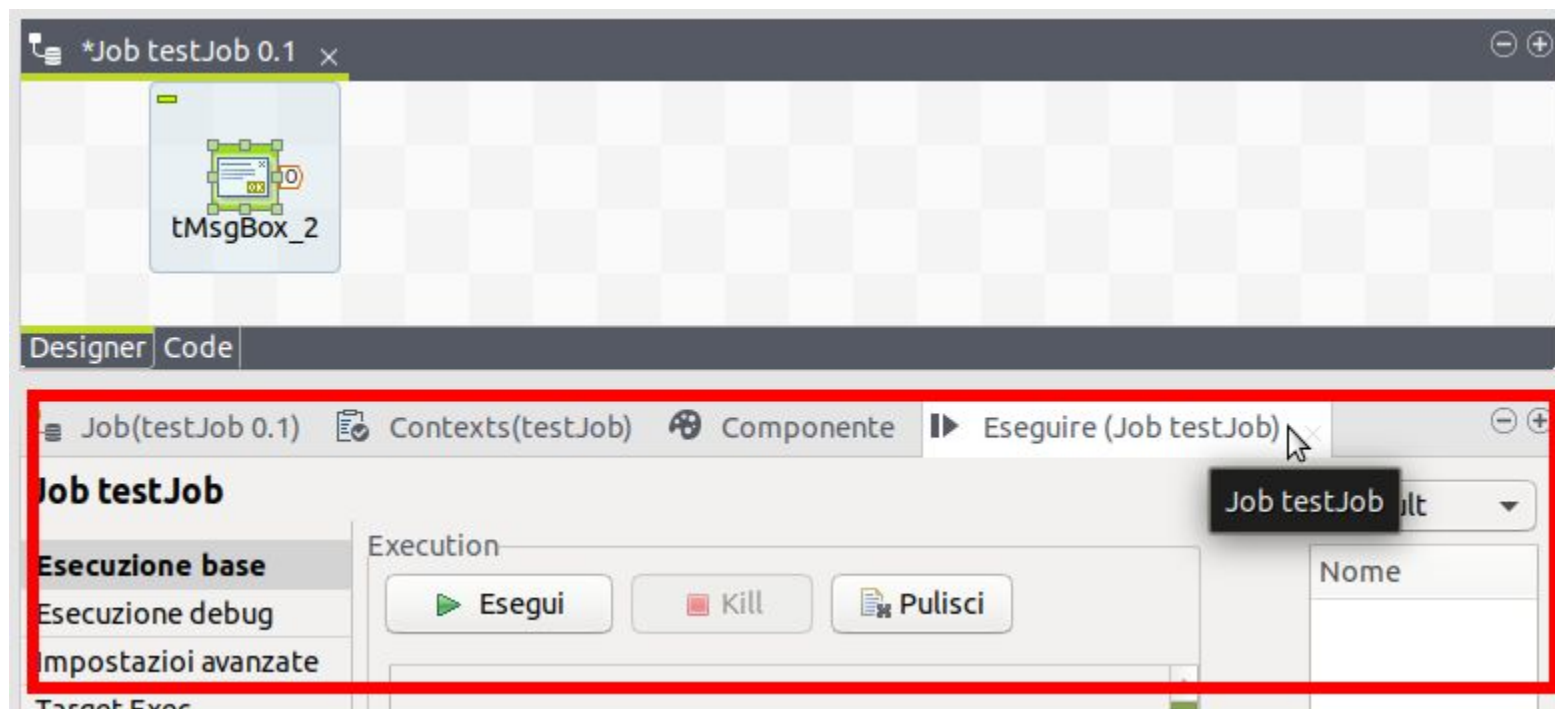
Aggiungi un componente **tMsgBox** al nostro Job. Per farlo lo seleziono in Palette, sotto la voce Varie e trascinalo in Job Designer (Progettazione job). Questo semplice componente, utile per i test, consente di visualizzare una casella di messaggio.



A questo punto puoi configurare il componente appena inserito. Seleziona il componente **tMsgBox\_1** dalla view Job Designer e, dopo aver selezionato la view **Componente**, modifica le informazioni di base del componente.



Sei pronto per eseguire il tuo Job! Ora non devi fare altro che selezionare la vista “Esegui” e selezionare il bottone “Esegui”.





Dopo aver effettuato la compilazione del progetto Talend mostrerà il risultato del Job appena eseguito.



Un job è formato da un insieme di componenti che si scambiano informazioni tramite un flusso rappresentato da frecce direzionali.

Come già accennato, per importare un componente all'interno del progetto, è sufficiente eseguire un'operazione di Drag and Drop dalla palette alla griglia principale.

Per unire due componenti, è sufficiente cliccare col tasto destro sull'elemento sorgente, e dopo aver selezionato il tipo di riga, cliccare sull'elemento di destinazione



Le righe disponibili per collegare due connettori sono: **Principale, Su subjob OK / Su errore subjob, Su componente OK / Su errore componente e Esegui se**

## Principale

Permette il transito di dati da un componente all'altro. Questo tipo di riga richiede che gli elementi di sorgente e destinazione abbiano tra le proprietà uno «schema». Lo schema è un insieme di attributi che possono essere scambiati tra due elementi.

Ad esempio, nel caso di un connettore che esegue una query, lo schema sarà strutturato in modo da poter contenere il risultato della query. Sulla freccia Principale sarà definito uno schema.

## Su subjob OK / Su errore subjob

Tutti i componenti uniti tramite una riga Principale fanno parte dello stesso **Subjob**. Il tipo di riga «**Su subjob OK**» viene attivata solo al completamento del **Subjob**. Quindi, il componente di destinazione di questa freccia sarà eseguito solo quando l'intero **subjob** di sorgente sarà completo. La riga **Su errore subjob** funziona in modo simile, ma si attiva al verificarsi di un errore nel subjob di sorgente.

## **Su componente OK / Su errore componente**

Il componente di destinazione di questa freccia sarà eseguito solo dopo l'esecuzione del componente sorgente.

La riga Su errore componente funziona in modo simile, ma si attiva al verificarsi di un errore nel componente sorgente.

## **Esegui se**

Questo componente permette di specificare una condizione. L'elemento di destinazione viene eseguito se, una volta eseguito il componente sorgente, la condizione è verificata.

L'inserimento della condizione non è opzionale!

L'unico connettore che richiede uno schema è quello principale.

N.B. Durante l'esecuzione del flusso, le informazioni vengono passate dai componenti di input ad output una riga alla volta. Questo significa, ad esempio, che un componente che esegue una query invia il risultato al componente di output una riga alla volta.

L'iterazione sul risultato viene eseguita automaticamente!

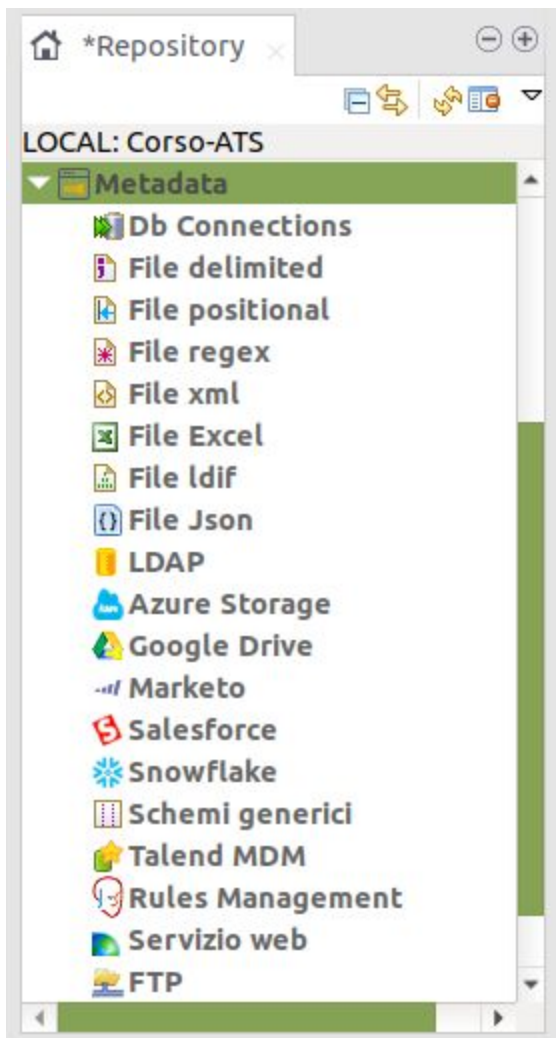
# Metadati

I metadati permettono di gestire le connessioni verso i sistemi esterni.

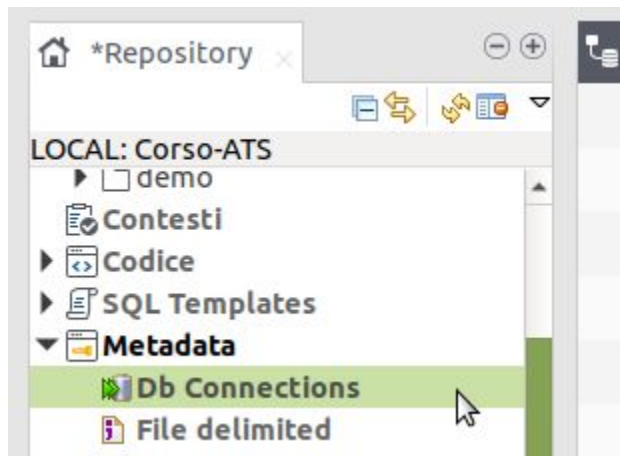
Una volta stabilita una connessione, è possibile parametrizzarla, modificarla, testarla e usufruire di tutte le funzionalità offerte da essa.

I metadati sono associati all'intero progetto e non al flusso, quindi possono essere dichiarati una sola volta, e utilizzati in tutti i job del processo. Le eventuali modifiche si rifletteranno automaticamente su Job/componenti interessati.

Talend può stabilire connessioni verso tutti i sistemi indicati all'interno del menù **metadata** presente nella view Repository.



Creiamo una connessione verso un database. All'interno della view **Repository** fare click con il pulsante destro del mouse su **Db Connections**.



e, dal menu contestuale che si apre, selezioniamo **crea connessione**.

Inseriamo le informazioni di base sulla connessione (nome, scopo e descrizione) e selezioniamo il bottone **Next**.

Nel passo 2/2 selezionare il tipo di database (nel nostro caso MySQL). La prima volta che selezioniamo un tipo di database ci verrà chiesto di scaricare i driver.



## Connessione Database

### Nuova Connessione al Database - Passo 2/2

Definire parametri di connessione



Tipo DB MySQL

Versione Db

MySQL 8

Stringa di connessione

jdbc:mysql://accademy.chdaxtz2ghx5.eu-central-1.rds.amazonaws.com:3306/acca

Login

accademy\_user

Password

.....

Server

accademy.chdaxtz2ghx5.eu-central-1.rds.amazonaws.com

Porta

3306

DataBase

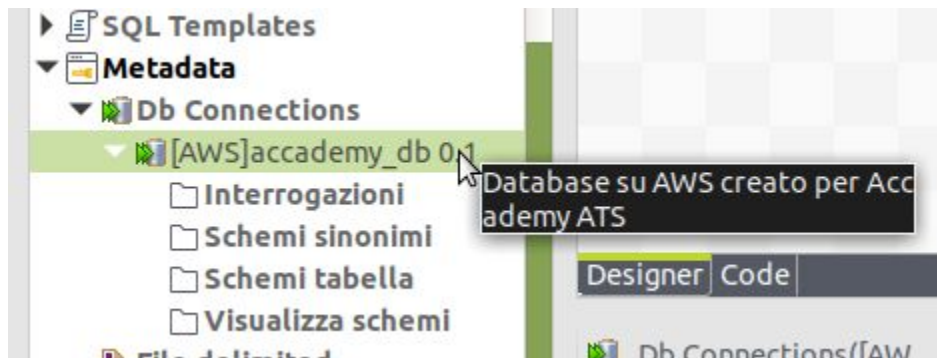
accademy\_db

Test connection

v

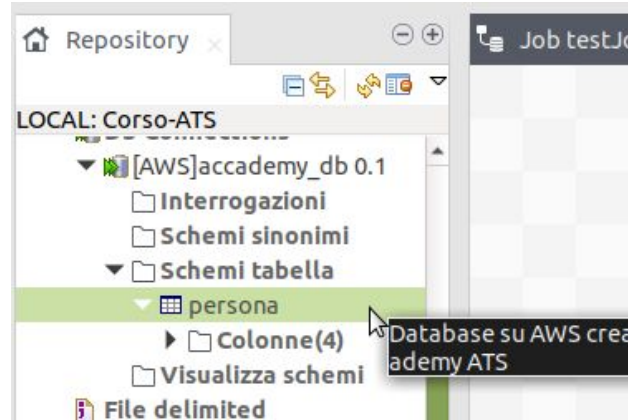
Una volta inserite tutte le informazioni proviamo ad effettuare un test della connessione con il bottone **Test Connection**. Se la connessione va a buon fine possiamo terminare la procedura di creazione con il bottone **Finish**

Una volta terminata la procedura dovremmo ritrovarci all'interno della sezione **metadata**



Per recuperare automaticamente tutti gli schemi delle tabelle, facciamo clic con il pulsante destro del mouse sui metadati **[AWS]accademy\_db 0.1** nel Project Repository quindi selezioniamo **Recupera schema** che ci consentirà di recuperare la struttura di tutte le tabelle presenti.

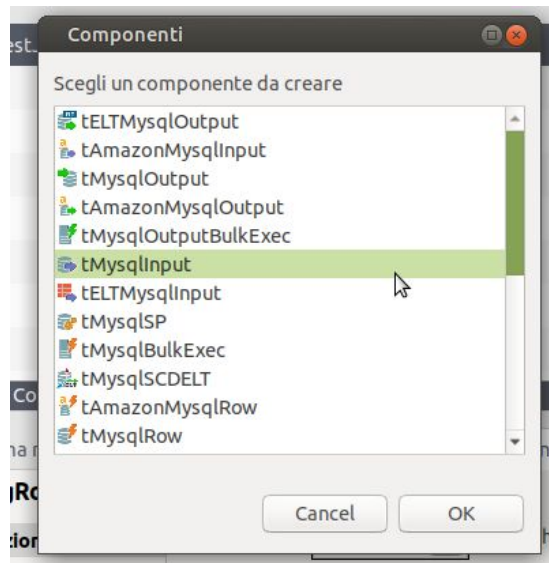
Per leggere i dati di una tabella nell'elenco, seleziona la tabella e trascinala in Job Designer.



Nella finestra Components (Componenti), fai clic su **tMySQLInput**, quindi su **OK**.

Viene creato un componente **tMySQLInput** con le informazioni del repository. Viene utilizzata la connessione **[AWS]accademy\_db 0.1** e per lo schema vengono utilizzate le informazioni del repository acquisite dalla tabella dei metadati **persona**.

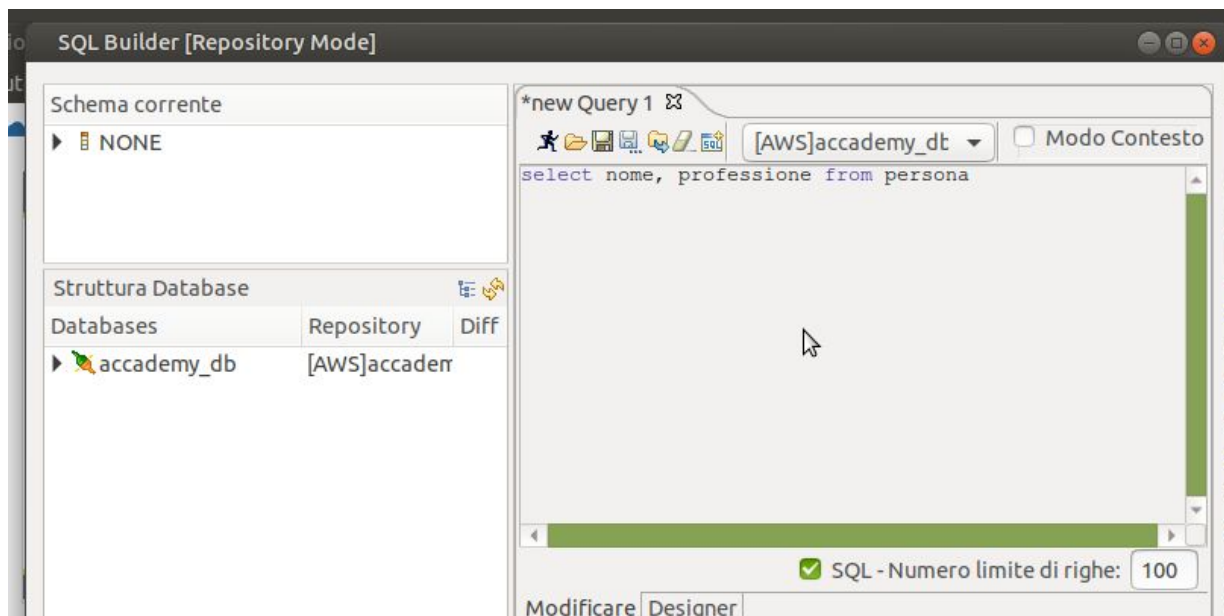
Inoltre, Talend genera la query SQL e la invia alla tabella **persona**.



Per visualizzare i dati della tabella, aggiungi il componente **tLogRow** e collega il componente **persona** al componente **tLogRow\_1**.

Per eseguire il job, fai clic su **Esegui** nella vista **Eseguire**. I dati della tabella **persona** vengono visualizzati.

È possibile creare e salvare delle query custom, per interrogare il database cliccando con il tasto destro sulla connessione creata e selezionando la voce modifica query dal menu contestuale.



Le query create saranno presenti nella cartella «Interrogazioni», e potranno essere modificate o lanciate.

Creiamo un metadato verso un file Excel. Nella sezione metadata facciamo click con il tasto destro del mouse su **File Excel** e selezioniamo la voce **Crea file Excel**.

A questo punto selezioniamo un nome per il nostro metadata e andiamo avanti con **next**.

Nel passo successivo sarà necessario selezionare il file Excel dal nostro file system tramite il pulsante **naviga**.



A questo punto sarà necessario selezionare i fogli (sheets) del file Excel che vogliamo importare.

Nei passi successivi è possibile determinare i caratteri di escape, la presenza di una riga con le intestazioni di colonna e definire i tipi di dato presenti nelle varie colonne.

La lista con i tipi di dato viene autogenerata ma è possibile modificarla.

Nel modificare le impostazioni delle tipologie di dato noterete che, selezionando la checkbox **nullable** il tipo di dato viene automaticamente impostato con la classe Wrapper mentre definendo un campo che non ammette valori null verrà utilizzato il tipo primitivo.

Una volta creato il Metadata, è possibile modificarlo semplicemente cliccando su di esso col tasto destro e selezionando la voce «Modifica connessione». Così facendo si avrà la possibilità di modificare tutte le impostazioni del Metadata. Cliccando sul tasto destro, sarà disponibile anche la voce «Cancella», che elimina il metadata.

# Esercitazioni

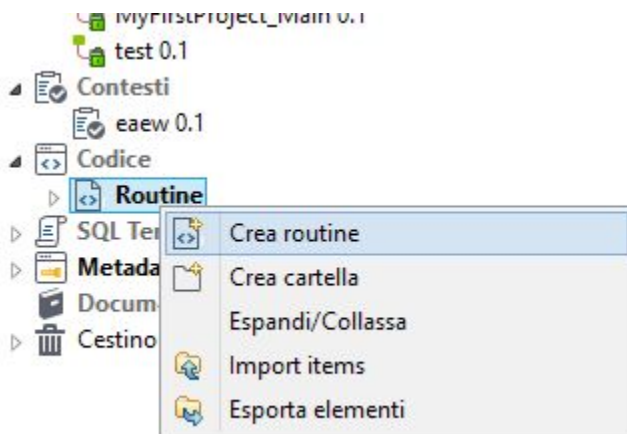
- Creare una connessione a un database MySQL, ed estrarne lo schema.
- Creare una connessione a un file CSV ed estrarne lo schema.



# Routine

Talend permette di scrivere codice custom, in modo da poter risolvere anche problematiche non previste, o particolarmente complesse.

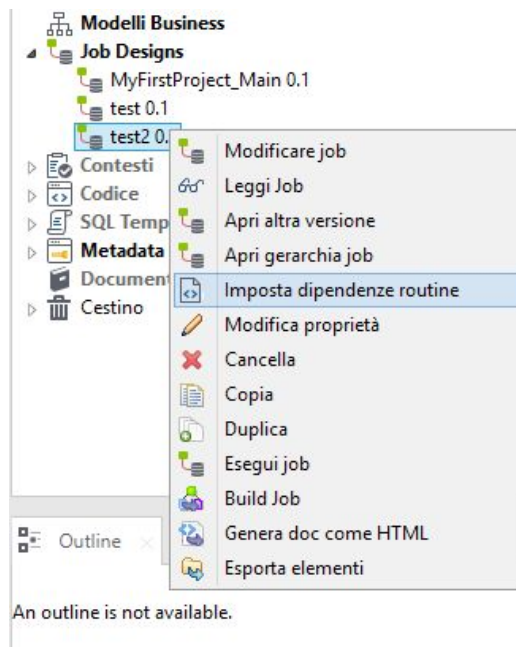
Per creare una classe, è sufficiente cliccare su «**Codice**», poi su «**Routine**» col tasto destro e selezionare la voce «**Crea routine**».



Il linguaggio di programmazione utilizzato per lo sviluppo di routine è il Java.

Le classi create godono quindi di tutte le caratteristiche relative a una classe Java.

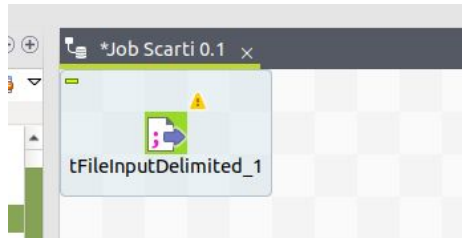
Per utilizzare una routine all'interno di un job è necessario dichiarare la dipendenza verso di essa



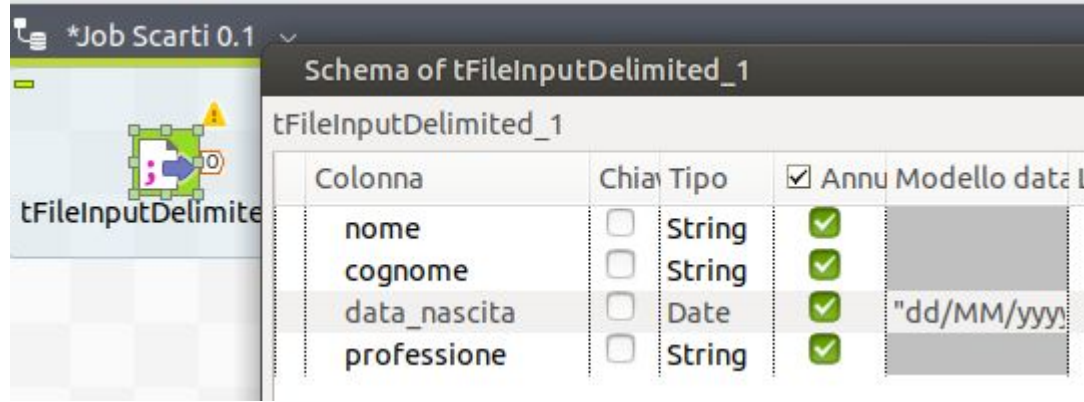
# Flusso “Scarti”

Il flusso **Scarto (reject)** consente di gestire eventuali dati non conformi allo schema impostato per un determinato componente. In alcuni casi, a seconda dei requisiti applicativi, gli scarti non sono accettabili. In questi casi, i flussi di **Scarto** dovrebbero essere disabilitati ed il nostro job dovrebbe fallire.

Creiamo un nuovo Job, ed aggiungiamo un componente **tFileInputDelimited**. Impostiamo il componente appena inserito per caricare i dati presenti nel file **persone.csv** presente nel path **datasets/reject** del repository del corso e selezioniamo il checkbox “Interrompi se rilevato errore” nelle impostazioni di base del componente.

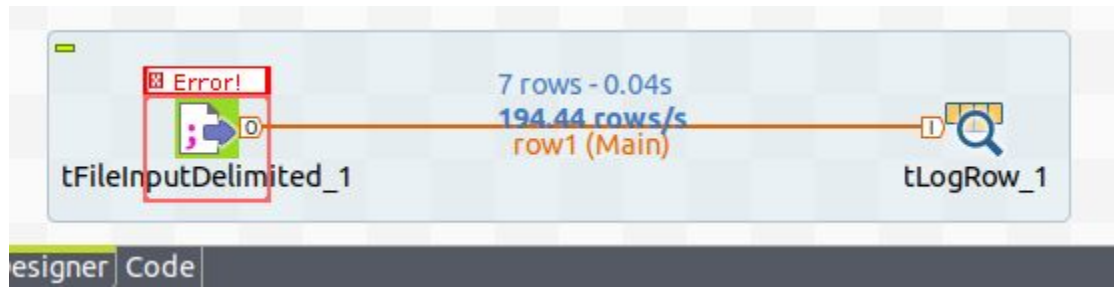


Aggiungiamo lo schema per la gestione delle informazioni presenti all'interno del file caricato



facendo attenzione ad impostare correttamente il tipo del campo data\_nascita e fornendo il giusto pattern utilizzato per la parserizzazione.

A questo punto aggiungiamo un componente **tLogRow** per stampare le informazioni contenute nel file e proviamo a lanciare il nostro Job.



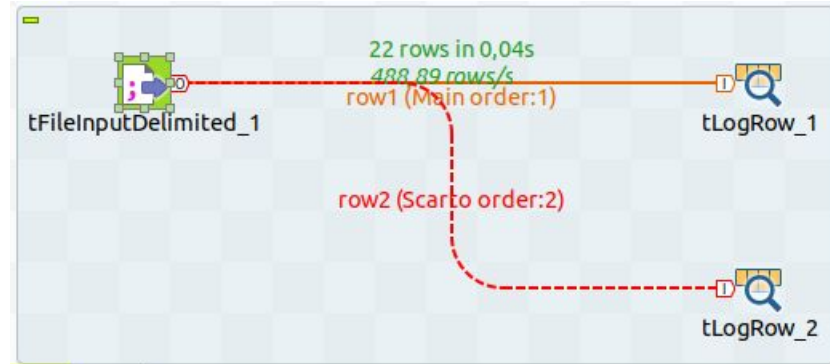
Come possiamo notare il nostro Job è andato in errore. Controllando i log, in effetti, possiamo notare che il nostro dataset contiene delle informazioni che non possono essere parserizzate correttamente nella colonna `data_nascita`.

Se volessimo far sì che il nostro Job completi la sua esecuzione anche in presenza di record che non possono essere validati sullo schema fornito possiamo deselezionare il controllo “Interrompi se rilevato un errore”.

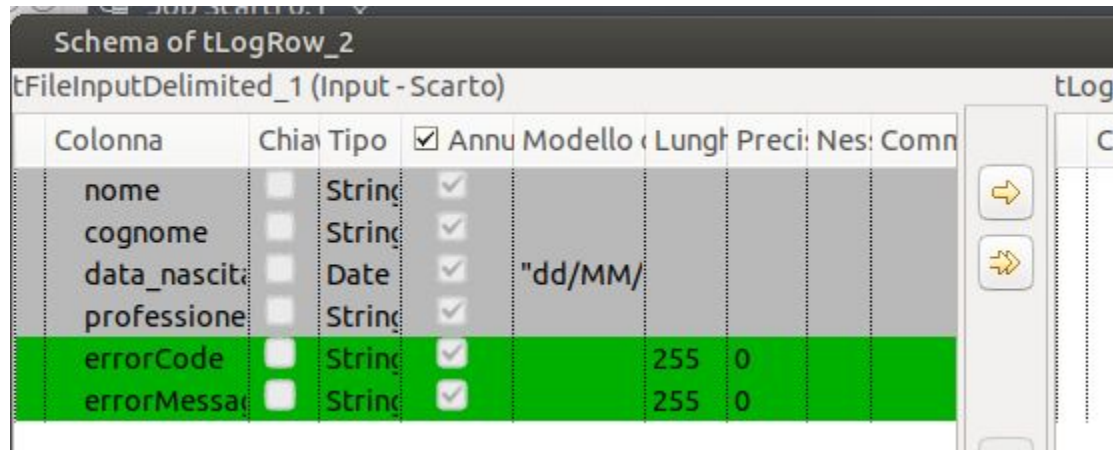
In effetti rilanciando il nostro job notiamo che questa volta non va in errore ed il nostro log mostra delle eccezioni relativamente ai record “scartati”.

In queste circostanze, se sono ammissibili gli scarti, potrebbe essere utile salvare in apposite strutture (file e/o database) i record che non sono stati processati correttamente al fine di poterne tenere traccia ed, eventualmente, apportare le opportune modifiche al dataset di input.

Aggiungiamo un nuovo componente **tLogRow** al nostro Job e colleghiamolo al componente che gestisce l'input tramite il flusso **scarto**. Per farlo selezionate il componente di input con il tasto **destro** -> **riga** -> **scarto** e selezionate nuovo componente **tLogRow**.



Eseguendo la nuova versione del nostro Job noterete che la nostra console non ha più errori ma i due componenti effettuano 2 stampe separate. Infatti i record conformi allo schema sono stati stampati dal componente **tRowMap** collegato mediante il flusso **principale** mentre i record che non sono stati parserizzati correttamente vengono passati al flusso **scarto** e stampate dal relativo componente. Inoltre potete notare che il componente legato al flusso di scarto ha delle colonne aggiuntive che non fanno parte dello schema principale.

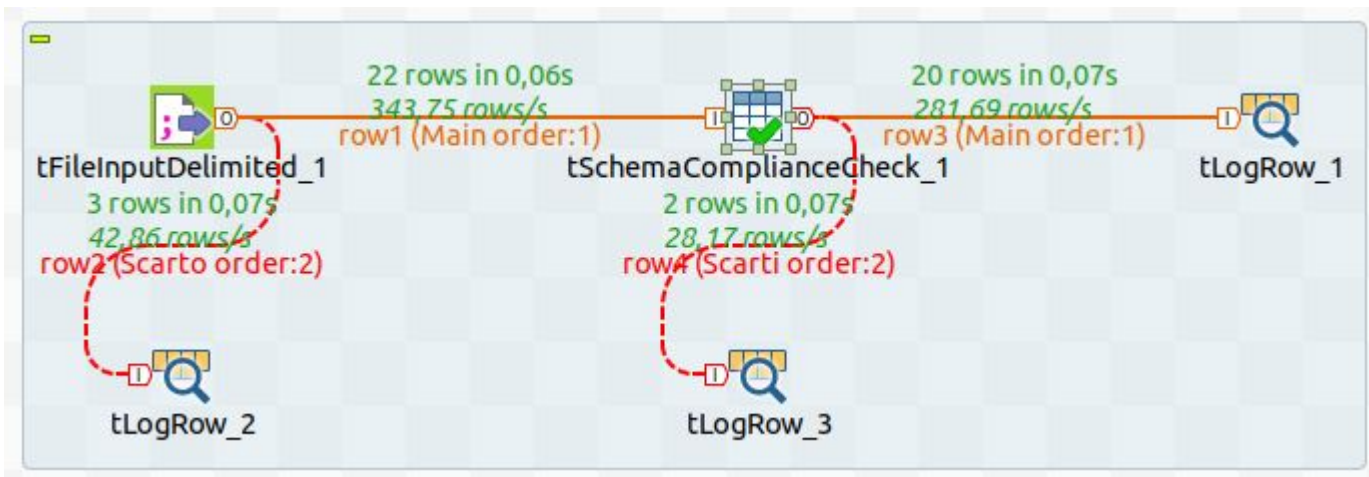


Schema of tLogRow\_2

tFileInputDelimited\_1 (Input - Scarto)

Colonna	Chia	Tipo	<input checked="" type="checkbox"/> Annu	Modello	Lung	Preci	Nes	Comm
nome	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
cognome	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
data_nascita	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>	"dd/MM/				
professione	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>					
errorCode	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		255	0		
errorMessage	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		255	0		

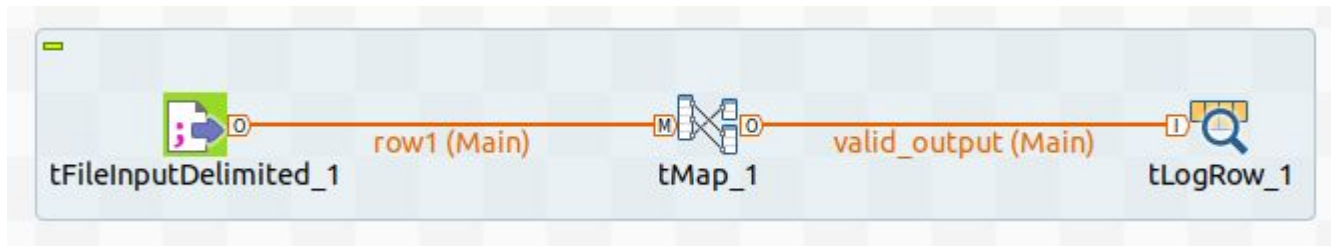
Un altro componente utile alla validazione dei dati **tSchemaComplianceCheck**.



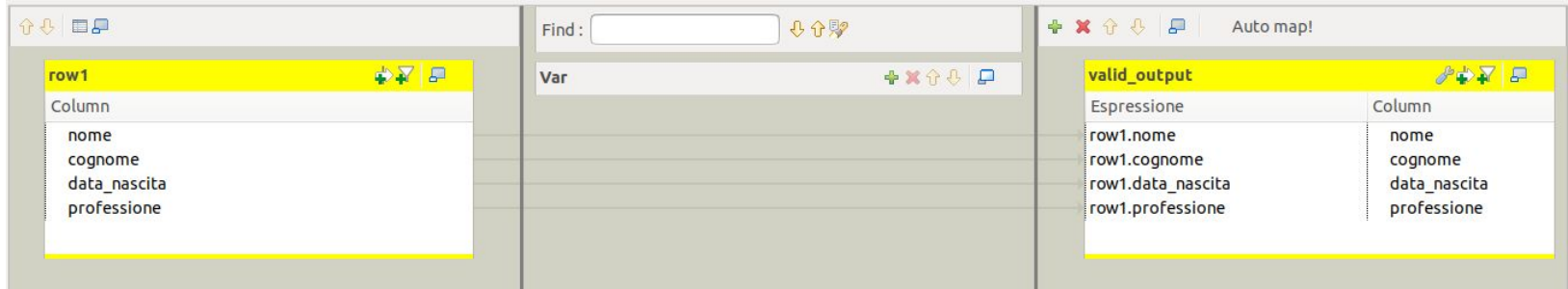


Anche un componente **tMap** può essere utilizzato per effettuare una verifica dei dati provenienti da una sorgente filtrando eventuali record. Questo comportamento può essere ottenuto con un **filtro** oppure con una **regola di validazione**.

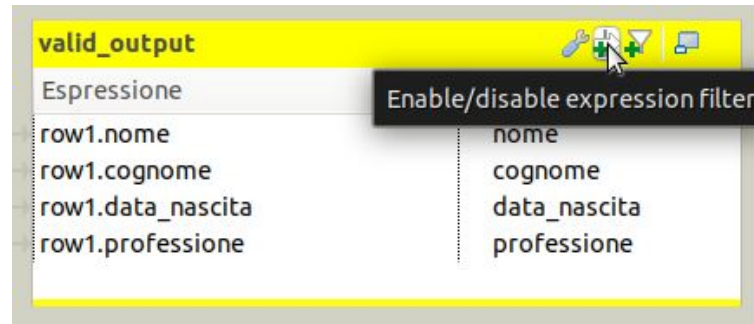
Creiamo un Job che abbiamo come input il nostro file **AST\_ETL/datasets/reject/personone.csv**. Colleghiamo il file di input ad un componente di tipo **tMap** e, quest'ultimo, ad un componente di tipo **tLogRow**.



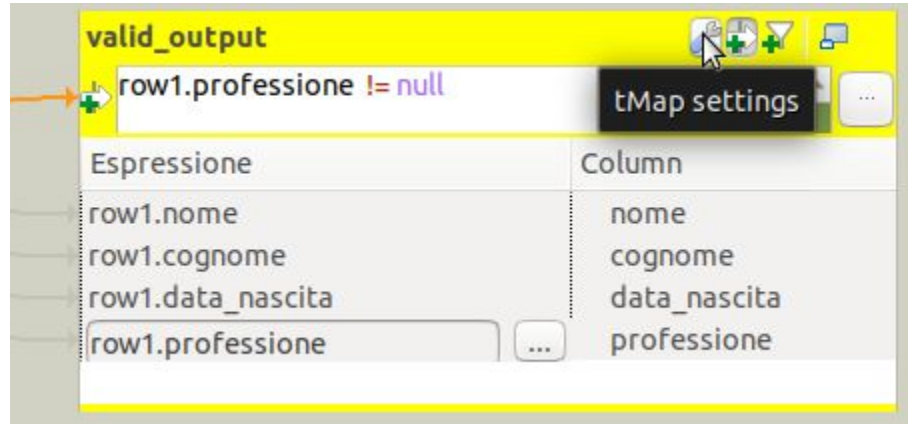
A questo punto apriamo il nostro componente **tMap** con doppio click sul componente in job designer e colleghiamo input ed output senza effettuare trasformazioni.



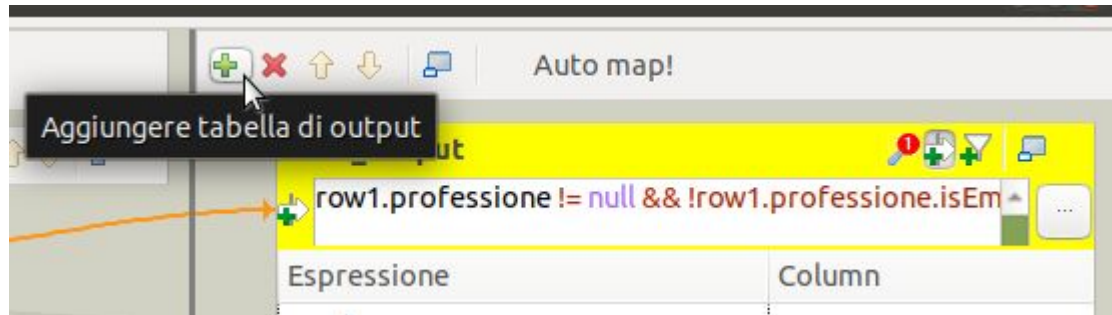
A questo punto, nella sezione output, abilitiamo il campo “expression filter”



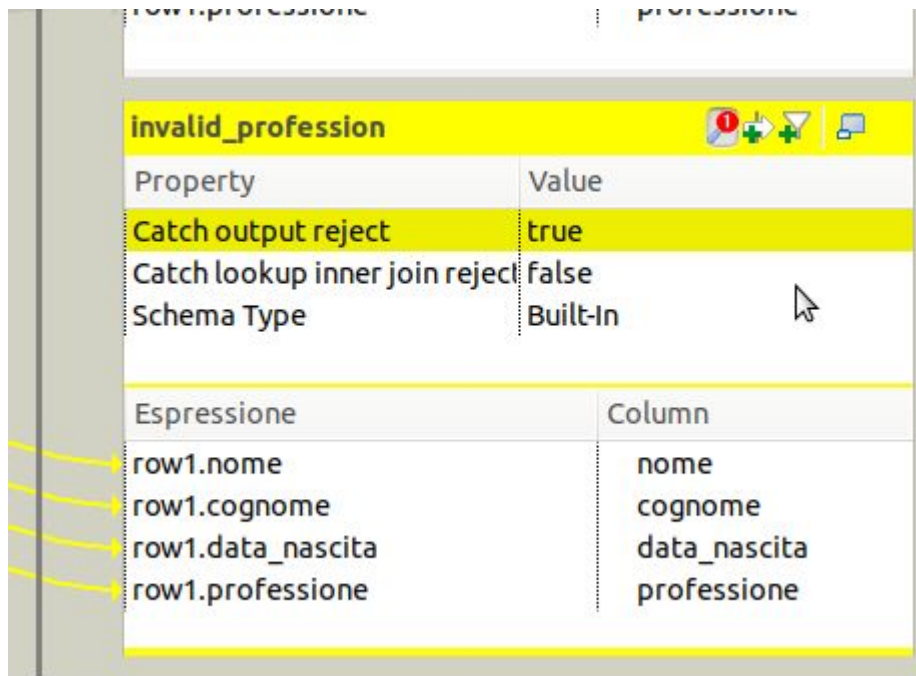
ed aggiungiamo il nostro filtro.



Aggiungiamo una nuova tabella di output nel nostro componente di tMap



Una volta aggiunta la nostra tabella di output selezioniamo il controllo **tMap setting**.



e selezioniamo il valore **true** per la proprietà **Catch output reject**

Nell'esempio precedente abbiamo, quindi, configurato un componente **tMap** per dividere l'input in due outputs separati in base ad una condizione (una sorta di IF ELSE).

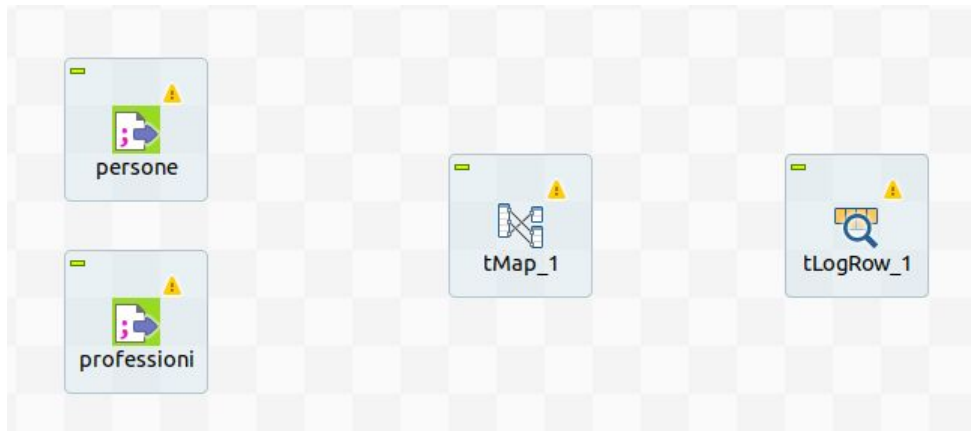
Tutte le righe che soddisfano la condizione verranno inviate al flusso **valid\_input** mentre, avendo abilitato l'opzione **Catch output reject** sul secondo output che abbiamo creato, le righe che non rispettano la condizione impostata vengono scritte su **invalid\_profession**.

Questa modalità può essere utilizzata per alimentare più di due output a seconda di una determinata condizione. Ricordatevi solo di effettuare il catch per quei record che non rientrano in nessuna delle condizioni configurate. Le condizioni dei vari output vengono controllate dall'alto al basso come in un costrutto if ... else.

Un altro modo per validare il contenuto di una colonna con un componente **tMap** potrebbe essere effettuato sulla base di valori contenuti in un altro input.

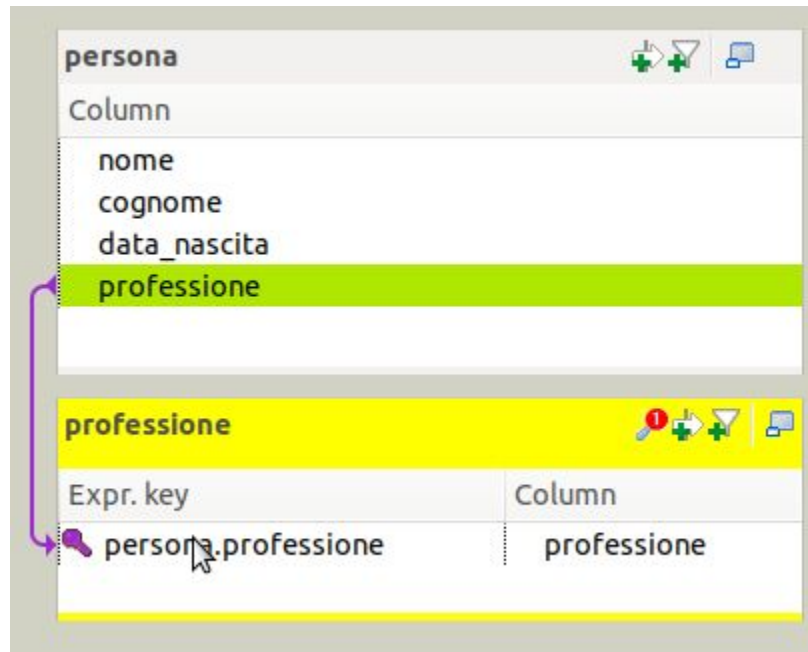
Ad esempio vorremmo recuperare tutte le persone, presenti nel file **persone.csv**, che svolgono una delle professioni presenti nel file **professioni.csv** (i due file sono nel path `AST_ETL/datasets/reject` del repository del corso).

Aggiungiamo all'interno del nostro Job i seguenti componenti: **tFileInputDelimited (persone)**, **tFileInputDelimited (professioni)**, **tMap** ed un **tLogRow**.

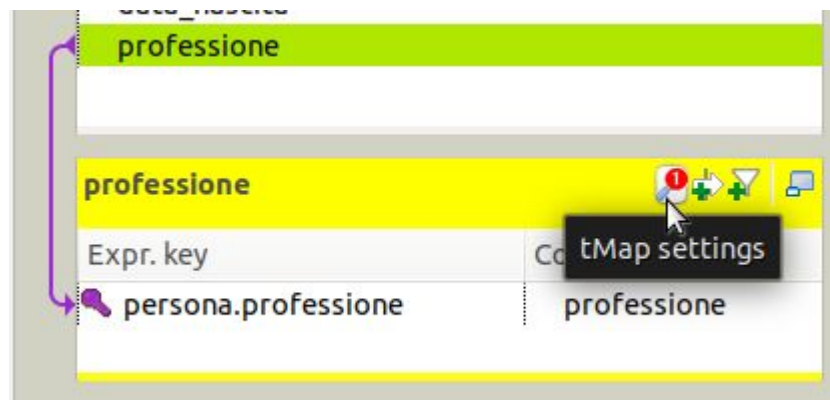


Collegiamo i nostri inputs al componente **tMap** ed apriamolo. Nella sezione relativa agli inputs vedremo che sono presenti entrambi gli schema dei nostri file.

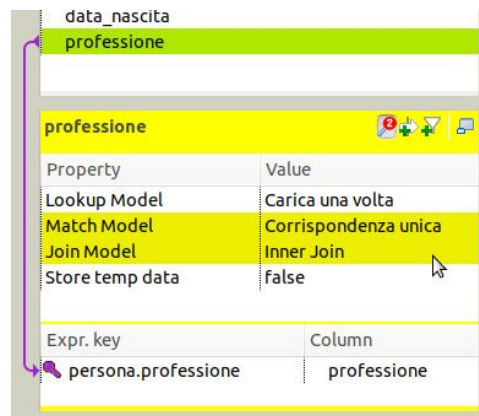
Collegiamo la proprietà professione di persone con la proprietà professione di professioni con un'operazione di Drag and Drop.



Selezioniamo il menu **tMap settings**







a questo punto selezionate per la voce **Join model** il valore **Inner join**









Aggiungiamo le due tabelle di output **valid\_profession** ed **invalid\_profession**

valid_profession		   	
Espressione	Column		
persona.nome	nome		
persona.cognome	cognome		
persona.data_nascita	data_nascita		
persona.professione	professione		

invalid_profession		   	
Espressione	Column		
persona.nome	nome		
persona.cognome	cognome		
persona.data_nascita	data_nascita		
persona.professione	professione		

Per la tabella di output **invalid\_profession** impostiamo, all'interno del menu **tMap setting**, la proprietà **Catch lookup inner join reject** a **true** e colleghiamo questo secondo output ad un altro componente **tLogRow**

In questo modo abbiamo messo in relazione i due inputs tramite il campo con valori comuni utilizzando una **inner join** per questo motivo gli unici records del file persone.csv restituiti come “validi” saranno quelli che hanno come valore del campo professione un dei valori presenti all'interno del file professioni.csv . La proprietà **Catch lookup inner join reject** presente sull'output con le professioni non valide fa sì che in questo flusso vengano passate tutti quei records che non hanno una corrispondenza con i valori di professioni.csv (tutti i record scartati dalla join)

Molte volte le validazioni da fare su un determinato campo sono più complesse di un semplice confronto tra valori o magari richiedono delle operazioni di trasformazione del dato piuttosto che la verifica di più parametri.

In questi casi potrebbe essere più comodo, e leggibile, l'utilizzo di un metodo Java da riutilizzare all'interno dei componenti di Talend.

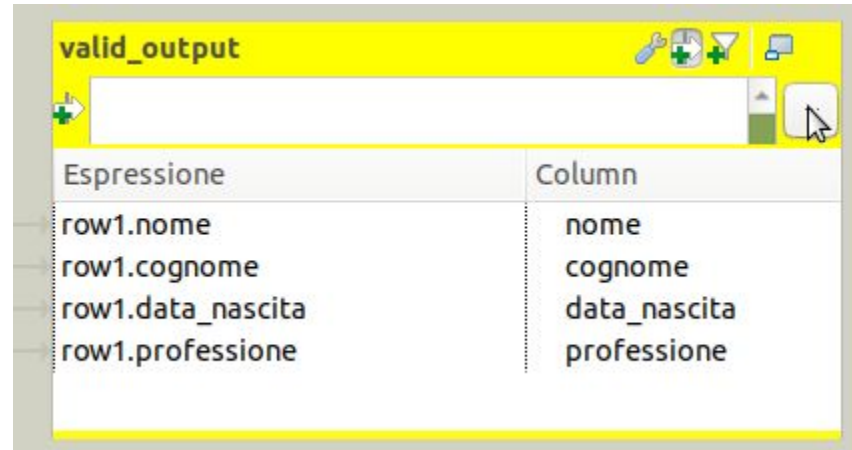
In questo caso ci tornano utili le **routine**. All'interno della view Repository, in **codice -> routine** creiamo la nuova routine **CheckRowUtils**. All'interno creeremo un metodo statico con la seguente firma

```
public static boolean check(String ... strings)
```

Attenzione al commento! Fate riferimento al commento presente sulla classe generata per sapere come scriverlo.

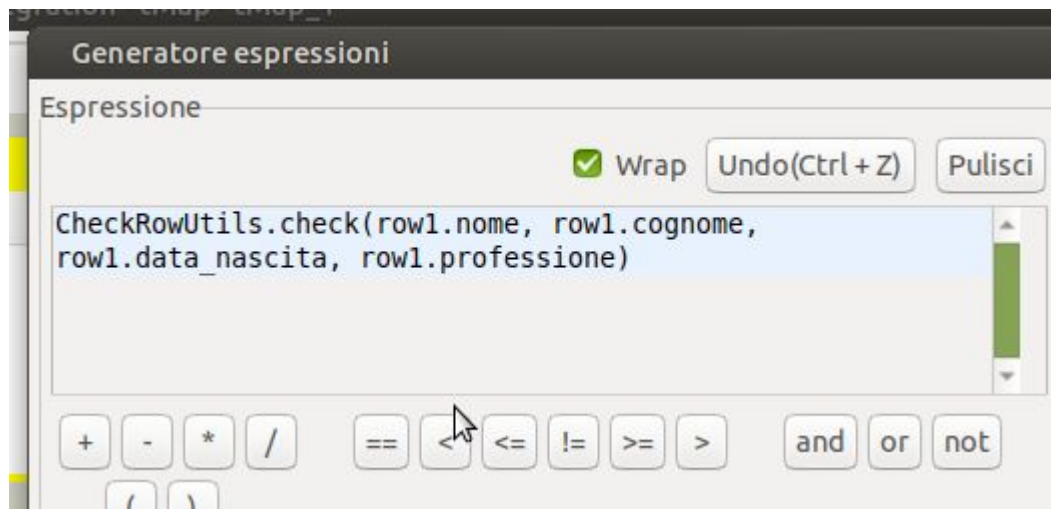
a questo punto riprendiamo il Job che abbiamo creato per filtrare il contenuto del file con il componente **tMap** (o creiamone uno con le stesse componenti).

Nel filtro del nostro tMap, però, non inseriamo il confronto del campo come fatto precedentemente ma selezioniamo il bottone accanto al campo (...)



Espressione	Column
row1.nome	nome
row1.cognome	cognome
row1.data_nascita	data_nascita
row1.professione	professione

Nella finestra di dialogo che si aprirà invochiamo il metodo che abbiamo creato precedentemente con la routine



salviamo e testiamo il nostro Job.

# “Mappare” i dati

Il componente **tMap** mette a disposizione diverse modalità per effettuare delle trasformazioni che ci consentono di mappare i valori di input nel formato di output desiderato.

Ad esempio, il componente ci consente di:

- aggiungere e rimuovere colonne;
- applicare regole di trasformazione ad una o più colonne;
- filtrare i dati di input ed output;
- mettere in relazione (join) i dati di più sorgenti e farli convergere all'interno uno o più outputs;
- dividere i dati di input in più output.

Il componente **tMap** è molto versatile e flessibile per questo motivo, delle volte, si è tentati di inserire quanto più codice possibile all'interno di un unico componente **tMap**.

Questa metodologia è altamente sconsigliata in quanto, all'aumentare della complessità, il codice diventa difficile da capire e mantenere. Per trasformazioni complesse si consiglia di utilizzare più componenti **tMap**.

Uno dei principali limiti di questo componente è dato dal fatto che le espressioni di trasformazione dell'output sono limitate ad una sola riga di codice. Questa limitazione, però, può essere superata dall'utilizzo delle **routine** o dall'utilizzo dell'operatore ternario che, se concatenato, può essere utilizzato per eseguire della logica condizionale.

Anche se limitato ad una singola riga di codice java, le espressioni possono contenere:

- Costanti;
- variabili di input;
- variabili di **globalMap** e **context**:

- variabili **tMap**;
- funzioni fornite da Talend;
- routines create dall'utente;
- metodi java custom;
- metodi provenienti da JAR inclusi nel progetto.

Utilizziamo le espressioni del componente **tMap** per trasformare l'output di un determinato campo.

Utilizziamo come input il file **/AST\_ETL/datasets/duplicati/clienti.csv** quindi, in accordo con il contenuto del file, creiamo un schema con i seguenti campi **id\_cliente**, **ragione\_sociale**, **sconto**. Aggiungiamo il nostro componente **tMap** e un componente che ci permetta di visualizzare l'output della trasformazione.

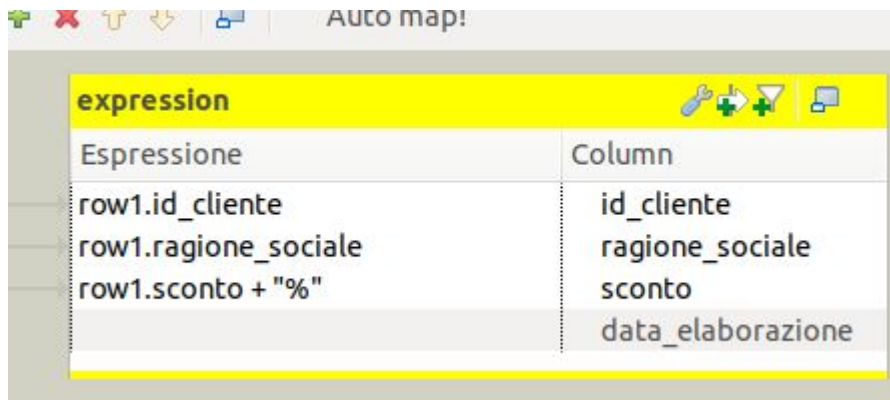
L'output che vorremmo ottenere alla fine della trasformazione è dato dai campi **id\_cliente** e **ragione\_sociale** così come inseriti all'interno del file di input.



Inoltre il nostro output dovrà mostrare il valore presente nel campo di input **sconto** seguito dal simbolo % ed un nuovo campo **data\_elaborazione** contenente la data corrente.

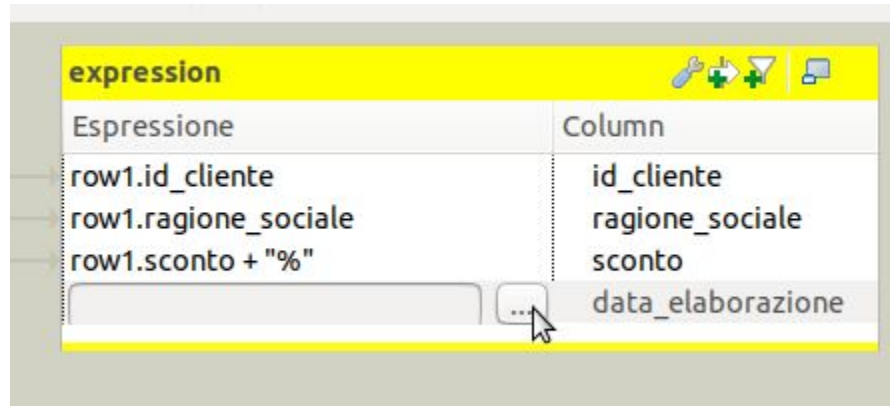
Apriamo il nostro componente **tMap** e riportiamo, se non presenti, i campi dell'input all'interno della tabella di output, poi aggiungiamo allo schema di output il nuovo campo **data\_elaborazione**.

Nell'espressione relativa al campo sconto concateniamo (Java) il simbolo % al valore presente.

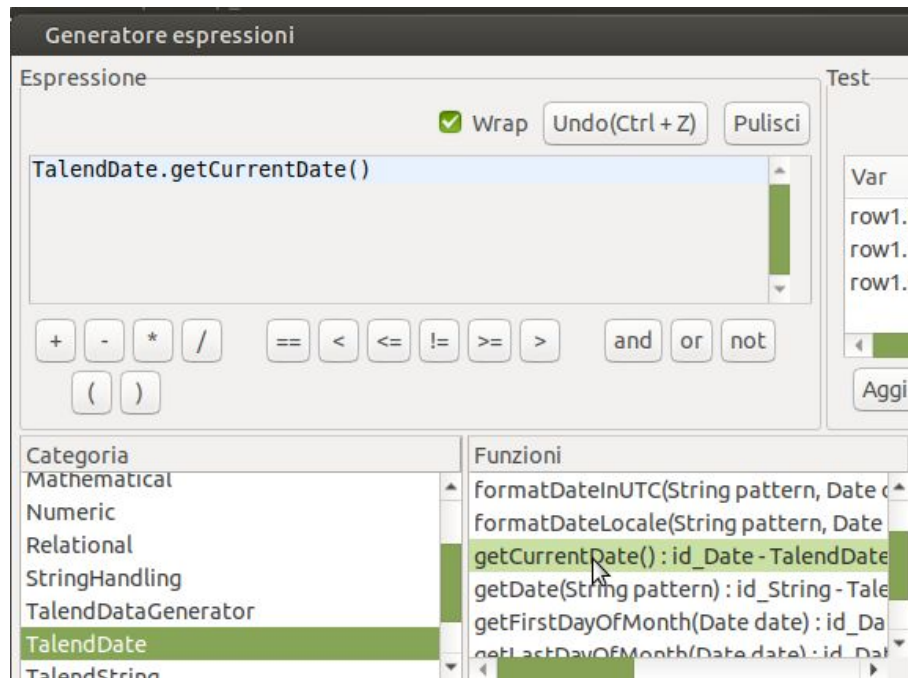


Espressione	Column
row1.id_cliente	id_cliente
row1.ragione_sociale	ragione_sociale
row1.sconto + "%"	sconto
	data_elaborazione

Ora selezioniamo il bottone per l'apertura del generatore di espressioni nella riga **expression** relativa al campo di output data\_elaborazione



e, nel generatore di espressioni selezioniamo la funzione offerta da Talend per ottenere la data odierna



Vi ricordo che in questo punto possiamo inserire una qualsiasi espressione java a patto che non ecceda la riga di codice. Se abbiamo la necessità di effettuare controlli complessi possiamo ricorrere alle **routines**. Naturalmente le funzioni invocate in questo punto possono far uso dei valori ottenuti in input da componente.

Auto map!

expression	
Espressione	Column
row1.id_cliente	id_cliente
row1.ragione_sociale	ragione_sociale
row1.sconto + "%"	sconto
TalendDate.getCurrentDate()	data_elaborazione

Lanciamo il job e valutiamo il contenuto dell'output.

E' possibile testare l'output prodotto da un'espressione con eventuali valori di prova

Generatore espressioni

Espressione

☒ Wrap Undo(Ctrl + Z) Pulisci

StringHandling.UPCASE(row1.ragione\_sociale)

Test

Var	Valore
row1.id_cliente	null
row1.ragione_	Ragione Sociale
row1.sconto	null

Test! Pulisci

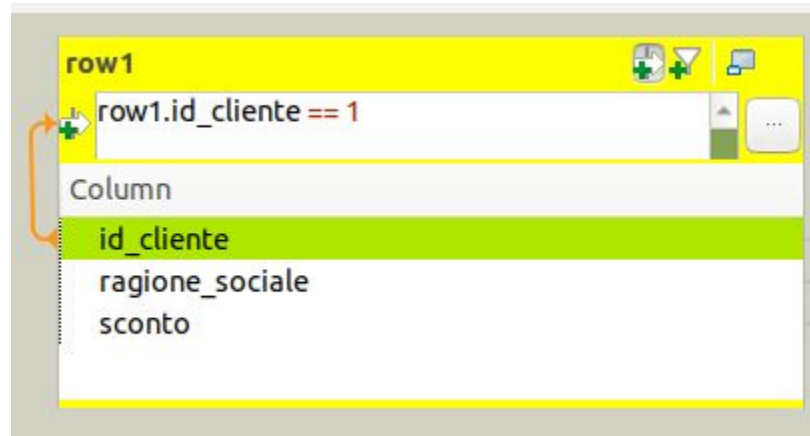
Aggiungere Cancellare

Come detto precedentemente non è possibile utilizzare espressioni su più linee, per questo motivo non è possibile effettuare nemmeno dei controlli condizionali con il costrutto **if... else**. Per fortuna Java ci viene in soccorso con l'operatore ternario che ci consente di scrivere un controllo condizionale equivalente su una sola riga.

Benché sia possibile utilizzare più operatori ternari annidati l'uso per condizioni molto complesse è sconsigliato per una questione di leggibilità del codice.

Il componente **tMap** mette a disposizione anche la possibilità di sfruttare delle variabili "intermedie". Queste variabili possono essere utilizzate per creare dei mapping complessi che utilizzano più variabili. Inoltre, come nel codice Java, è possibile utilizzare il valore di una variabile in un'altra variabile dichiarata successivamente.

Alcune volte potrebbe essere necessario filtrare l'input di un componente **tMap** in base ad un determinato valore. Questa operazione può essere fatta senza scomodare un'operazione di Join, aggiungendo un filtro alla tabella di input.



In questo caso l'input del componente verrà limitato ai soli record che rispettano la condizione.

# Utilizzare Java in Talend

Il componente **tJava** consente di eseguire una porzione di codice java all'interno del nostro job. Generalmente questo componente viene utilizzato per impostare variabili di contesto o globali prima delle fasi principali di elaborazione dei dati.

Inseriamo un componente tJava in un nuovo Job ed aggiungiamo del codice java per scrivere del testo verso lo standard output.



Se proviamo ad esaminare il codice generato di renderemo conto che quanto scritto all'interno del nostro componente è stato semplicemente aggiunto così com'è, per questo motivo dobbiamo sempre ricordarci di terminare i nostri statement con il ; altrimenti il nostro job andrà in errore.

In un qualsiasi componente Talend possiamo far riferimento alle variabili **context** e **globalMap**. Per testarne il funzionamento potremmo creare 2 componenti **tJava**, il primo legge una variabile dal context e ne sovrascrive il contenuto oltre ad inizializzare una variabile all'interno del globalMap con il seguente codice:

```
System.out.println("tJava_1");  
System.out.println(context.context_value);  
context.context_value = "NUOVO VALORE";  
globalMap.put("gmTestValue", "gmTestValue inizializzato");
```

Il secondo legge entrambi i valori



```
System.out.println("tJava_2");  
System.out.println("context.testValue è: "+context.context_value);  
System.out.println("gmTestValue è: "+(String) globalMap.  
get("gmTestValue"));
```

L'esempio precedente presuppone che sia presente una variabile nel contesto corrente denominata `testValue`. Inoltre il due componenti dovrebbero essere collegati con un flusso **onComponentOK**.

Le variabili **context** e **globalMap** sono salvate a livello globale, quindi è possibile fare riferimento ai propri valori da qualsiasi componente del progetto come `tMap`, `tFixedFlowInput` e `tFileInputDelimited`.

Il componente **tJavaRow** consente di eseguire del codice java per ogni record presente all'interno del flusso.

Creiamo un job come il seguente:



nei dettagli del componente **tJavaRow** modifichiamo lo schema replicando le informazioni provenienti dal flusso di input e aggiungendo i campi **eta** e **full\_name**.

Sempre dalle impostazioni di base del componente tJavaRow selezioniamo il bottone **Genera codice** che ci mostrerà il codice di base utilizzato per passare i valori di input nell'output.

# Contesti

Durante lo sviluppo di un software, capita sempre di dover impostare più variabili di connessione, le quali dovranno essere modificate quando il software sarà pronto ad essere rilasciato in Collaudo, e ancora una volta durante la fase di rilascio in Produzione.

Capita spesso che un software debba fare un passo indietro con conseguente modifica di tutte le variabili di connessione.

Distinguiamo quindi tre situazioni principali:

- **Locale:** solitamente le connessioni avvengono su localhost;
- **Collaudo:** le connessioni avvengono verso un server del tutto simile a quello di produzione, ma con dei dati adibiti alla fase testing

- **Produzione:** le connessioni avvengono verso il server su cui il cliente sta lavorando

Come si può risolvere, quindi, il problema di dover passare spesso da un ambiente all'altro?

Talend lo risolve grazie all'introduzione del concetto dei **Contesti**.

Ogni Contesto è visto come un insieme di variabili, capaci di assumere più valori, e quindi in grado di passare da un valore all'altro sotto istruzione dello sviluppatore.

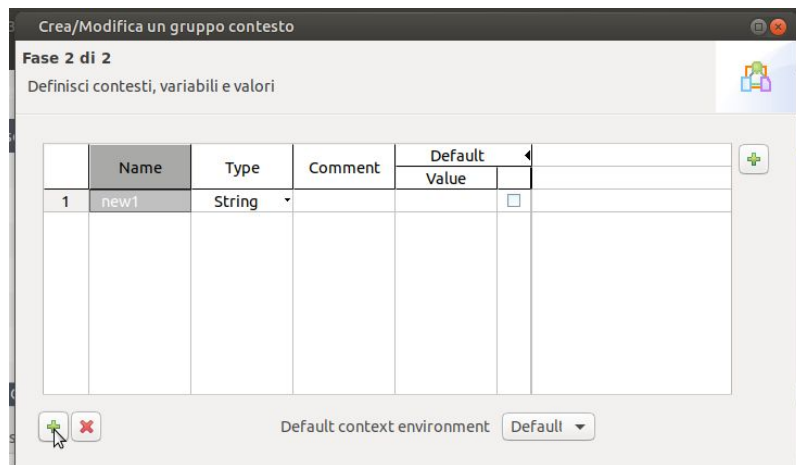
Talend permette di creare, modificare ed eliminare contesti e gruppi contesti.

Le variabili di contesto vengono utilizzate anche per passare parametri ad un Job, tramite la riga di comando o durante l'invocazione di un altro job.

Talend gestisce i **gruppi di contesto** essi possono essere visti come un insieme di variabili (generalmente) relazionate da utilizzare all'interno del progetto.

Creiamo il nuovo gruppo **GeneralSettings** di contesto facendo click con il tasto destro su **Contesti > Creare gruppo di contesto**.

Nella prima Scheda inserite il nome del contesto e selezionate **next**. A questo punto sarà possibile aggiungere nuove variabili al nostro contesto con il pulsante +

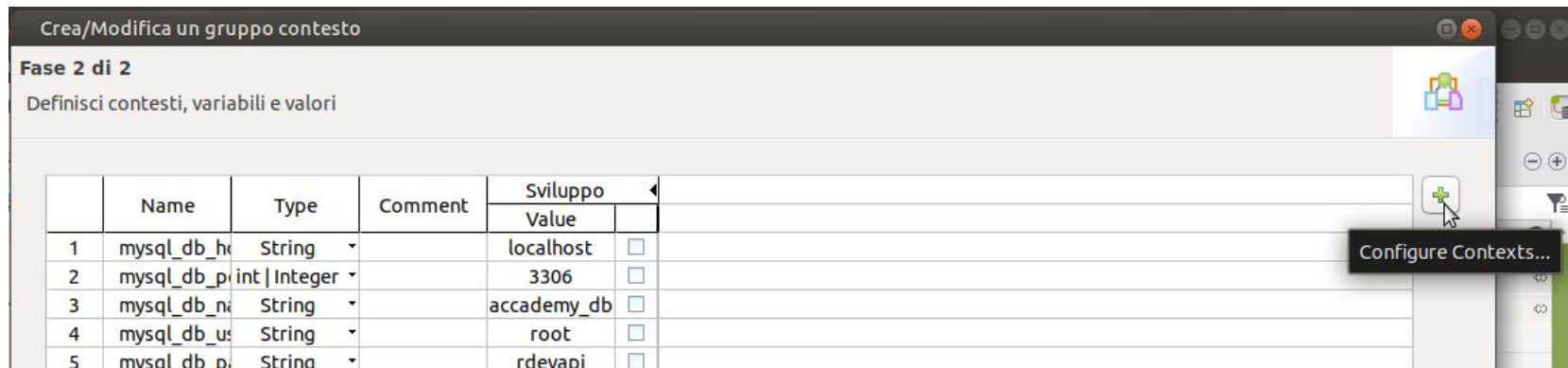


The screenshot shows a window titled "Crea/Modifica un gruppo contesto" with a sub-header "Fase 2 di 2" and the instruction "Definisci contesti, variabili e valori". The main area contains a table with the following structure:

	Name	Type	Comment	Default Value	
1	new1	String			<input type="checkbox"/>

At the bottom of the table, there is a "Default context environment" label and a dropdown menu currently set to "Default". To the right of the table, there is a green plus icon button. At the bottom left, there are icons for adding (+) and deleting (-) items.

Una volta aggiunte le variabili che ci interessano possiamo settare ulteriori “ambienti”



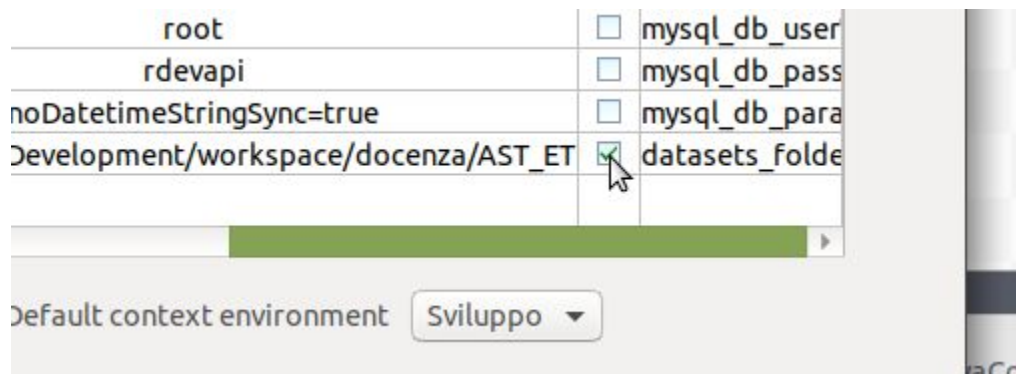
una volta inserito l’ambiente sarà necessario modificare i valori per l’ambiente specifico.

I valori inseriti potranno essere utilizzati all’interno del nostro progetto per parametrizzare i metadati oppure potranno essere acceduti dai nostri jobs a runtime variando l’ambiente a seconda delle nostre esigenze.

Oltre ad essere utilizzate per gestire le variabili legate ad uno specifico ambiente le variabili di contesto sono spesso utilizzate per definire dei valori comunemente utilizzati

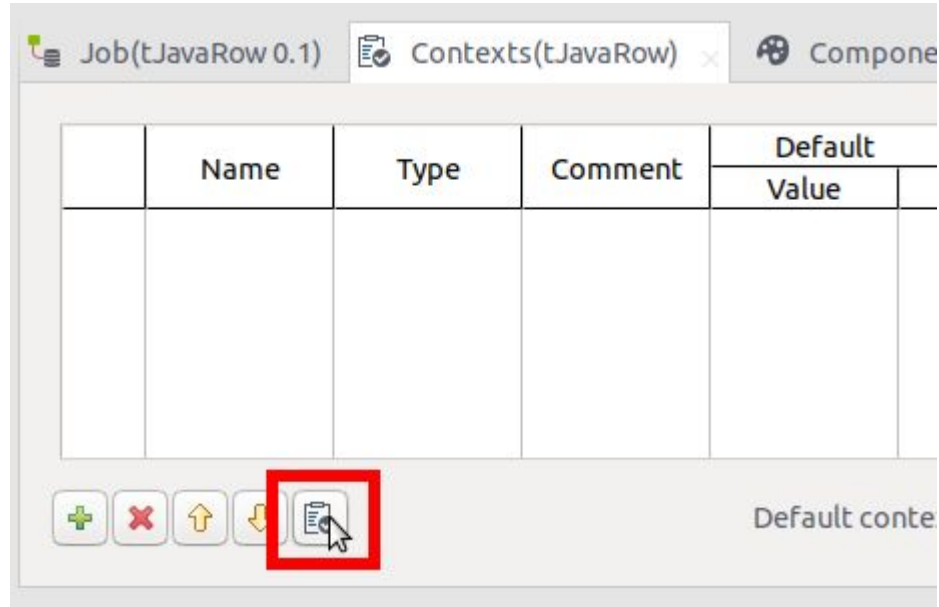
all'interno di un progetto. Come, ad esempio, le costanti, il path delle directory utilizzate per lo storage di determinati files, etc.

Selezionando il checkbox accanto ad una specifica variabile del context all'avvio di un job che ne fa uso verrà avviato un prompt in cui sarà possibile confermare o modificare il valore di quella specifica variabile.



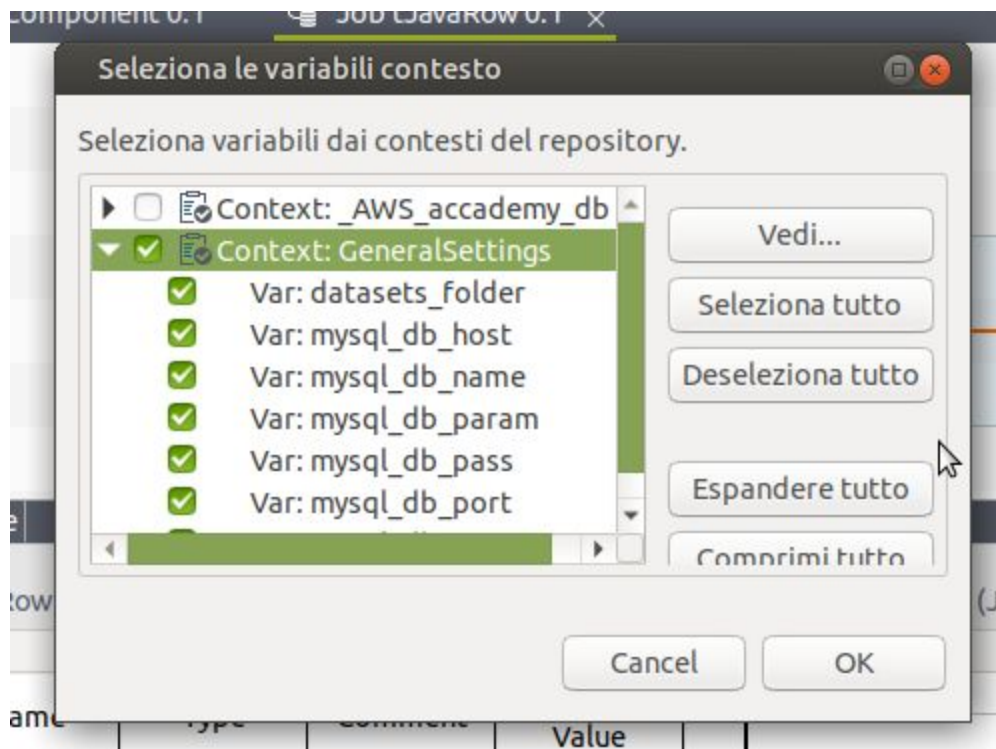
Un Job, di default, non utilizza uno specifico gruppo di contesto.

Per aggiungere un gruppo di contesto ad uno specifico Job è necessario aprire la view **Contexts** e selezionare l'icona per la selezione del contesto










A questo punto potrete selezionare i contesti che volete utilizzare nel job



prima di avviare un determinato Job è possibile variare l'ambiente e di conseguenza il valore delle variabili

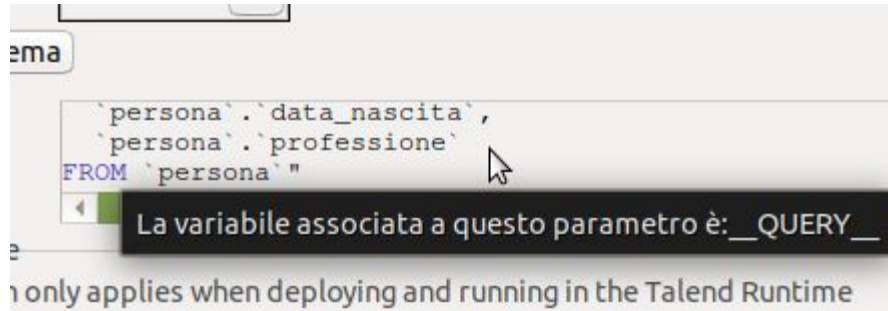
3	mysql_db_host	String		accademico.cnuaxlzzgmx5.eu-
4	mysql_db_name	String		accad
5	mysql_db_param	String		noDatetimeS
6	mysql_db_pass	String		KAef!H



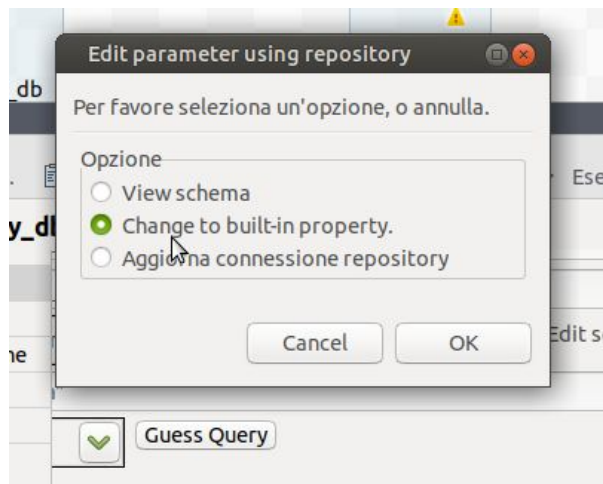
Default context environment 

# Lavorare con i Database

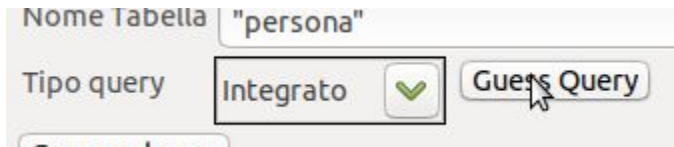
Abbiamo visto come, a partire da un componente **tMydbInput** opportunamente configurato, sia possibile recuperare le informazioni di una tabella del nostro database. In molti casi, però, non abbiamo bisogno di recuperare le informazioni dell'intera tabella ed è sicuramente più efficiente filtrare i dati direttamente dal database. Per recuperare un set di dati filtrato andiamo nelle impostazioni di base del componente e modifichiamo la query sql presente



Apriamo lo schema e cambiamo il tipo in **built-in**



A questo punto modifichiamo lo schema al fine di renderlo compatibile con le informazioni estratte dalla query modificata prima. Infine selezioniamo il bottone **guess schema**



A volte è più efficiente lasciar fare parte del lavoro al database come, ad esempio, il caso in cui il set di dati su cui dobbiamo lavorare può essere, in parte, già trasformato in fase di estrazione.

Pensiamo, ad esempio, al caso in cui si debbano recuperare informazioni da più tabelle oppure effettuare delle operazioni di raggruppamento. Recuperare le informazioni dal database per poi effettuare tali operazioni in fase di trasformazione, anche se possibile, sarebbe oneroso in termini di sviluppo e poco efficiente.

Anche in questi casi possiamo cambiare la query effettuata dal componente **tDbInput** e selezionare il bottone **guess schema** in modo da aggiornare lo schema integrato.

Talend eseguirà la query SQL definita nel componente ed allocherà i campi nell'ordine in cui vengono estratti all'interno dello schema

Naturalmente non è un invito a creare delle mega query super complesse ma solo cercare di sfruttare le potenzialità del DB quando possibile.

Le query SQL utilizzate dal componente **tDBInput** sono di tipo **String**, pertanto può essere manipolata come una qualsiasi altra stringa in Java.

I componenti di output servono ad inserire o modificare i dati di un sistema di destinazione essi sono collegati a una freccia di tipo Principale, associata, quindi, a uno Schema da cui prendono tutti i dati.

Per utilizzare il componente di output di MySQL è sufficiente selezionare la tabella che dev'essere soggetta a un inserimento (o aggiornamento) e trascinarla all'interno del Job interessato. A quel punto basterà selezionare nella lista il connettore **tMysqlOutput**.

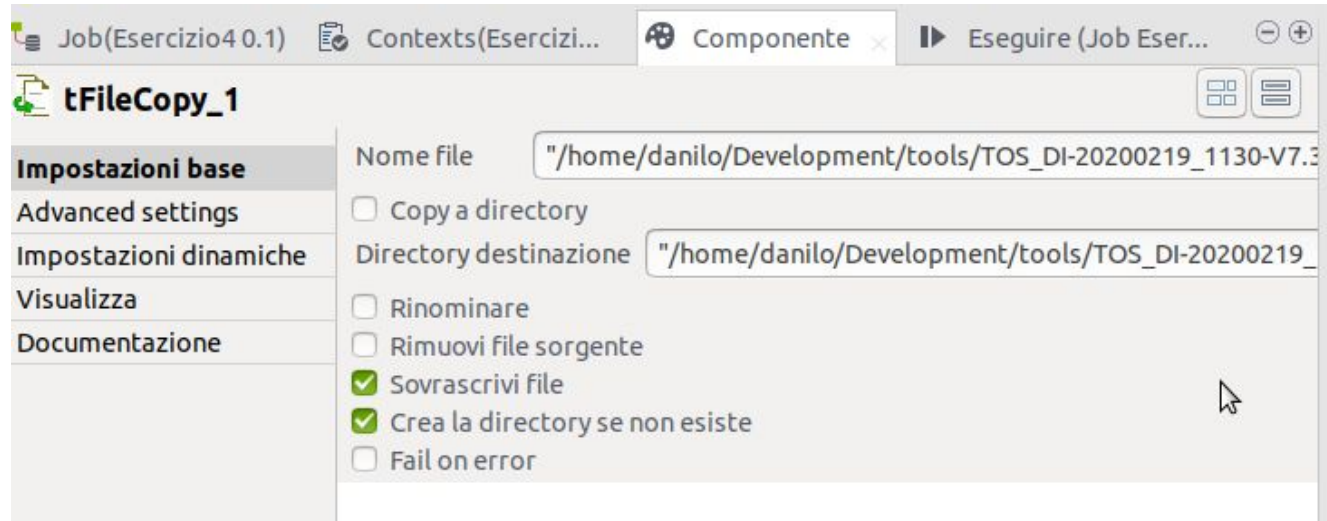
Sul componente tMysqlOutput è possibile selezionare il tipo di azione da svolgere:

- **Inserisci:** le righe vengono inserite nella tabella. In caso di violazione di vincoli di tipo UNIQUE o PRIMARY KEY viene sollevata un'eccezione;
- **Aggiorna:** le righe vengono modificate, a parità di campi chiave. Non vengono inseriti nuovi inserimenti;

- **Inserisci o aggiorna:** nella tabella vengono inserite nuove righe, oppure aggiornate, sempre sulla base dei campi chiave; Questo tipo di inserimento prova ad inserire una nuova riga, e, se non ci riesce, esegue un'operazione di UPDATE;
- **Aggiorna o inserisci:** come Inserisci o aggiorna, ma con un algoritmo diverso. Di fatti, Aggiorna o inserisci, prova prima ad eseguire un UPDATE, e, se nessuna riga viene aggiornata, esegue un inserimento;
- **Cancellare:** elimina i record in base alla chiave indicata;
- **Sostituire:** al contrario dell'Aggiorna, questa impostazione farà in modo da eliminare la riga che dev'essere aggiornata, e inserirne una nuova;
- **Inserisci o aggiorna su chiave duplicata o indici unici:** come aggiorna, ma prende in esame tutti i vincoli di unicità dichiarati nella tabella;
- **Insert Ignore:** inserisce nuovi record, e, nel caso di chiavi duplicate, non fa nulla.

# Lavorare con i File

Talend mette a disposizione diversi componenti per la gestione dei files.  
ad esempio per copiare un file potremmo utilizzare il componente **tFileCopy**. Questo componente consente di effettuare anche operazioni come rinominare un file piuttosto che spostarlo a seconda delle impostazioni di base settate





**tFileDelete** viene utilizzato per eliminare i file

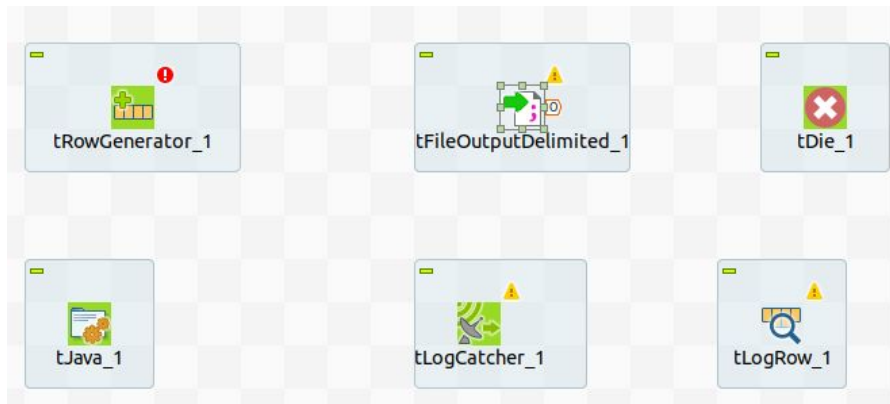
# Gestione degli errori

In determinate circostanze potrebbe essere necessario terminare l'esecuzione del nostro Job e comunicare informazioni sulla causa che ha generato l'errore.

Questo meccanismo viene gestito principalmente da due componenti: **tDie** utilizzato per bloccare il Job e lanciare l'eccezione e **tLogCatcher** che intercetta eventuali errori lanciati da **tDie** (oltre ai warning di **tWarn**).

Per testarne il funzionamento creiamo un Job con i seguenti componenti:

- **tRowGenerator**
- **tJava**
- **tFileOutputDelimited**
- **tDie**
- **tLogCatcher**
- **tLogRow**

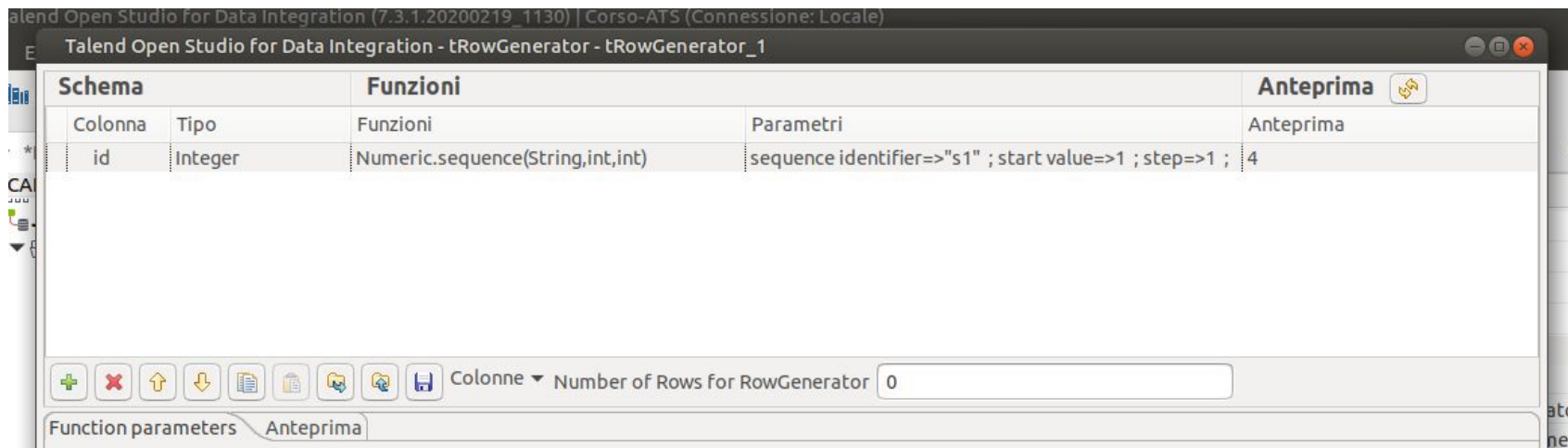


Collegiamo il **tRowGenerator** al **tFileOutputDelimited** tramite **Riga > Principale**, poi colleghiamo **tFileOutputDelimited** al componente **tDie** tramite **Attivare > Eseguire se**.

Collegiamo il componente **tRowGenerator** al **tJava** tramite **Attivare > OnSubjobOk** e successivamente collegare il componente **tLogCatcher** al **tLogRow** tramite **Riga > Principale**

A questo punto non ci resta che configurare il nostro Job affinché possa catturare il messaggio lanciato dal componente **tDie**

Configuriamo il componente **tRowGenerator** come segue



Assicuriamoci che il componente **tFileOutputDelimited** abbiamo un file configurato nelle **impostazioni base** del componente.

Ora configuriamo la condizione di attivazione (**IF**) presente tra il componente **tFileOutputDelimited** e **tDie**. Questa è la condizione che attiverà il nostro componente **tDie** se verificata. Abbiamo configurato il componente **tRowGenerator** affinché generi 0 record. Possiamo utilizzare questa condizione:

```
((Integer)globalMap.get("tRowGenerator_1_NB_LINE")) <= 0
```

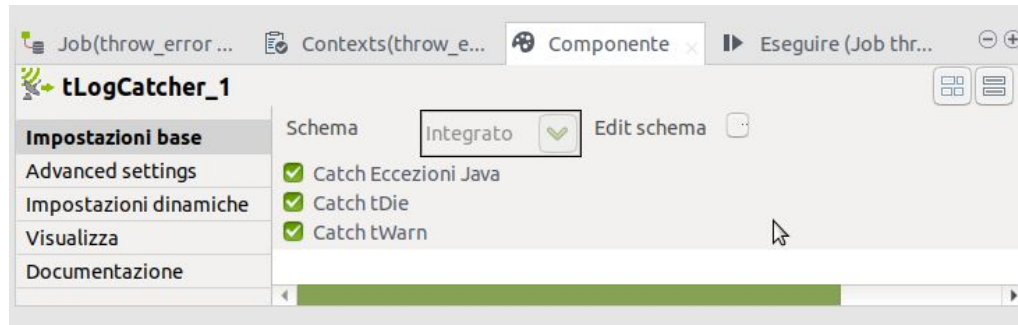
Configuriamo il componente **tDie**



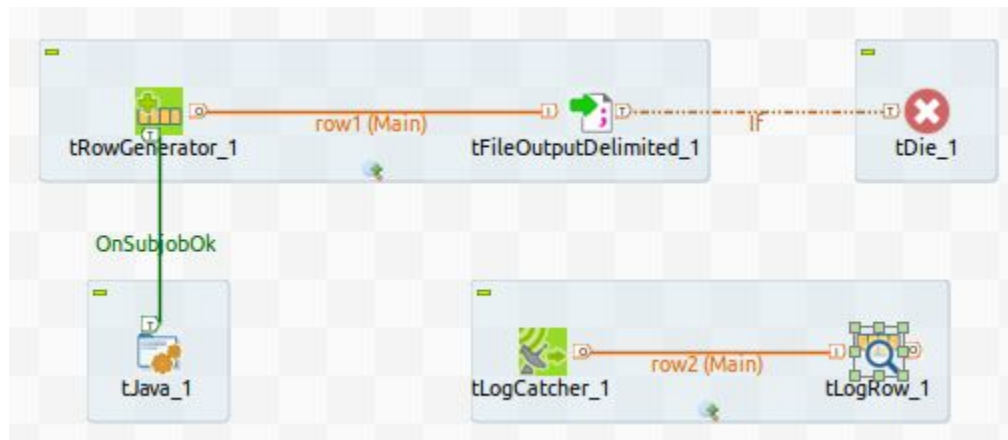
Nel componente **tJava** aggiungiamo un `System.out.println()` che ci stampi il numero di record generato

```
System.out.println("Il numero di righe generato è: " +  
((Integer)globalMap.get("tRowGenerator_1_NB_LINE")));
```

ed assicuriamoci che il componente **tLogCatcher** sia configurato per catturare i messaggi lanciati da **tDie**.



A questo punto il nostro Job dovrebbe essere simile al seguente



Eseguendo il Job con la configurazione precedente verrà attivato il componente **tDie**, effettuata la stampa dell'errore tramite il componente **tLogCatcher** e terminata l'esecuzione

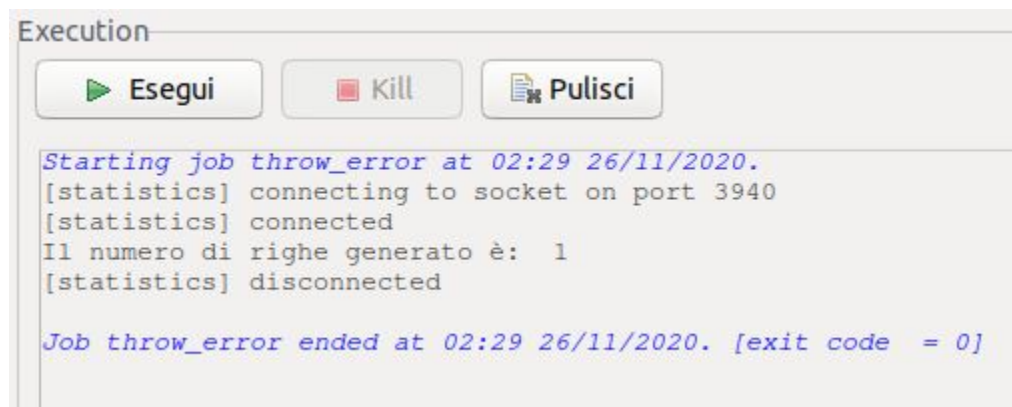
```
Execution
[Esegui] [Kill] [Pulisci]

Starting job throw_error at 02:29 26/11/2020.
[statistics] connecting to socket on port 3947
[statistics] connected
Non è stata generata alcuna riga
+-----+-----+
|               #1. tLogRow_1               |
+-----+-----+
| key      | value                                |
+-----+-----+
| moment   | 2020-11-26 02:29:20                 |
| pid      | Pe0Wuv                              |
| root_pid | Pe0Wuv                              |
| father_pid | Pe0Wuv                             |
| project  | CORSO_ATS                           |
| job      | throw_error                         |
| context  | Default                             |
| priority | 5                                   |
| type     | tDie                                |
| origin   | tDie_1                              |
| message  | Non è stata generata alcuna riga    |
| code     | 1                                   |
+-----+-----+

[statistics] disconnected

Job throw_error ended at 02:29 26/11/2020. [exit code = 1]
```

Nel caso in cui provassimo a configurare il componente **tRowGenerator** affinché generi almeno un record, invece, l'esecuzione seguirà un flusso differente (verrà attivato il componente **tJava**) che stamperà il numero di record generati.



The screenshot shows a window titled "Execution" with three buttons at the top: "Esegui" (Run) with a green play icon, "Kill" with a red stop icon, and "Pulisci" (Clear) with a document and trash icon. Below the buttons is a text area containing the following log output:

```
Starting job throw_error at 02:29 26/11/2020.  
[statistics] connecting to socket on port 3940  
[statistics] connected  
Il numero di righe generato è: 1  
[statistics] disconnected  
  
Job throw_error ended at 02:29 26/11/2020. [exit code = 0]
```

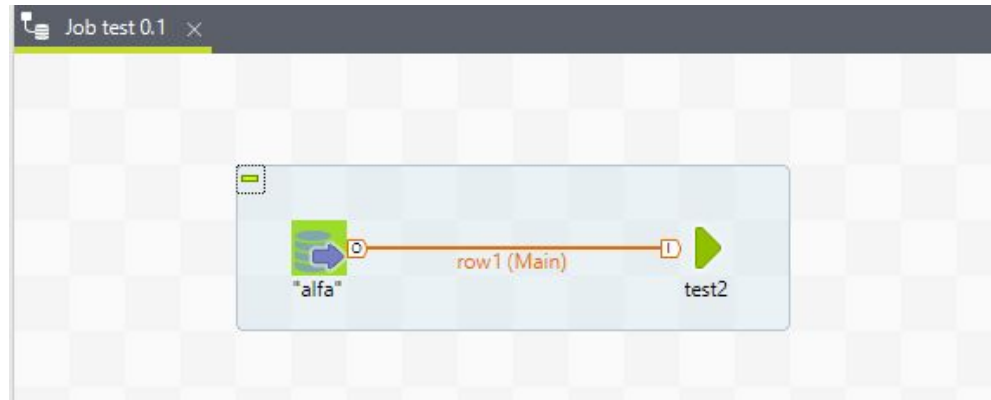


# Modularizzare il progetto

Un Job può essere anche invocato da un altro Job, passando ad esso parametri attraverso il contesto.

In questo modo è possibile rendere tutto il codice modulare, creando un Job principale per gestire l'intero flusso, e tanti piccoli Job dedicati ad attività minori.

Nell'esempio di seguito, è visibile un componente di input che estrae informazioni da una tabella chiamata «alfa», per poi passare le righe a un job chiamato «test2»




Cliccando sulla freccia, sarà possibile analizzare il suo schema. In questo caso, è lo stesso schema del componente di input, ossia, la struttura della tabella «alfa»

Contexts(test) Componente Esegui (Job test)

Edita schema

Schema from tMySQLInput\_1 output

Colonna	Db Column	C...	Tipo	Tipo DB	<input checked="" type="checkbox"/>	A...	Modello...	Lun...	Pre...	N...	Co..
 id	id	<input checked="" type="checkbox"/>	int	INT	<input type="checkbox"/>			10	0		
email	email	<input type="checkbox"/>	Stri...	VARC...	<input type="checkbox"/>			255	0		
var	var	<input type="checkbox"/>	Stri...	VARC...	<input type="checkbox"/>			255	0		

Cliccando sul job sarà possibile analizzare tutti i suoi dati. È importante spuntare la checkbox **Trasmetti intero contesto**, nonché indicare quali campi dovranno essere passati al job, che in questo caso è **test2**. I parametri da passare a test2 dovranno essere indicati manualmente, così come il loro valore dovrà essere impostato manualmente.

Il nome del parametro è **id**, mentre il valore è **row1.id**. Per **row1.id** si intende «il campo **id** proveniente dalla freccia **row1**».

Nel campo **Valori** è possibile eseguire operazioni o invocare metodi.

Schema Integrato ▼ Edita schema ... Sincronizza colonne Copia schema job figlio

☐ Utilizza job dinamico

Job test2 ... Versione Latest ▼\* Contesto Default ▼\*

☐ Use an independent process to run subjob

☒ Finisci se il figlio fa errori

☒ Trasmetti intero contesto

Parametri contesto

Parametri	Valori
id	row1.id

Il job **test2**, per utilizzare i parametri passatigli, dovrà avere nel contesto la variabile dichiarata nello schema del Job chiamante. La variabile sarà accessibile tramite la nomenclatura **context.id**

# Esportazione e deploy

I progetti di Talend possono essere estratti e importati.

Per estrarre un progetto è sufficiente cliccare su:

**File > Export**

Successivamente bisogna selezionare:

**General > Archive File**

Nella schermata successiva bisogna spuntare la casella che rappresenta il progetto da estrarre, e indicare la cartella in cui posizionare il file compresso.

Per «Deploy» s'intende la generazione del file eseguibile.

Il deploy, in Talend, si esegue cliccando col tasto destro del mouse sul Job principale (ossia il Job che deve essere eseguito per primo), e selezionando la voce Build Job.

È ora necessario indicare quale contesto si intende utilizzare, il tipo di estensione (.bat per Windows, .sh per Linux).

Cliccando Finish sarà generata la cartella contenente l'eseguibile e tutte le librerie necessarie al suo corretto funzionamento.

# Esercizio

Data una tabella clienti, contenente i campi:

idCliente | ragioneSociale | sconto

Modificare tutti i record della tabella, rendendo il campo ragioneSociale maiuscolo, eliminando le righe duplicate e raddoppiando la percentuale di sconto.

In caso di ragioneSociale duplicata, eliminare la riga con sconto maggiore.