UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Progetto e Sviluppo di un'Applicazione Java per l'Assistenza alle Vendite in Ambito Ortofrutticolo tramite la Piattaforma Pentaho

Stefano Predonzani

Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2016/2017

RINGRAZIAMENTI

Ringrazio l'Ing. Riccardo Martoglia, mio relatore, per la disponibilità e i consigli ricevuti per questo lavoro di tesi.

Ringrazio Chiara per il supporto dato durante le ultime settimane.

Infine un ringraziamento va ai mia genitori che hanno sempre creduto in me e mi hanno dato la possibilità di arrivare fino a questo punto.

PAROLE CHIAVE

java
pentaho
database
xml
dati

Sommario

Introduzione 1
Capitolo 1 4
Tecnologie utilizzate 4
1.1 Situazione iniziale 4
1.2 Introduzione Pentaho-Kettle 5
1.3 Parti che costituiscono Pentaho-Kettle 8
1.3.1 Trasformazioni 8
1.3.2 Jobs 12
1.3.3 metadati di Trasformazione o di Job 16
1.3.4 Connessione al database 16
1.3.5 Repositories 18
1.4 Parametri e variabili 20
Capitolo 2 22
Progetto AssistenteVendite 22
2.1 Estrazione dati 22
2.1.1 Formato XML e XPath 22
2.1.2 Formato TXT e CSV 24
2.2 Librerie adoperate 25
2.2.1 Librerie per l'elaborazione di file XML 26
2.2.2 Librerie Kettle 29
2.3 Salvataggio dei dati sul database 31
Capitolo 3 33
Progettazione 33
3.1 Requisiti funzionali 33
3.1.1 Generazione file dati 33

3.1.2 Trasformazione dati 34
3.1.3 Salvataggio dati nella tabella di appoggio 34
3.1.4 Allineamento 35
3.2 Requisiti non funzionali 35
3.3 Progettazione database 36
3.4 Activity Diagram 38
Capitolo 4 40
Implementazione 40
4.1 Struttura del progetto 40
4.1.1 main.java 42
4.1.2 Classe CustomTask 42
4.1.3 Classe JobStarter 43
4.1.4 Classe astratta AbstractJob 44
4.1.5 Interfaccia IPlugin 48
4.2 Trasformazioni e Job 53
4.2.1 Job assistenzaMarketnews 53
4.2.2 Job AssistenzaArticoli 56
4.3 Codice SQL 58
Capitolo 5 60
Test del sistema 60
Conclusione e sviluppi futuri 62
Bibliografia 63
Sitografia 63

Elenco delle figure

Figura 1.1: virtuale Vs fisico 8
Figura 1.2 esempio di trasformazione 9
Figura 1.3 esempio di esecuzione di un job 13
Figiura2.1: esempio JTidy usato su XML 27
Figura 2.2: esempio utilizzo jsoup per estrarre tag 29
Figura 2.3: Esempi di API Kettle per la chiamata di un job 30
Figura3.1: diagramma ER schema PRODOTTI 36
FIgura 3.2: diagramma ER schema ARTICOLI 37
Figura 3.3: Activity Diagram caso generale 39
Figura 4.1: Struttura del progetto 41
Figura 4.2 Job assistenzaMarketnews 54
Figura 4.3 Trasformazione caricaMarketnews 55
Figura 4.4 Trasformazione VeronaMercato 56
Figura 4.5 Job AssistenzaArticoli 56
Figura 4.6 Trasformazione caricaArticoli 57

Introduzione

Nell'epoca moderna l'accesso veloce alle informazioni è di vitale importanza per poter gestire al meglio tutti gli aspetti all'interno di una ditta. Negli ultimi 50 anni si è andati a sviluppare quelle che vengono chiamate tecnologie per la "data integration", le quali permetto il recupero, la gestione e il salvataggio delle informazioni. Di queste l'approccio più promettente è quello ETL, il quale è stato implementato da Pentaho nel tool Kettle.

Il lavoro di tesi svolto all'interno di Vignola-import, una ditta che esporta prodotti ortofrutticoli, la quale vuole tenersi al passo coi tempi per quanto riguarda il sistema informatico. Ha portato alla creazione di un sistema di assistenza alla vendita, in grado di recuperare le informazioni sparse sul web e raggrupparle tutte all'interno di un database. In particolar modo si è andato ad approfondire il comportamento del tool Kettle, il quale è un ottimo tool ETL Open source.

La tesi è strutturata su due parti principali, costituiti da 4 capitoli:

1. *Caso di studio* comprendente:

   Capitolo 1 - Tecnologie utilizzate, in cui vengono studiate le tecnologie ed analizzati gli strumenti messi a disposizione AssisteVendite;

   Capitolo 2 - Progetto AssistenteVendite, per la descrizione delle scelte comportamentali riguardanti l'estrazione, la trasformazione e il salvataggio dei dati.

2. *L'applicazione* comprendente:

   Capitolo 3 - Progettazione, prevede la raccolta dei requisiti funzionali da sviluppare e la progettazione del sistema senza entrare in dettagli implementativi.

   Capitolo 4 - Implementazione, descrittivo della fase implementativa del progetto.

Parte I

Il caso di studio

Capitolo 1

Tecnologie utilizzate

In questa parte vengono descritti in modo completo gli strumenti usati da AssistenteVendite per la realizzazione del progetto.
A partire dalla situazione iniziale, alla quale sono state aggiunte le funzionalità richieste, si elencano i requisiti richiesti e le tecnologie utilizzate, per raggiungere tale scopo.

1.1 Situazione iniziale

Il progetto è ideato da una ditta import-export di prodotti ortofrutticoli, la quale vuole avere uno strumento in grado ti tener traccia, dell'andamento di mercato, dei propri prodotti. Tale sistema comprendeva all'inizio solo le funzionalità di download del file XML contenente i dati e visualizzazione dello stesso, inoltre era supportato un solo sito web. Questo non permetteva di avere un'ampia visione temporale dei dati, perciò è nesessario, con l'ausilio di un database, un sistema in grado di elaborare le informazioni e salvare, ampliando allo stesso tempo i siti web da cui reperirle. Oltre a questo si vuole avere una più rapida visualizzazione di notizie riportate da diverse testate giornalistiche riguardanti il settore agricolo, in modo da essere sempre aggiornati su gli ultimi avvenimenti. Infine si voleva un sistema per il salvataggio dei dati riguardanti container spediti per poter tenere traccia dello stato degli stessi.
Per implementare tutte queste nuove funzionalità si è deciso di sfruttare i software di tipo ETL (extract, transform, load), che è una famiglia di programmi sviluppati appunto per estrarre, trasformare e salvare informazioni da varie sorgenti. In particolare si è deciso di utilizzare la piattaforma Pentaho-Kettle una suite open source ETL.

Mentre per il salvataggio dei dati come già anticipato è stato scelto un database relazionale PostgreSQL e come linguaggio di programmazione si è deciso di utilizzare Java, il quale ha già un supporto per la suite pentaho.

1.2 Introduzione Pentaho-Kettle

In questa parte andremo ad esporre i principi e la base di Pentaho-Kettle [1], il cui nome ora è stato cambiato in “pentaho data integration”, il quale è un tool ETL, inoltre andremo ad analizzare le varie differenze tra le diverse strategie di “data integration”.

Pentaho-Kettle è un tool ETL usato per fare data integration, è open source dal 2005 ed è scritto nel linguaggio di programmazione Java, il quale gli permette di avere una portabilità maggiore su vari sistemi operativi. È molto completo e copre tutte le funzionalità di un ETL, in più fa anche il passo di “transport”, quindi a volte viene utilizzato l'acronimo ETTL. È basato sui seguenti principi: facilità di sviluppo, evita la necessità di codice personalizzato, tutte le funzionalità sono disponibili nell'interfaccia utente, nessun limite sui nomi, trasparenza, data paths flessibili e mappa solo i campi interessanti. I componenti principali che implementa sono: Spoon (disegno grafico dei passi dell'ETL), Pan (esecuzione da linea di comando delle trasformazioni), Kitchen (esecuzione dei job), Carte (console per l'esecuzione remota).

ETL è uno dei principali approcci alla data integration ed è “l'insieme dei processi per ottenere dati dai sistemi OLTP ed inserirli all'interno di un data warehouse”. All'inizio del'ETL poteva essere una definizione valida, ma ora ad esempio i dati possono provenire da siti web, database di posta elettronica, fogli di calcolo, database personali, ecc. In aggiunta a ciò ETL non è utilizzato solo per caricare i dati ma può avere molte altre situazioni di utilizzo come ad esempio generare fogli di calcolo o valutare i clienti utilizzando modelli di data mining. I passaggi principali ETL, tuttavia, possono essere raggruppati in tre sezioni:

1. *Extract*: Tutti i processi necessari per connettersi a diverse sorgenti di dati, estrarre i dati da queste sorgenti e renderli disponibili per i successivi passi di elaborazione,
2. *Transform*: Qualsiasi funzione applicata ai dati estratti, tra l'estrazione dalla sorgente e il caricamento nella destinazione. Queste funzioni comprendono le seguenti operazioni: movimento dei dati, convalida dei dati secondo le regole di qualità, modifica del contenuto o della struttura dei dati, integrazione dei dati con dati provenienti da altre fonti, calcolo dei valori derivati o aggregati.
3.*Load*: Tutti i processi necessari per caricare i dati in una tabella di destinazione. Parti del processo di caricamento includono, per esempio, la gestione di chiavi derivate e la gestione della dimensione delle tabelle.

Quando si parla di data integration, ETL non è l'unico approccio disponibile, infatti sono disponibili sia ELT ed EII. Il primo che è l'acronimo di extract, load e transform, nella quale prima si estraggono i dati dalla sorgente, poi si salvano sul database e infine vengono integrati e trasformati nel formato desiderato. Il vantaggio di questo approccio, in generale, è che un database è più adatto per la gestione di grandi carichi di lavoro dove centinaia di milioni di record devono essere integrati ed è generalmente ottimizzato per I/O, che aiuta a elaborare i dati più velocemente. Però c'è un grande ma qui: per poter avere un efficiente approccio ELT, lo strumento ELT deve sapere come utilizzare la piattaforma del database di destinazione e il dialetto SQL specifico in uso, per questo motivo non esistono molte soluzioni ELT sul mercato rispetto a quelle ETL. Tuttavia, la maggior parte dei tradizionali ETL proprietari hanno ampliato gli strumenti forniti inserendo pushdown SQL come funzionalità, fondamentalmente con conseguente supporto all’ETLT (Extract, transform, load e transform). Scenari, dove la trasformazione può avvenire sia all'interno dell'engine, principalmente le operazioni non supportate dal database di destinazione, o dopo il caricamento all'interno del database.

Sia l'ETL che l'ELT spostano o copiano i dati fisicamente in un altro archivio dati, dalla sorgente al sistema di memorizzazione; ma in molti casi, tuttavia, non c'è

bisogno di muovere o copiare i dati. Infatti la maggior parte degli utenti non è interessato a sapere se esiste un processo ETL e un data warehouse integrati, vogliono solo avere un accesso rapido e facile ai dati. Quindi al posto di integrare fisicamente i dati, si può decidere di avere un sistema virtualmente integrato, rendendo i dati accessibili in tempo reale quando è necessario, questo si chiama enterprise information integration o EII. Il principale vantaggio di questo approccio è ovviamente il fatto che i dati sono sempre aggiornati, un altro vantaggio è che non esiste uno strato di memorizzazione aggiuntivo e non si ha duplicazione di dati supplementari. Usando tecniche di integrazione virtuale dei dati, i dati sono accessibili per l'utente finale come se fosse un data marts, ma in realtà è lo strumento EII che si occupa di tutte le traduzioni e trasformazioni in background. Anche se sembra una strategia vincente, ha alcuni inconvenienti, analizzando la tabella 1.1 è possibile trarre alcune conclusioni. Una è quella che gestire grandi volumi di dati filtrati usando un approccio virtuale sarà impegnativo, se non impossibile. Un’altra conclusione potrebbe essere che uno strumento ETL che appartiene tipicamente all'ambiente fisico come categoria di integrazione, offre alcuni dei vantaggi di una soluzione virtuale combinati con tutte le funzionalità di un tool ETL completo.

Caratteristcihe	Fisico	Virtuale
Diffusione dati	○	●
Performance/latenza Query	●	⊙
Frequenza di accesso	●	⊙
Diversita sorgenti dati	⊙	●
Diversità tipi di dati	⊙	●
Sorgenti dati non relazionali	○	●
Trasformazioni e pulizia	●	○
Prevedibilità performance	●	⊙
Interfacce multiple per gli stessi dati	○	●
Large query/volume dati	●	○
Bisogni di storico/aggregazione	●	○

Legend: ○ = Debole, ⊙ = Accettabile, ● = Forte

Figura 1.1: virtuale Vs fisico

1.3 Parti che costituiscono Pentaho-Kettle

Ogni strumento di data integration utilizza diversi nomi per identificare le parti che costituisco il tool, i concetti sottostanti e le cose che puoi costruire con esso, e Kettle non fa eccezione. Questa sezione introduce e spiega alcune terminologie specifiche di Kettle.

1.3.1 Trasformazioni

Una trasformazione è la principale componente di una soluzione ETL, gestisce la manipolazione di righe o dati nel senso più generale possibile dell'acronimo extract, transform e load. È costituito da uno o più passaggi che eseguono il

lavoro fondamentale ETL come la lettura di dati da file, il filtraggio delle righe, la pulizia dei dati o il caricamento dei dati in un database.

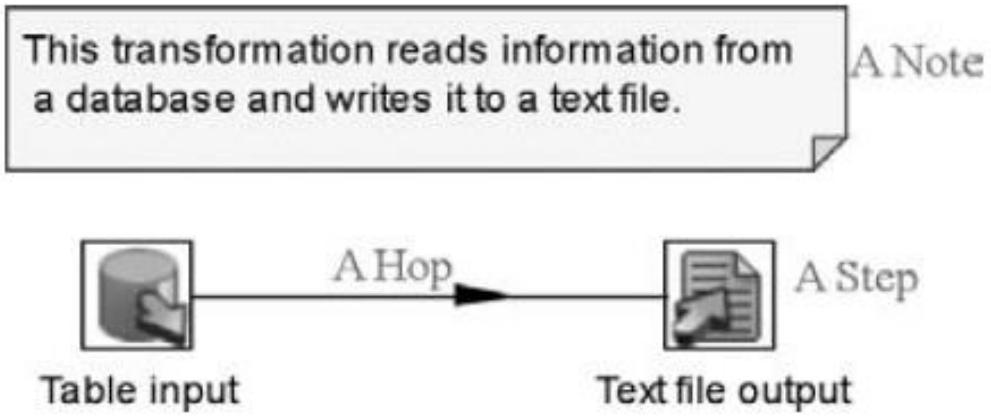

Figura 1.2 esempio di trasformazione

I passi di una trasformazione sono collegati tramite hop, che sono un canale di trasmissione a senso unico che consente la trasmissione dei dati.

In Kettle l'unità di misura è la riga e un flusso di dati è il movimento di righe da un passo all'altro. Oltre passi e hop, le trasformazioni possono contenere note, le quali sono piccoli riquadri, che posso essere collocati ovunque, in una trasformazione e possono contenere testo arbitrario e servono per documentare la trasformazione.

Un passo è l'elemento centrale di una trasformazione ed è graficamente rappresentato da un'icona. Ecco alcune delle caratteristiche chiave di un passo:

- Un passo deve avere un nome unico in una singola trasformazione.
- Praticamente ogni passo è in grado di leggere e scrivere righe di dati.
- Può scrivere dati su uno o più hop in uscita e leggere i dati che arrivano da uno o più hop in ingresso.
- La maggior parte dei passi può avere più hop in uscita, e può essere configurato a distribuire o copiare i dati. Quando si distribuiscono i dati, il passaggio si alterna tra tutti gli hop in uscita (round robin), mentre quando

si copiano ogni riga viene inviata a tutti gli hop.

- Quando si esegue una trasformazione, vengono avviate una o più copie di ogni passo, ognuna esegue nel proprio thread. Durante l'esecuzione, ogni copia del passo vengono eseguite contemporaneamente, con le righe di dati che costantemente passano sui loro hop di connessione.

Oltre a queste funzionalità standard ogni passo ovviamente ha una funzionalità distinta che viene rappresentata dal tipo di passo. Ad esempio, in figura 1.2 il passo “Table input” esegue una query SQL per leggere i dati da una tabella di un database relazionale e scrive i dati come le righe sui suoi hop in uscita, e il passo “Text file output” legge le righe dai suoi hop in entrata e le scrive su un file di testo.

Un hop, rappresentato da una freccia tra due passi, definisce il percorso dei dati tra i vari passi, rappresenta anche un buffer di righe tra due passi. Quando un set di righe è pieno, il passo che scrive si arresta se c'è ne è la possibilità. Quando un set di righe è vuoto, il passo che fa la lettura attenderà un po’ fino a quando le righe sono nuovamente disponibili.

Le semplici regole fornite dall’hop consentono di eseguire i passi in maniera parallela in thread separati. Questo perché le righe sono costrette a passare attraverso la rete di passi di trasformazione, producendo la massima quantità di parallelismo. Le regole permettono anche che i dati siano elaborati in modalità streaming con minimo consumo di memoria. Per quanto riguarda Kettle, non è possibile definire un ordine di esecuzione, e non è possibile o necessario identificare alcuna partenza o fine di una trasformazione. Questo perché tutti i passi vengono eseguiti in parallelo: quando viene avviata una trasformazione, tutti i suoi passi vengono avviati e continuano a leggere le righe dai loro hop in arrivo e scrivono le righe sui loro hop in uscita finché non sono esaurite, terminando l'esecuzione del passo. La trasformazione nel complesso si ferma dopo che l'ultimo passo è terminato, detto ciò funzionalmente una trasformazione ha quasi sempre un inizio e una fine definiti. Ad esempio, la trasformazione mostrata in Figura 1.2 “inizia" al passo "Table input" (perché quel passo genera le righe) e

"finisce" al passo "Text file output" (perché quel passo scrive le righe su un file e non le invia ad un altro passo successivo per ulteriori elaborazioni).

I dati che passano da un passo ad un altro su un hop arrivano in forma di una riga di dati. Una riga è una raccolta di zero o campi che possono contenere i seguenti tipi di dati:

- *Stringhe*: qualsiasi tipo di dati carattere senza alcun limite particolare.
- *Numero*: numero a virgola mobile a doppia precisione.
- *Intero*: un numero intero lungo (64 bit).
- *BigNumber*: un numero con precisione arbitraria.
- *Data*: un valore data-ora con precisione al millisecondo.
- *Booleano*: un valore che può contenere vero e falso.
- *Binario*: i campi binari possono contenere immagini, suoni, video e altri tipi di Dati binari.

Ogni passo è in grado di descrivere le righe che vengono emesse. Questa descrizione delle righe è anche chiamata “meta-dati di riga” che contiene le seguenti informazioni:

- *Nome*: Il nome del campo deve essere unico in una riga.
- *Tipo di dati*: Il tipo di dati del campo.
- *Lunghezza*: La lunghezza di una stringa o il numero di tipo BigNumber.
- *Precisione*: la precisione decimale di un numero di tipo BigNumber.
- *Maschera*: Il formato di rappresentazione (o maschera di conversione).
- *Decimale*: il simbolo decimale in un numero.
- *Gruppo*: il simbolo di raggruppamento.
- *Origine del passo*: Kettle tiene traccia dell'origine di un campo nei meta-dati della riga.

Infine ecco alcune regole sui tipi di dati da ricordare durante la progettazione delle trasformazioni: tutte le righe di un set devono avere sempre lo stesso layout o struttura. Oltre il tipo di dato e il nome, i meta dati di campo non vengono forzati durante l'esecuzione della trasformazione. Per impostazione predefinita, le stringhe vuote ("") sono considerate equivalenti a NULL.

1.3.2 Jobs

Nella maggior parte dei progetti ETL è necessario eseguire ogni tipo di manutenzione. Per esempio si desidera definire cosa deve essere fatto nel caso qualcosa vada male e come i file devono essere trasferiti; si desidera verificare se esistano tabelle in un database e così via. È anche importante che questi compiti siano eseguiti in un certo ordine, poiché le trasformazioni eseguono i passi in maniera parallela, è necessario un job per gestire questo. Un job è costituito da una o più job entries eseguite in un determinato ordine, che è determinato dai job hops tra le job entries e dai risultati dell'esecuzione stessa. Come una trasformazione, un lavoro può contenere note per scopi di documentazione. Una job entry è un elemento fondamentale del job, come un passo, è anche rappresentato graficamente sotto forma di icona. Tuttavia, se si guarda da vicino si vede che le job entry differiscono in diversi modi:

- Mentre il nome di una nuova job entry deve essere univoco, è possibile creare le cosiddette copie shadow di una job entry. Ciò consente di inserire la stessa job entry in un job su più posizioni. La copia shadow punta alle stesse informazioni, in modo che se modifichi una copia, le modifichi tutte.
- Una job entry passa un oggetto risultato tra le job entries. Sebbene questo oggetto risultato possa contenere righe di dati, esse non vengono trasmesse in modalità streaming. Piuttosto, dopo che la job entry finisce, le righe vengono trasferite tutte in una volta alla job entry successiva.
- Tutte le job entry vengono eseguite in una determinata sequenza per impostazione predefinita. Solo in casi particolari vengono eseguiti in parallelo.

Poiché un job esegue in sequenza le job entry, è necessario definire un punto di partenza, che è una speciale job entry chiamata “start”. Come si può dedurre è possibile inserire un solo start in un job.

I job hops vengono utilizzati in un job per definire un percorso di esecuzione tra le job entries. Questo è fatto sotto forma di collegamento tra due job entries e da un tipo di valutazione dei risultati. Questo tipo di valutazione dei dati può essere uno dei seguenti:

- *Incondizionato:* significa che la prossima job entry viene eseguita, non importa cosa è successo nella precedente.
- *Seguire quando il risultato è vero*: questo percorso viene seguito quando il risultato di esecuzione della precedente job entry è vero. In generale significa che essa è andata a buon fine.
- *Seguire quando il risultato è falso*: questo percorso viene seguito quando il risultato di esecuzione della precedente job entry è falso o non riuscito.

Il tipo di valutazione può essere impostato andando ad agire direttamente sull'hop desiderato, o attraverso click destro del mouse o con un click su le icone dell'hop.
Le Job entries vengono eseguite usando il cosiddetto algoritmo di backtracking. Il percorso stesso è determinato dal risultato effettivo delle Job entries, tuttavia, l'algoritmo di backtracking significa che un percorso di Job entries viene sempre seguito fino alla fine prima che la successiva possibilità venga presa in considerazione. Si penda di esempio l'immagine 1.3.

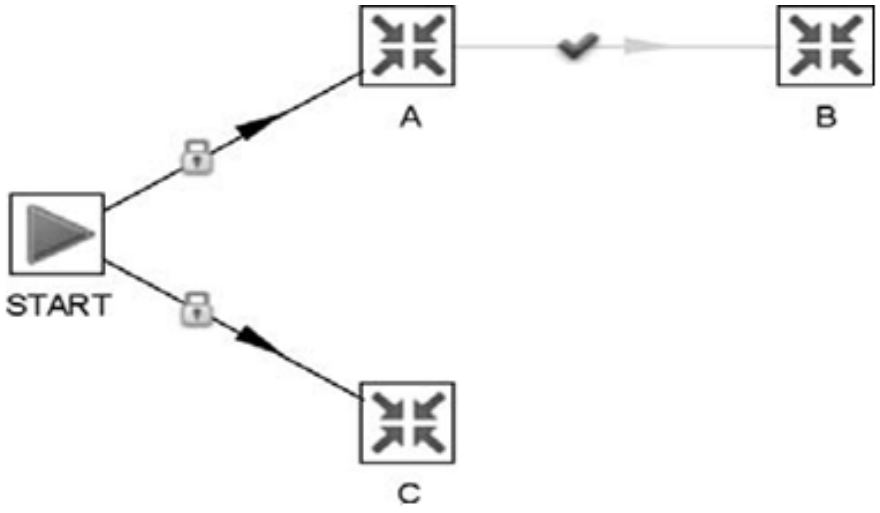

Figura 1.3 esempio di esecuzione di un job

Nella situazione mostrata in figura, le entries A, B e C vengono eseguite in sequenza:

- In primo luogo la job entry Start cerca tutte le job entries successive e trova A e C.
- A viene eseguito.
- A cerca tutte le job entries successive e trova B.
- B viene eseguito.
- B cerca tutte le job entries successive ma non ne trova.
- A non ha ulteriori job entries da eseguire.
- Start ha un’altra job entry da eseguire: C.
- C viene eseguito.
- C cerca tutte le job entries successive ma non ne trova.
- Start non ha più job entries da eseguire.
- Il job finisce.

Tutta via, poiché nessun ordine è definito a priori, si potrebbe avere anche CAB. La natura backtracking di un lavoro è importante per due motivi principali:

- il risultato del job stesso viene preso dall'ultima job entry che è stata eseguita. Poiché l'ordine di esecuzione delle job entries ABC o CAB non abbiamo nessuna garanzia che il risultato dell'ultima job entries C sia preso. Potrebbe anche solo essere A.
- Nei casi in cui si crea un loop, si pone un onere sullo stack dell'applicazione in quanto tutte le job entries e i loro risultati verranno mantenute in memoria per la valutazione.

A volte è necessario eseguire in parallelo le job entries. Questo è possibile, disponendo una job entry per eseguire le job entries successive in parallelo. Tuttavia, a volte si vuole avere alcune job entries eseguite in parallelo e poi continuare con altri lavori. In quella situazione, è importante mettere tutte le job entries eseguite in parallelo in un nuovo job, il quale può essere eseguito in un altro job.

Il risultato dell'esecuzione di una job entry non determina solo la prossima job entry che verrà eseguita. Inoltre passa l'oggetto risultato stesso alla prossima job entry. L'oggetto risultato contiene le seguenti informazioni:

- *Un elenco di righe risultato*: queste righe risultato possono essere copiate in una trasformazione usando il passo “Copy rows to result” e posso essere lette usando il passo “Get rows from result”. Alcune job entries come shell, trasformazioni o job hanno un'opzione per ciclare su le righe risultato, consentendo la parametrizzazione avanzata delle trasformazioni e dei job.
- *Un elenco di nomi di file*: questi nomi di file vengono assemblati durante l'esecuzione di una job entry, l'elenco contiene i nomi di tutti i file con cui entra in contatto una job entries. Ad esempio, se una trasformazione legge e ha elaborato dieci file XML, il nome dei file letti saranno presenti nell'oggetto risultato. È possibile accedere a questi nomi di file in una trasformazione utilizzando il passo “Get files from result”, il quale mostra anche il tipo di file.
- *Il numero di righe letto, scritto, input, output, aggiornato, cancellato, rifiutato e in errore di una trasformazione*
- *Lo stato di uscita di una job entry Shell*: consente di valutare in modo specifico il risultato di un valore di ritorno dello script shell.

Oltre all'utilizzo semplice, come looping e gestione file in determinate job entries, è anche possibile eseguire valutazioni più avanzate utilizzando la job entry JavaScript. Nella job entry JavaScript, in quanto tale, è possibile specificare condizioni alternative per vedere se un determinato percorso deve essere seguito o meno. Ad esempio, È possibile contare il numero di righe rifiutate in una trasformazione. Se tale numero è superiore a 50, è possibile determinare che la trasformazione non è riuscita, anche se quello non è il comportamento standard.

1.3.3 metadati di Trasformazione o di Job

Le trasformazioni o i job sono elementi fondamentali del tool ETL Kettle. Come trattato, posso essere rappresentati in XML, memorizzate in un repository o in forma di API Java. Ciò rende cruciale mettere in evidenza alcune proprietà dei meta dati principali:

- *Nome*: Il nome della trasformazione o del job. Anche se questo non è assoluto è consigliabile utilizzare un nome univoco non solo all'interno di un progetto, ma anche tra diversi progetti ETL e dipartimenti. Questo aiuterà durante l'esecuzione remota o la memorizzazione in un repository centrale.
- *Nome file*: Questo è il nome del file o dell'URL in cui è stata caricata la trasformazione o il job. Questa proprietà viene impostata solo quando l'oggetto viene memorizzato sotto forma di file XML. Quando caricato da un repository questa proprietà non è impostata.
- *Directory*: Questa è la directory (cartella) in un repository kettle dove la trasformazione del job viene caricata. Quando viene caricato da un file XML questa proprietà non è impostata.
- *Descrizione*: È possibile utilizzare questo campo opzionale per fornire una breve descrizione di cosa la trasformazione o il job fa. Se si utilizza un repository kettle, questa descrizione verrà mostrata nell'elenco dei file nella finestra di dialogo.
- *Descrizione estesa*: Un altro campo opzionale che è possibile utilizzare per dare una descrizione più estesa di una trasformazione o di un job

1.3.4 Connessione al database

Le connessioni al database Kettle vengono utilizzate da trasformazioni e job allo stesso modo per connettersi a basi di dati relazionali. Queste in realtà sono dei descrittori di connessione al database: sono una formula che può essere utilizzata per aprire una connessione effettiva con il database. Per definizione, la connessione effettiva con il RDBMS viene resa disponibile a runtime, l'atto di

definizione della connessione da solo non apre una realmente la connessione alla base di dati. Purtroppo, pochissimi database si comportano esattamente nello stesso modo, perciò abbiamo un numero elevato di opzioni per la connessione al database. È necessario specificare tre opzioni principali nella finestra di dialogo per la connessione:

- *Nome connessione*: Il nome identifica una connessione, in quanto tale deve essere unico in ogni trasformazione o job.
- *Tipo connessione*: Il tipo di server del database relazionale al quale si desidera accedere. La scelta determina le impostazioni, le opzioni di accesso e il dialetto SQL usato da Kettle per generare istruzioni SQL.
- *Accesso*: Da questo elenco, si seleziona il tipo di accesso che si desidera utilizzare. Tipicamente, si consiglia di utilizzare una connessione nativa (JDBC), tuttavia è possibile adoperare altri tipi di connessione.

A seconda delle opzioni selezionate, verrà presentato un ulteriore insieme di opzioni specifiche per il database selezionato. Di solito includono le seguenti opzioni:

- *Nome host*: Questo è il nome dell'host o l'indirizzo IP del server del database.
- *Nome database*: Per i server che supportano più database, è possibile specificare il nome del database a cui accedere.
- *Numero porta*: qui verrà inserita la porta predefinita per il tipo di connessione selezionato, assicurarsi che sia coretto.
- *Nome utente e password*: Il nome dell'utente e la password che si desidera utilizzare per accedere al server.

Per la maggior parte degli utenti, le impostazioni predefinite della connessione al database sono sufficienti. Tuttavia, potrebbe essere necessario utilizzare una delle opzioni avanzate che sono: supporta al tipo di dato booleano, virgoletta tutto nel database, forzare tutto al minuscolo, forza tutto al maiuscolo, nome dello schema preferito, istruzioni SQL da eseguire dopo la connessione.

Oltre a queste opzioni avanzate è possibile specificare alcune funzionalità specifiche del database. Qui è possibile specificare un elenco di parametri driver

specifici. Per alcuni tipi di database Kettle imposta alcuni valori predefiniti per aiutare. Infine, è possibile abilitare la connessione Apache Commons Database Connection che condivide il back end fornito con Kettle. Fare ciò avrà senso solo nelle situazioni in cui si stanno eseguendo molte trasformazioni o lavori molto piccoli con connessioni al database di breve durata. Non limiterà in alcun modo il numero di connessioni di database simultanee.

Una connessione al database Kettle viene utilizzata solo durante l'esecuzione di un job o di una trasformazione. In un job, ogni job entry apre e chiude le connessioni effettive al database indipendentemente, lo stesso accede in una trasformazione. Tuttavia, a causa della natura parallela di una trasformazione, ogni passo che utilizza una connessione ne apre una separata e avvierà una transazione separata. Mentre questo generalmente fornisce prestazioni ottimali per le situazioni più comuni, può causare gravi blocchi e problemi di integrità quando diversi passaggi aggiornano le informazioni nella stessa tabella. Per risolvere i problemi che possono derivare dall'apertura di più connessioni di database contemporanee, Kettle consente di abilitare l'opzione "Make transformation database transactional". Quando questa opzione è abilitata, la trasformazione apre una singola connessione al database effettiva per ciascuna connessione al database Kettle. Inoltre, Kettle eseguirà una COMMIT quando la trasformazione termina con successo e un ROLLBACK quando fallisce.

1.3.5 Repositories

Quando si trovano ad affrontare grandi progetti ETL con molti sviluppatori che lavorano insieme, è importante disporre di strutture che consentano la cooperazione. Kettle fornisce un modo per definire i tipi di repository in modo pluggable e flessibile, aprendo la strada a diversi tipi di repository da utilizzare. Tuttavia, le basi restano uguali indipendentemente dal tipo di repository utilizzato: utilizzeranno tutti gli stessi elementi di interfaccia grafica utente e meta dati memorizzati saranno uguali. Esistono diversi tipi di repository:

- *Database repository*: Il Database repository è un database relazionale. Un repository di questo tipo è facile da creare: basta creare un nuovo database vuoto e una connessione allo schema del database. Se non si dispone ancora di questo schema, è possibile utilizzare la finestra di dialogo Database Repository per creare il nuovo set di tabelle e indici del repository.
- *Pentaho repository*: Il Pentaho repository è un plug-in istallato come parte della Enterprise Edition di Pentaho Data Integration. È supportato da un sistema di gestione dei contenuti (CMS) che assicurerà che siano soddisfatte tutte le caratteristiche importanti del repository ideale, tra cui la gestione della versione e controlli di integrità.
- *File repository*: Questo tipo di repository semplificato consente di definire un repository in qualsiasi tipo di cartella. Poiché Kettle utilizza un back-end di un file system virtuale, ciò include posizioni non banali, quali file zip, server Web e server FTP

Indipendentemente dal tipo, il repository ideale dovrebbe possedere queste caratteristiche:

- *Memoria centrale*: Memorizzare le trasformazioni e i job in una posizione centrale. Questo permetterà agli utenti ETL di accedere all'ultima visione del progetto completo.
- *Blocco dei file*: Questa funzione è utile per impedire agli altri utenti di modificare una trasformazione o il job in cui si sta lavorando.
- *Gestione delle revisioni*: Un repository ideale memorizza tutte le versioni precedenti di trasformazioni e jobs per un futuro riferimento. Consente di aprire le versioni precedenti e di visualizzare il registro dettagliato delle modifiche.
- *Controllo integrità referenziale*: Questa funzione verificherà l'integrità referenziale di un repository. Questa funzionalità è necessaria per assicurarsi che non ci siano collegamenti, trasformazioni, jobs o connessioni al database mancanti in un repository.

- *Sicurezza*: Un repository protetto impedisce agli utenti non autorizzati di modificare o eseguire la logica ETL.
- *Riferimento*: È molto utile per uno sviluppatore di data integration di essere in grado di riorganizzare le trasformazioni e i jobs o di rivederli semplicemente. Dovrebbe essere possibile farlo in modo tale che i riferimenti a queste trasformazioni e ai jobs siano mantenuti intatti.

La gestione flessibile e uniforme dei file è molto importante per qualsiasi strumento ETL. Ecco perché Kettle supporta la specifica dei file nel senso più ampio come URL. Il back end di Apache Commons VFS che è stato messo in atto si prenderà cura della complessità. Ad esempio, con Apache VFS, è possibile elaborare una selezione di file all'interno di un archivio .zip nello stesso modo in cui si elaborerebbe un elenco di file in una cartella locale.

1.4 Parametri e variabili

È molto importante quando si utilizza uno strumento di data integration che alcuni parametri del tuo lavoro possono essere parametrizzati. Questo rende più facile far sì che il lavoro sia mantenibile. Ad esempio, è importante che la posizione della cartella con i file di input sia specificata in una posizione centrale in modo che i diversi componenti dello strumento di data integration possano utilizzare tal valore. In Kettle, viene fatto con le variabili. Ogni variabile ha un nome univoco e contiene una stringa di qualsiasi lunghezza, sono impostate a livello di sistema o dinamicamente in un job. Hanno uno spazio specifico che consente di eseguire lo stesso job o la trasformazione in parallelo con diverse variabili impostate sullo stesso sistema, Java Virtual Machine o contenitore J2EE senza alcun problema. Esistono due modi principali per la creazione di un'istanza di una variabile: può essere impostata dal sistema o definita dall'utente. Le variabili di sistema includono quelle definite dalla Java Virtual Machine e quelle definite da Kettle. Le variabili possono essere impostate in più modi. Il modo più comune di definire una variabile è collocarlo nel file kettle.properties nella directory home Kettle (${KETTLE_HOME}/.kettle). È anche possibile impostare le variabili

dinamicamente durante l'esecuzione di un lavoro. Ci sono diversi modi per farlo, ma il modo più semplice è con una trasformazione. Il passo Set Variables consente di impostare una variabile con l'ambito appropriato per l'utilizzo nel job corrente. Per situazioni più dinamiche puoi anche utilizzare un passo JavaScript o un passo di una classe Java definito dall'utente. Infine, le variabili possono essere formalmente specificate quando sono dichiarate come parametri nominali.

Se ci sono variabili che si considerano parametri per la trasformazione o il job, è possibile dichiararli nella scheda Parametri, in cui è possibile immettere il nome del parametro, il valore predefinito e una descrizione. Il principale vantaggio di utilizzare i parametri nominali è che sono elencati esplicitamente, documentati con una descrizione e portano un valore predefinito facoltativo. Questo rende evidente che è possibile impostare i loro valori in un job o una trasformazione. Una volta definiti i parametri, è possibile specificare i valori in tutti i percorsi possibili in cui si esegue una trasformazione o un job: nelle finestre di dialogo di esecuzione, nella riga di comando di Pan o di Kitchen, o nelle trasformazioni o nelle job entries.

Capitolo 2

Progetto AssistenteVendite

In questo capitolo andrò ad elencare le scelte fatte per l'estrazione, l'elaborazione e il salvataggio dei dati.
Nella prima parte si descrive l'approccio scelto per l'utilizzo di Pentaho-Kettle, andando a descrivere i formati di file utilizzati e le problematiche affrontate per il recupero di informazioni da essi. Successivamente vengono descritte le librerie esterne usate per rendere i dati conformi agli standard di Pentaho-Kettle e quelle per permettere l'uso da parte di Java di job e trasformazioni. Infine andrò a descrivere la strategia utilizzata per il salvataggio dei dati sul database relazionale.

2.1 Estrazione dati

Visto che i dati sono stati interamente reperiti da siti web, il principale formato di file con cui si è lavorato è stato quello XML. Tuttavia Kettle permette l'estrazione di dati, attraverso il passo "get data from XML", solo ed esclusivamente da file XML well-formed. Siccome molti siti non sono scritti rispettando queste regole è necessario rielaborarli in modo tale da essere conformi con esse, per fare ciò ci sono diverse strategie. Una di queste, quella adotta, consiste nell'avvalersi di uno strumento che in modo automatico manipola il file XML e lo rende well-formed. Quando ciò non è stato possibile si è scelto di estrarre dal file le tag che contengono le informazioni necessarie e le si è scritte su un file TXT in formato CSV, dal quale sono poi stati estratti con il passo "input CSV".

2.1.1 Formato XML e XPath

Un file in formato XML è un file scritto con il linguaggio eXtensible Markup Language (XML), che è un linguaggio di marcatura estensibile. Questo linguaggio permette di inserire all'interno del testo delle etichette o marcature (tag), che

permettono di descrivere il contenuto del documento stesso. Al contrario di HTML, l'insieme delle marcature utilizzabili non è predefinito, perciò ogni autore del documento XML definisce i propri tag. È stato progettato per trattare la descrizione delle informazioni, cioè concentra l'attenzione sul significato dei dati, le marcature indicano in maniera non ambigua il significato di ciascun dato. Una caratteristica fondamentale dell'XML è l'estensibilità. Siccome ogni autore può creare nuovi tag per descrivere i contenuti semantici dei propri dati, si ha la necessità di definire delle regole grammaticali, o vincoli, alle quali gli elementi devono attenersi. Queste regole sono definite nelle specifiche XML e sono codificate nel DTD. Quando un documento XML è ben formato e rispetta le regole del DTD a cui si riferisce si dice che è un documento XML valido.

In un documento XML, come già accennato, possiamo inventare i tag, la cui scelta può essere effettuata a seconda delle informazioni che interessa rappresentare e che la specifica applicazione dovrà riconoscere. I tag si dividono in:

- *Tag di apertura*: <nome tag>
- *Tag di chiusura*: </nome tag>
- *Tag vuoti*: <nome tag/>

Gli elementi in XML sono in relazione tra loro, che vanno a determinare il modello del documento. Esso è organizzato come un albero, in cui la relazione di contenimento tra tag è equivalente alla relazione nodo-sotto nodo, l'entità che non è sotto entità di nessuno è l'entità radice. Ogni elemento deve avere il tag di chiusura e in XML un elemento può essere contenuto in un altro, però l'ordine di chiusura dei tag deve essere rispettato in ordine inverso. Si possono usare attributi per aggiungere una caratteristica o una proprietà ad un elemento. L'attributo è una coppia nome-valore, separate da un segno di uguale, che può essere presente all'interno del tag di apertura, ci possono essere più attributi per lo stesso elemento. Esso è sempre associato ad un elemento e può ricorrere in più elementi.

Problemi con gli attributi:

- non possono contenere valori multipli.
- sono difficilmente espandibili

- non possono descrivere strutture
- non hanno un supporto standard per la gestione nei programmi
- sono difficili da controllare

Le regole sintattiche di XML sono le seguenti: in XML tutti i tag devono essere chiusi, i tag devono essere correttamente innestati, XML è case sensitive (maiuscole e minuscole devono essere rispettate), gli spazi sono preservati, i commenti possono essere inseriti tra i segni **<!--** e **-->.**

XPath è un linguaggio[2], parte della famiglia XML, che permettere di individuare i nodi all'interno di un documento XML. Le espressioni XPath non servono ad identificare la struttura di un documento, ma a localizzare con precisione i nodi. Il tipo più comune di espressione XPath è la path expression che è scritta come una sequenza di passi per raggiungere un nodo XM, può essere un percorso relativo o un percorso assoluto, i cui elementi sono separati da '/'.
Sono state definite due notazioni:

-Sintassi abbreviata, compatta, che permette la realizzazione di costrutti intuitivi e facilmente realizzabili;

-Sintassi completa, più complessa, che contiene maggiori opzioni, in grado di specificare con più particolarità gli elementi.

Per estrarre i dati da un documento XML si usa il passo “get data from XML”, che dà la possibilità di leggere dati da ogni tipo di file XML usando un XPath specifico.

2.1.2 Formato TXT e CSV

L'estensione .txt denota semplici file di testo e il tipo di file TXT, di conseguenza. Un file TXT è un file che contiene solo caratteri stampabili più il carattere particolare end-of-line (EOL), in più può avere, facoltativamente, caratteri di controllo end-of-file (EOF). Non contengono metadata, ma possono avere solo una formattazione più semplice e sono normalmente codificati in ASCII, ANSI o

Unicode. Essi sono direttamente leggibili e convenzionalmente indicati come file di “testo normale”.
Il formato CVS invece è un formato di file basato su quello di testo, usato per l'importazione ed esportazione di una tabella di dati. Non si ha uno standard che lo definisce, ma solo alcune prassi. In questo formato ogni riga della tabella è normalmente rappresentata da una linea di testo, che a sua volta è divisa in campi separati da un apposito carattere separatore, ciascuno dei quali rappresenta un valore. Questo formato non specifica una codifica di caratteri, la convenzione per indicare il fine linea, il carattere da usare come separatore tra i campi, convenzioni per rappresentare date e numeri, se la prima riga è solo di intestazione o meno. Questi dettagli possono dover essere specificati dall'utente tutte le volte che si importano o esportano dati in formato CSV. Nel migliore dei casi non sono ammessi valori che contengano caratteri di fine linea o il carattere che separa i campi: in questo caso, ad ogni riga della tabella corrisponde esattamente una linea di testo.

Visto che dobbiamo lavorare con delle tabelle i file TXT sono stati scritti in formato CSV, dal quale estraiamo le informazioni con il passo “CSV input”. Questo passo permette di leggere dati da un file delimitato, specificando il tipo di separatore utilizzato.

2.2 Librerie adoperate

Siccome non tutti i siti web utilizzati non ritornavano un file XML well-formed, si è dovuto ricorre a le librerie esterne JTidy e Jsoup, usate per poter modificare la struttura del documento in modo da renderla conforme allo standard, oppure per estrarre le tag, al cui interno, si trovano le informazioni ricercate. Un altro tipo di librerie sono quelle per usare Kettle come un API Java

2.2.1 Librerie per l'elaborazione di file XML

JTidy: JTidy[3] è una libreria esterna di Java che implementa delle classi che realizzano un HTML Tidy. Questo strumento si occupa di correggere tutti gli errori che tipicamente si fanno durante la scrittura di codice web. Si occupa anche di rendere leggibile il codice generato da un editor HTML, e di segnalare all'utente dove deve prestare più attenzione. Riesce a riconoscere una grande gamma di errori, segnalandoli come warning, indicando riga e colonna in cui li ha individuati. Nel caso in cui trovi problemi per cui non riesce a generare una soluzione, il motore Tidy non genererà codice, ma li metterà in evidenza etichettandoli come errors. Tidy come feature principale ha quella di correggere il linguaggio di markup rispettando la sintassi osservata in alcuni browser quali Firefox, Internet Explorer, Opera, Netscape. Questo avviene attraverso le seguenti operazioni: correzione dei tag di chiusura mancanti o non accoppiati, correzione dell'ordine di chiusura dei tag, correzione dei problemi nei tag heading, correzione di tag posizionati male, coretto posizionamento dei tag hr, aggiunta di '/' finale nelle ancore, correzione di liste. Dal punto di vista della grammatica invece implementa:

- *Aggiunta di "*: inserisce gli apici dove mancano, anche quelli finali dove dimenticati.
- *Attributi sconosciuti/proprietari segnalati*: conosce tutti gli attributi HTML e XHTML, e aiuta lo sviluppatore a trovare subito dove ha sbagliato grazie a segnalazioni.
- *Elementi proprietari segnalati*: riconosce quale versione HTML si sta usando, segnalando e aggiungendo il coretto DOCTYPE al documento.
- *Segnalazione di '>' mancanti*: Tidy corregge anche la mancanza di > nei tag

Inoltre offre la possibilità di decidere quale stile si vuole usare nella generazione di markup, inizialmente costruisce un albero di parsing pulito, direttamente dal file sorgente. Il quale viene poi visualizzato a seconda delle impostazioni nel file di configurazione. Per finire implementa anche le seguenti features:

- *Internazionalizzazione*: Offre una vasta scelta di caratteri tra cui scegliere, e si preoccupa di correggere eventuali errori nell'utilizzo degli stessi all'interno del documento.
- *Accessibilità*: Offre soluzioni a problemi riguardanti le persone che non usano browser text_based. Ad esempio, suggerisce sempre di inserire un sommario per gli attributi delle tabelle, in modo da favorire la navigazione a chi usa browser "aural".
- *Pulizia*: L'opzione -clean consente di rimuovere gli eccessi di font, style e color e di inserirli piuttosto in un foglio di stile CSS. Inoltre, cancella paragrafi ed heading vuoti, inserendo lo stile corretto nei fogli di stile, o usando il tag.
- *XML*: aiuta a correggere gli errori di well-formedness, che causano il rifiuto di file XML. Purtroppo non riconosce ancora tutte le funzionalità XML; per esempio, non comprende sezioni CDATA o sottogruppi DTD.

Nel progetto che si è andato a realizzare, JTidy è stato usato principalmente per questa feature, di seguito un codice di esempio che l'implementa:

```
7  public class Jtidy {
8
9      URL url;
10     Tidy tidy = new Tidy();
11     tidy.setMakeClean(true);
12     tidy.setXmlOut( true);
13
14     url = new URL("templateURL");
15
16     //lo scrivo su file
17     FileOutputStream fileOutputStream = new FileOutputStream("outputFileName");
18
19     Document doc = tidy.parseDOM( url.openStream(), fileOutputStream);
20
21 }
```

figiura2.1: esempio JTidy usato su XML

Come mostrato in figura 2.1 questo si fa creando un'istanza della classe Tidy, poi si imposta makeClean che serve per dire a Tidy di pulire il file XML. Dopo si imposta che il file generato da Tidy, sia un XML impostando XmlOut sempre a true. Infine si chiama il metodo parseDOM della classe Tidy al quale passiamo un

file in input dal quale leggere il codice XML da correggere ed un file in output su il quale scrivere il codice coretto.

Jsoup: È un parser HTML5 [4] le cui API consentono di estrarre dati e manipolare in maniera estremamente semplice documenti sfruttando le potenzialità di DOM, CSS e metodi di accesso simili a quelli offerti da Jquery. In Jsoup un documento è rappresentato dalla classe document, i nodi del documento da istanze della classe node e i tag da istanze della classe element. Esistono tre modi per effettuare il parsing di un documento XML ed ottenere così il document che lo rappresenta:

- *Caricare il documento da un URL*: Si utilizza il metodo connect della classe Jsoup.
- *Caricare il documento da un file:* Si utilizza il metodo parse della classe Jsoup passando come argomenti il file di input, il charset e il base-uri.
- *Effettuare il parsing da una stringa rappresentante il contenuto del documento XML o una parte di esso*: Si utilizza il metodo parse passando una String rappresentante il contenuto del documento e il base-uri.

Una volta ottenuto il document è possibile procedere alla navigazione del documento sfruttando diversi metodi. Ad esempio si può accedere al titolo, al body, recuperare l'elemento figlio i-esimo, recuperare l'elemento successivo o precedente a ad uno selezionato, recuperare tutti gli elementi di un dato tag, recuperare un elemento a partire dall'id o recuperare il padre di un dato elemento. Poi Jsoup permette anche di recuperare elementi che abbiano un dato attributo o che abbiano un dato attributo valorizzato in modo ben preciso. Queste sono le principali operazioni per accedere agli elementi, se si vuole invece operare sugli attributi abbiamo le seguenti operazioni: recuperare tutti gli attributi di un elemento, recuperare l'attributo con un determinato nome. In più a questi metodi tradizionali introduce uno strumento molto potente per recuperare uno o più elementi attraverso l'uso di selettori. Un selettore è un’espressione che individua un sotto insieme degli elementi presenti nel documento. Accanto a questi metodi per selezione e navigare un documento XML, mette a disposizione un insieme di metodi per manipolarne il contenuto.

Jsoup è stato utilizzato per connettersi e caricare i documenti dai siti web e per estrarre gli elementi di un determinato tag.

```
File input = new File("/tmp/input.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://example.com/");

Elements links = doc.select("a[href]"); // a with href
Elements pngs = doc.select("img[src$=.png]");
  // img with src ending .png

Element masthead = doc.select("div.masthead").first();
  // div with class=masthead

Elements resultLinks = doc.select("h3.r > a"); // direct a after h3
```

Figura 2.2: esempio utilizzo jsoup per estrarre tag

Il primo come già detto sopra usando il metodo connect al quale passiamo l'URL del sito e facciamo la get, il secondo attraverso il metodo select della classe document al quale passiamo la tag richiesta, in maniera simile a quella mostrata in figura 2.2.

2.2.2 Librerie Kettle

Siccome c'era l'esigenza di usare Kettle in un programma Java, abbiamo utilizzato le librerie di Kettle per permettere di utilizzarlo come API Java [5]. Le API principali sono 4:

- *Core*: Contiene le classi core per Kettle, memorizzate nel file kettle-core.jar.
- *Database*: Contiene le classi riguardanti i database supportati. Le classi sono memorizzate nel file kettle-db.jar.
- *Engine*: Package che contiene le classi di runtime Kettle, memorizzate nel file kettle-engine.jar.

- *GUI*: Tutte le classi relative a interfacce utente grafiche come Spoon e che dipendono da classi Eclipse SWT sono memorizzate nel file kettle-ui-swt.jar.

Di solito non sono necessarie tutte e quattro ma è meglio includerle lo stesso perché non si sa a che trasformazioni o job si andrà incontro.

```
1. package it.dontesta.lab.jobs.kettle;
2. importorg.pentaho.di.core.KettleEnvironment;
3. importorg.pentaho.di.core.logging.LogChannel;
4. import org.pentaho.di.core.logging.LogChannelInterface;
5. import org.pentaho.di.core.plugins.PluginRegistry;
6. import org.pentaho.di.core.plugins.RepositoryPluginType;
7. import org.pentaho.di.job.Job;
8. import org.pentaho.di.job.JobMeta;
9. import org.pentaho.di.repository.RepositoriesMeta;
10.import org.pentaho.di.repository.Repository;
11.import org.pentaho.di.repository.RepositoryDirectoryInterface;
12.import org.pentaho.di.repository.RepositoryMeta;13.public class ExecuteGenericJob {
14.    public static void main(String[] args) throws Exception {
15.            String jobName = args[0];
16.            KettleEnvironment.<em>init</em>();
…
17.            JobMeta jobMeta = new JobMeta();
18.            jobMeta = repository.loadJob(jobName, directory, null, null);
19.            Job job = new Job(repository, jobMeta);
20.            job.start();
21.            job.waitUntilFinished();
22.            if (job.getErrors() != 0) {
23.                    log.logError("Job Error: " + job.getErrors());
24.                    log.logError("Error encountered!");
25.            }
26.    }
27.}
```

Figura 2.3: Esempi di API Kettle per la chiamata di un job

Queste librerie sono state usate per accedere al repository di Kettle e per far partire i job. Non abbiamo avuto bisogno di far lanciare le trasformazioni da Java perché tutte quelle create sono state poi utilizzate come job entry. Come si vede nell'immagine 2.3, per prima cosa bisogna inizializzare l'ambiente Kettle con il metodo init della classe kettleEnviroment, poi si carica il job dal repository in un’istanza della classe jobMeta. Infine si crea un’istanza della classe job passandogli il jobMeta e il repository, e si far partire il job usando il metodo start della classe job. Il metodo waitUntilFinished serve per fermare l'esecuzione del programma fino a che il job non ha finito di eseguire.

2.3 Salvataggio dei dati sul database

Un altro problematica affrontata è che non sempre i dati potrebbero essere salvati in modo coretto. Per risolverla si è deciso di andare ad estrarre, non solo i dati recenti, ma anche quelli precedenti in ordine di tempo. Questa finestra temporale è stata impostata ad una settimana. Tuttavia in una tabella in un database relazionale non è possibile avere la stessa chiave duplicata, perciò bisogna salvare su di essa solo i dati nuovi o quelli mancanti, a causa di problemi. Inizialmente ho ideato una soluzione attraverso pentaho ma risultava molto elaborata, perciò si è deciso di ripiegare su una procedura di allineamento. Si è creata una tabella di appoggio nella quale andavamo prima a troncarla, poi a inserire i dati estratti e elaborati. Dopo questo si andava a chiamare una procedura SQL che inseriva dentro la tabella principale i dati non presenti in essa ma solo su quella di appoggio.

Parte II

L'applicazione

-

Assistente vendite basato su Pentaho Kettle

Capitolo 3

Progettazione

La progettazione del software prevede raccolta dei requisiti funzionali da sviluppare e una descrizione completa e non ambigua delle funzionalità principali del software che è stato realizzato e le rispettive caratteristiche. Essa illustra le scelte fatte riguardo ai compiti che il software svolge, senza entrare in dettagli implementativi.

3.1 Requisiti funzionali

Di seguito verranno descritti i requisiti funzionali, quindi il comportamento del sistema in risposta a determinati input e le principali operazioni che vengono svolte all'interno del software di assistenza.

3.1.1 Generazione file dati

La generazione dei file dei dati è la prima cosa che il sistema svolge all'avvio. Ciò è fatto connettendosi all'URL desiderato e accedendo al file XML associato a quel determinato URL. In questa fase si sceglie quanto andare indietro nel tempo a recuperare i dati. A partire dal file XML associato al sito web, esso viene rielaborato, se necessario e poi si genera il file contenente i dati grezzi nel formato desiderato. Questo si fa andando a salvare il contenuto del file XML rielaborato, in modo appropriato, all'interno di un file il cui nome è passato come input. Da questo file ora è possibile estrarre i dati che stiamo andando a ricercare.

RF01	Estrazione	Generazione file
Input	URL, nome file generato	
Processo	Si connette all'URL e accede al file XML associato, lo rielabora e salva il risultato in un file con nome passato in input.	
Output	File da cui e possibile estrarre i dati	

Tabella 1.1: Requisito funzionale 1

3.1.2 Trasformazione dati

modo che siano conformi allo schema della tabella in cui vengono salvati, all'interno del database. La trasformazione dei dati avviene selezionando i dati rilevanti, eliminando quelli superflui tra cui anche quelli duplicati e rielaborando i campi estratti, cancellando quelli non necessari. Durante questo procedimento si aggiungono tutti. Dopo aver generato il file è necessario estrarre da esso i dati grezzi, e trasformarli in dati ausiliari non presenti direttamente nel file, come ad esempio chiavi hash e date qualora non siano già presenti.

RF02	Trasformazione	Trasformazione dati
Input	File contente i dati	
Processo	Estrae i dati e li trasforma	
Output	Dati con lo stesso schema della tabella in cui devono essere salvati	

Tabella 1.2: Requisito funzionale 2

3.1.3 Salvataggio dati nella tabella di appoggio

Dopo che i dati sono stati trasformati bisogna salvarli all'interno della tabella di appoggio nel database. La tabella di appoggio va prima troncata, cioè il suo contenuto viene cancellato, questo non causa perdita di informazioni perché comunque già presenti in quella principale. Tutto questo viene fatto perché nella trasformazione non si tiene conto dei dati già presenti all'interno del database. Non è possibile salvare su di esso dati già presenti, in quanto si avrebbe duplicazione della chiave.

RF03	Salvataggio	Tabella di appoggio
Input	Dati trasformati	
Processo	Inserisce i dati all'interno della tabella di appoggio, precedentemente troncata	
Output	La tabella di appoggio contiene i dati trasformati	

Tabella 1.3: Requisito funzionale 3

3.1.4 Allineamento

L'ultima cosa che rimane da fare è allineare la tabella principale con quella di appoggio. Questo si fa andando a salvare i dati che sono presenti solo all'interno della tabella di appoggio in quella principale. Questo è molto facile da fare in quanto le tabelle hanno la stessa struttura, e perciò basta solo inserire i dati la cui chiave non è nella tabella principale.

RF04	Salvataggio	Allineamento
Input	Tabella di appoggio	
Processo	Inserisce i dati la cui chiave non è presente	
Output	Tabella principale aggiornata con i nuovi dati	

Tabella 1.4: Requisito funzionale 4

3.2 Requisiti non funzionali

I requisiti non funzionali analizzati per il progetto riguardano le proprietà del sistema e le caratteristiche dei servizi che offre. Essi definiscono i vincoli di sviluppo del sistema.

Il sistema deve essere automatizzato a cadenza giornaliera, cioè deve eseguire ogni giorno senza che ci sia l'intervento di un utente, una volta che è stato avviato. Non deve avere perdite di informazione a causa di malfunzionamenti, e se ciò dovesse succedere deve poter recuperare le informazioni perse. Deve essere aggiornato sempre alle informazioni più recenti. Deve tenere traccia della data in cui le informazioni sono state reperite, in modo da permettere di tenere traccia di uno storico di esse.

Il sistema si basa sulla memorizzazione di dati attraverso un DBMS, il quale è modulare e diviso in base alla categoria di informazioni reperite, in modo da rendere più facili la modifica e il controllo delle singole parti.

3.3 Progettazione database

Il progetto prevede il salvataggio dei dati all'interno di un database, la quale struttura è banale, siccome in realtà si utilizza il database più per poter sfruttare le potenzialità offerte dal linguaggio SQL. Il database che sono andato a progettare è composto da 2 schemi, lo schema "PRODOTTI" e lo schema "ARTICOLI".

Schema "PRODOTTI":

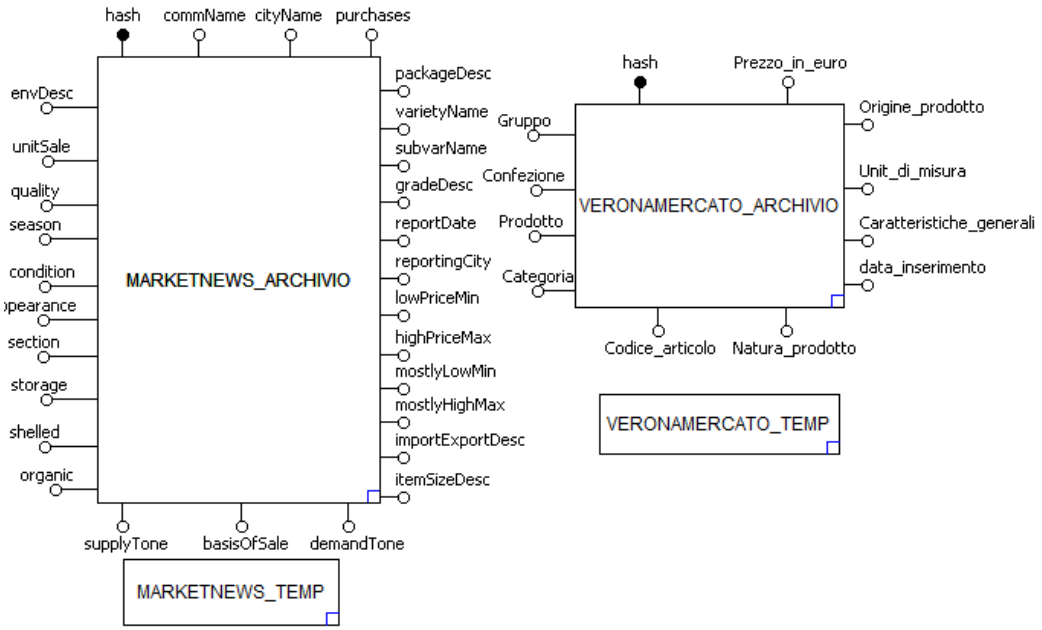

figura3.1: diagramma ER schema PRODOTTI

Il primo schema creato è quello che mantiene i dati riguardanti l'andamento del mercato ortofrutticolo, memorizzando il tipo di prodotto, le caratteristiche e il prezzo di vendita. Si è deciso di ricorre a sole entità e non a relazioni in quanto, per le esigenze richieste, le singole tabelle era più che sufficienti. È composto da quattro tabelle, due di appoggio e due principali:

-***MARKETNEWS_TEMP*** (commName, cityName, organic, packageDesc, varietyName, subvarName, gradeDesc, reportDate, lowPriceMin, highPriceMax, mostlyLowMin, mostlyHighMax, season, itemSizeDesc, color, envDesc, unitSale, quality, condition, appearance, storage, shelled, purchases, section, importExportDesc, supplyTone, demandTone, basisOfSale, marketTone, priceComment, commenti, reportingCity, hash)

-***MARKETNEWS_ARCHIVIO*** (commName, cityName, organic, packageDesc, varietyName, subvarName, gradeDesc, reportDate, lowPriceMin, highPriceMax, mostlyLowMin, mostlyHighMax, season, itemSizeDesc, color, envDesc, unitSale, quality, condition, appearance, storage, shelled, purchases, section, importExportDesc, supplyTone, demandTone, basisOfSale, marketTone, priceComment, commenti, reportingCity, hash)

-***VERONAMERCATO_TEMP*** (Codice_articolo, Gruppo, Prodotto, Natura_prodotto, Origine_prodotto, Confezione, Caratteristiche_generali, Categoria, Unit_di_misura, Prezzo_in_euro, data_inserimento, hash)

-***VERONAMERCATO_ARCHIVIO*** (Codice_articolo, Gruppo, Prodotto, Natura_prodotto, Origine_prodotto, Confezione, Caratteristiche_generali, Categoria, Unit_di_misura, Prezzo_in_euro, data_inserimento, hash)

Le prime 2, che hanno gli stessi campi, contengono i dati provenienti dal sito https://www.marketnews.usda.gov/. Sono composte dai campi che rimanda il file XML del sito in questione. Le due seguenti invece, che condividono anche loro i campi, contengono i dati provenienti dal sito http://www.veronamercato.it/. Anche queste hanno gli stessi campi che restituisce il sito. Si è deciso di creare due strutture diverse essendo la struttura dei dati diversa tra i due siti. In più abbiamo due procedure per l'allineamento.

Schema "ARTICOLI":

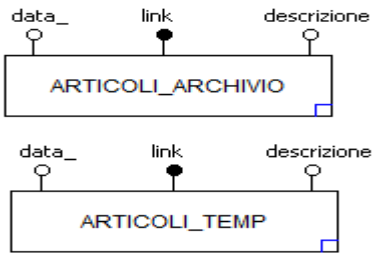

figura 3.2: diagramma ER schema ARTICOLI

Il secondo schema memorizza i link e la descrizione di vari articoli di giornale, i quali contengono informazioni che riguardano il mercato ortofrutticolo. Anche questo è composto solo da tabelle in quanto non era richiesto altro. È composto da due tabelle, una di appoggio e una principale:

- *ARTICOLI_ARCHIVIO* (link, descrizione, data_)
- *ARTICOLI_TEMP* (link, descrizione, data_)

Le quali contengono i dati che vengono estratti da i seguenti tre siti:

- http://www.fruitnet.com/eurofruit
- http://www.freshplaza.us/
- http://www.freshplaza.it/

Qui si è deciso di utilizzare la stessa struttura per tutti e tre i siti, in quanto i file da cui si estraevano i dati hanno tutti e tre gli stessi campi. In più abbiamo una procedura per l'allineamento.

3.4 Activity Diagram

Alla prima accensione il sistema imposta la finestra temporale su cui recuperare i dati a un mese, mentre nelle altre esecuzioni lo imposta ad una settimana. Viene poi fatto l'ascesso al sito e recuperato il file XML associato. Se non posso direttamente accedere ai dati dall'XML, lo rielaboro in modo da renderlo possibile, se no procedo direttamente alla fase successiva. Ora che è possibile accedere ai dati grezzi li trasformo in modo da renderli uguali allo schema della tabella di destinazione, se ci sono problemi durante questa fase viene inviata una mail per notificare l'errore. Se tutto va a buon fine salvo i dati sulla tabella di appoggio anche qui notificando eventuali errori. Ora i dati sono nella tabella di appoggio, e procedo con allineare la tabella principale con i nuovi dati presenti su quella di appoggio. A questo punto i dati più aggiornati sono stati salvati sul database. Il sistema si ferma fino a che non è passato un giorno e riesegue il processo per reperire le nuove informazioni.

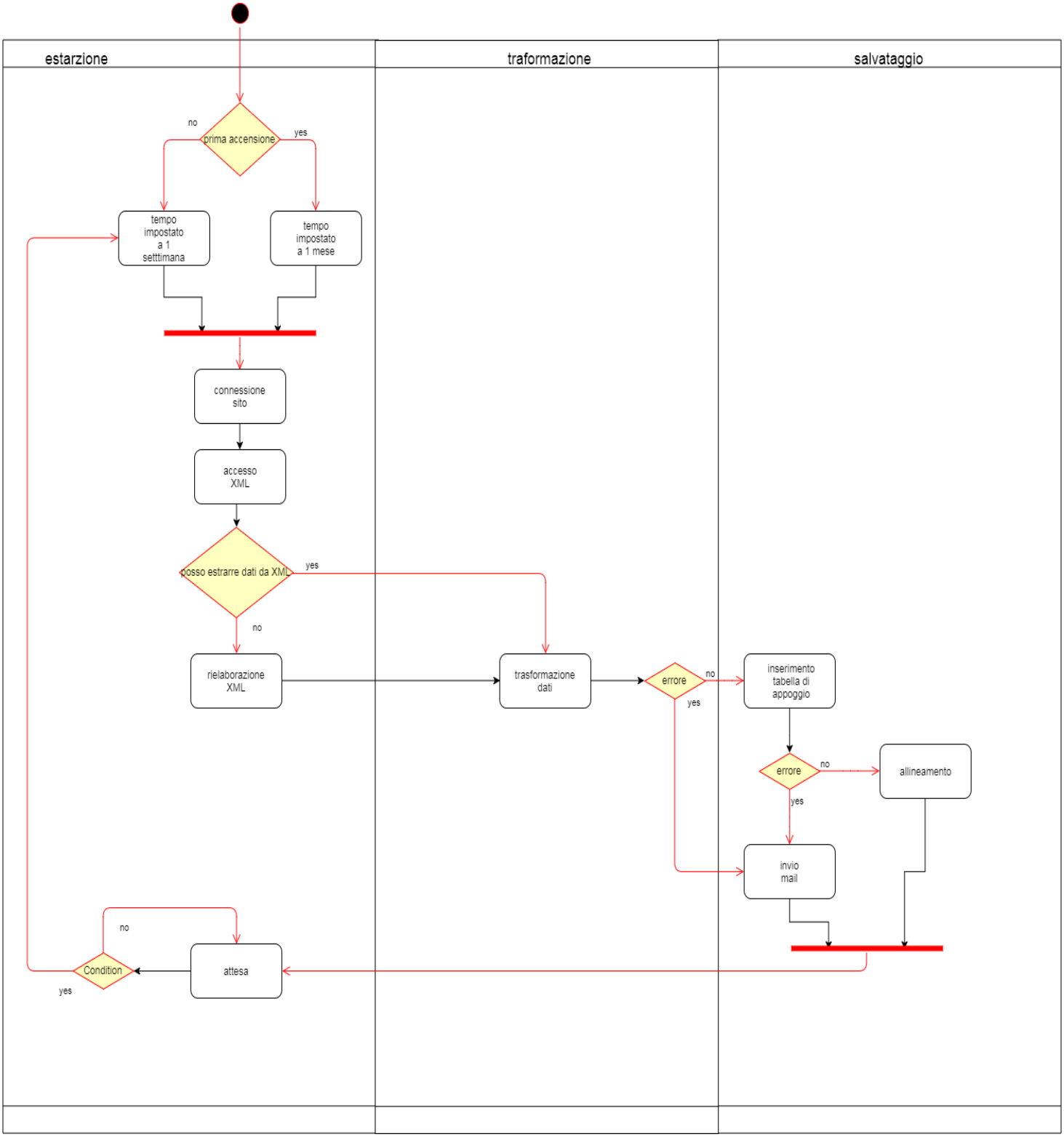

Figura 3.3: Activity Diagram caso generale

Capitolo 4

Implementazione

In questo capitolo viene analizzato il progetto dal punto di vista implementativo. Viene descritto come il sistema reperisce i file, da cui recupera i dati, e come li trasforma per permettere a Kettle di lavorare su di essi. Viene anche mostrato come vengono eseguiti e cosa fanno i job e le trasformazioni.

4.1 Struttura del progetto

L'avvio del sistema parte dalla classe *main.java*, il quale imposta le impostazioni di schedule e fa partire, *CustomTask.java*, il quale va ad eseguire in ordine *Job01.java*, *Job02.java* e *Job03.java* attraverso *JobStarter.java*. Queste 3 classi sono quelle che eseguono il lavoro di estrazione, trasformazione e salvataggio dei dati, andando a reperire i file necessari e facendo partire i job di Kettle.
Job01.java esegue il download e la generazione dei file da cui andare ad estrarre i dati. *Job02.java* avvia il job *assistenzaMarketnews.KJB*, che estrae i dati di mercato, passandogli, di volta in volta, percorsi dei file come parametri. Mentre *Job03.java* avvia il job *caricaArticoli.KJB*, che recupera gli articoli di giornale, anche qui passando i percorsi dei file di volta in volta. La classe *IJob.java* è un'interfaccia che viene implementata da *AbstractJob.java*, la quale è una classe astratta da cui derivano quelle citate sopra.
Per il download e l'elaborazione dei file si usano le seguenti classi ognuna specifica per la categoria di sito su cui si va ad agire:
Iplugin.java, *DownloadVeronamercato.java*, *DownloadArticleBasePlugin.java*, *DownloadMarketnewsBasePlugin.java*. La prima è un'interfaccia che viene implementata dalle altre tre classi. Da queste 3 classi poi derivano altre classi le quali vanno solo a modificare l'URL e il nome del file che si genera.

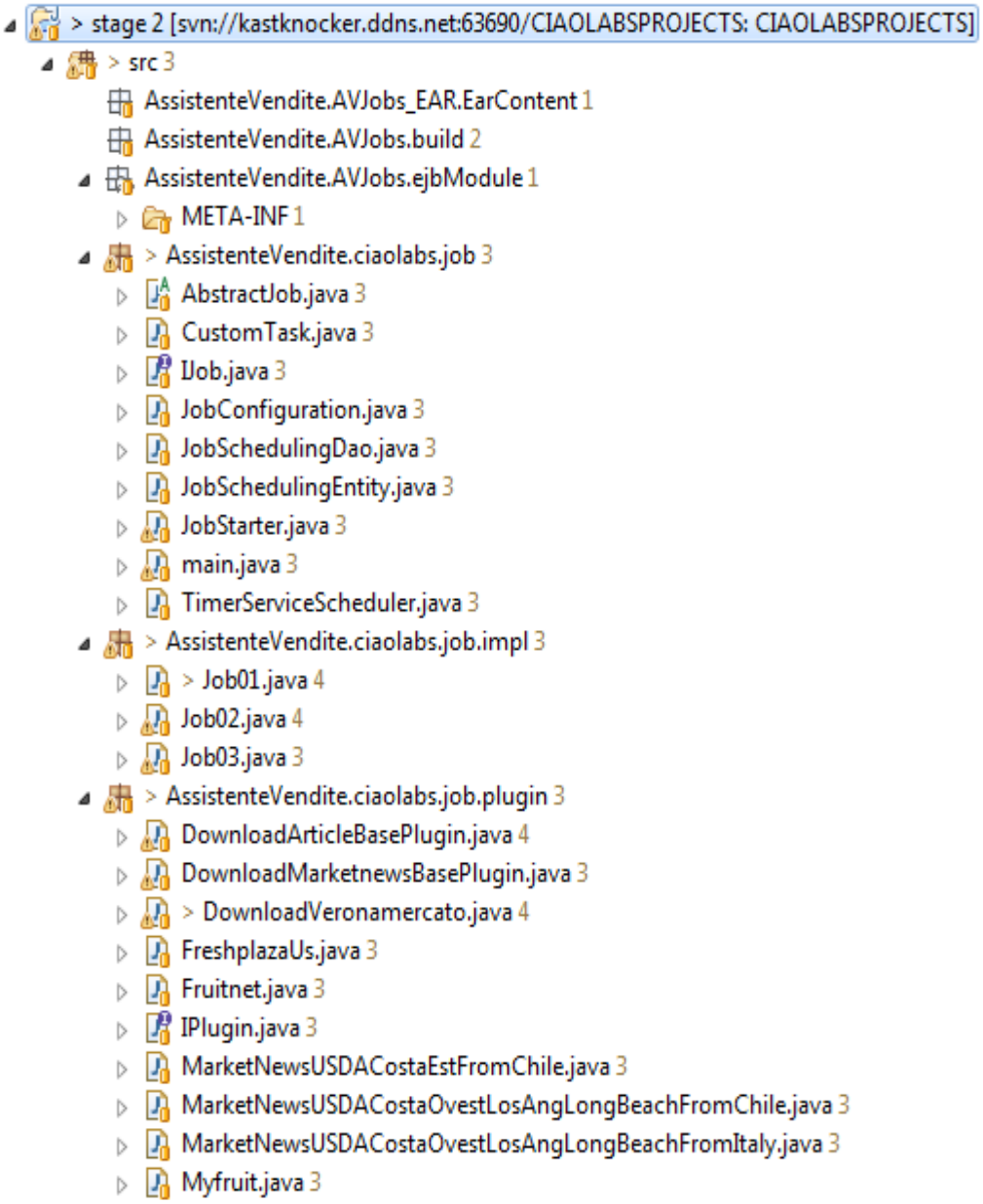

Figura 4.1: Struttura del progetto

4.1.1 main.java

La classe main serve esclusivamente ad impostare la schedulazione, giornaliera, della classe CustomTask e ad avviarla. Questo lo implementa usando il metodo schedule della classe Timer.

```
public class main {
        public static void main(String [] args) {

                  Calendar calendar = Calendar.getInstance();
                calendar.set(Calendar.DAY_OF_WEEK,Calendar.MONDAY);
                calendar.set(Calendar.HOUR_OF_DAY, 0);
                calendar.set(Calendar.MINUTE, 0);
                calendar.set(Calendar.SECOND, 0);
                calendar.set(Calendar.MILLISECOND, 0);

                        // Instantiate Timer Object
                        Timer time = new Timer();
                        time.schedule(new CustomTask(), calendar.getTime(),
                                                TimeUnit.MINUTES.toMillis(10));

        }
}
```

Codice 4.1: classe main

4.1.2 Classe CustomTask

Questa classe serve perché permette di essere schedulata, estende la classe astratta TimerTask ed implementa il metodo run. Questo metodo va a eseguire il metodo executePlugin della classe JobStarter passandogli la posizione di Job01, Job02 e Job3 in modo da eseguirli

```
public class CustomTask extends TimerTask {

        @Override
        public void run() {
                // TODO Auto-generated method stub
                String[] jobConfigurations = new String[]{
                                "AssistenteVendite.ciaolabs.job.impl.Job01",
                                "AssistenteVendite.ciaolabs.job.impl.Job02",
                                "AssistenteVendite.ciaolabs.job.impl.Job03"
                        };

                        JobStarter pluginStarter = new JobStarter();
                        pluginStarter.executePlugin(jobConfigurations);
        }

}
```

Codice 4.2: classe CustomTask

4.1.3 Classe JobStarter

La funzione principale e quella di implementare il metodo executePlugin che prende in ingresso una lista, con i percorsi, degli IJob e va ad eseguire il metodo execute di questi. Implementa anche il metodo getObjectByReflection che crea e ritorna un'istanza della classe IJob il cui percorso gli viene passato in ingresso.

```
public class JobStarter {

	private final Logger logger = Logger.getLogger(JobStarter.class.getName());

	public void executePlugin(String[] pluginClassPaths){
		if(pluginClassPaths == null || pluginClassPaths.length == 0){
			return;
		}

		IJob[] plugins = new IJob[pluginClassPaths.length];

		for(int i=0 ; i<pluginClassPaths.length ; i++){
			plugins[i] = getObjectByReflection(pluginClassPaths[i]);
		}

		for(IJob tmpPlugin : plugins){
			try{
				tmpPlugin.execute();
			}catch(Exception e){
				logger.log(Level.SEVERE,
				"JobStarter.executePlugin(String[])", e);
			}
		}
	}

	@SuppressWarnings("unchecked")
	public static <T extends IJob> T getObjectByReflection(String classPath) {
		Object decodedObject = null;

		@SuppressWarnings("rawtypes")
		Class thisClass = null;
		try {
			thisClass = Class.forName(classPath);
			decodedObject = thisClass.newInstance();
		} catch (Exception e) {
			Logger.getLogger(JobStarter.class.getName()).log(Level.SEVERE,
				"JobStarter.getObjectByReflection(String)", e);
			return null;
		}

		if (decodedObject instanceof IJob)
			return (T) decodedObject;
		else
			return null;
	}

}
```

Codice 4.3: classe JobStarter

4.1.4 Classe astratta AbstractJob

Implementa l'interfaccia IJob e definisce il metodo executeImpl. Il metodo execute stampa a schermo l'inizio dell'esecuzione del job, esegue il metodo executeImpl e stampa se l'esecuzione è andata a buon fine o no. Da questa classe derivano Job01, Job02 e Job03. L'interfaccia IJob è usata per poter andare a eseguire queste classi, creando una lista e invocando il metodo execute all'interno di un for.

```
public abstract class AbstractJob implements IJob {

        protected String jobName;
        protected JobConfiguration jobConfiguration;

        protected abstract void executeImpl();

        public AbstractJob(){
                this.jobName = this.getClass().getSimpleName();
        }

        public AbstractJob(String jobName, JobConfiguration jobConfiguration){
                this.jobName = this.getClass().getName();
                this.jobConfiguration = jobConfiguration;
        }

        @Override
        public String execute() {
                System.out.println(String.format("%s - Execute start", jobName));
                try {
                        executeImpl();
                        System.out.println(String.format("%s - Execute end correctly",
                                jobName));

                        return String.format("%s - Execute end correctly", jobName);
                } catch (Exception e){
                        System.out.println(String.format("%s - Execute end with errors",
                                                         jobName));

                        return String.format("%s - Execute end with errors", jobName);
                }
        }

        public JobConfiguration getJobConfiguration() {
                return jobConfiguration;
        }

        public void setJobConfiguration(JobConfiguration jobConfiguration) {
                this.jobConfiguration = jobConfiguration;
        }


}
```

Codice 4.4: classe astratta AbstractJob

4.1.4.1 Classe Job01

Il metodo executeImpl di questa classe genera una lista di istanze di classi IPlugin e ne chiama il metodo execute. Questo genera i file da cui poi noi andremo ad estrarre, trasformare e salvare i dati.

```
public class Job01 extends AbstractJob {

	@Override
	public Date getNextExecutionDate() {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	protected void executeImpl() {
		// TODO Auto-generated method stub
		List<IPlugin> listaPlugin = new ArrayList<IPlugin>();

		listaPlugin.add(new DownloadMarketnewsBasePlugin());
		//listaPlugin.add(new MarketNewsUSDACostaOvestLosAngLongBeachFromItaly());
		//listaPlugin.add(new MarketNewsUSDACostaOvestLosAngLongBeachFromChile());
		listaPlugin.add(new MarketNewsUSDACostaEstFromChile());
		listaPlugin.add(new DownloadVeronamercato());
		listaPlugin.add(new DownloadArticleBasePlugin());
		listaPlugin.add(new Fruitnet());
		listaPlugin.add(new FreshplazaUs());
		listaPlugin.add(new Myfruit());

		for(IPlugin plugin : listaPlugin){
			boolean esito = plugin.execute();
			System.out.println("Esecuzione "+plugin.getPluginName()+":
"+esito);
		}
	}
}
```

Codice 4.5: classe Job01

4.1.4.2 Classe Job02

Il metodo executeImpl esegue il job assistenzaMarketnews. Per prima cosa inizializza l'ambiente Kettle, dopo accede al repository stage ed entra nella cartella pentaho. Da qui preleva il job e ne crea un'istanza in una classe Job di Kettle, gli passa le variabili che rappresentano i percorsi dei file e va ad eseguirlo. Infine attende che il Job abbia finito la propria esecuzione e stampa se ci sono stati errori.

```
public class Job02 extends AbstractJob {

	@Override
	public Date getNextExecutionDate() {
```

```
            // TODO Auto-generated method stub
            return null;
        }

        @Override
        protected void executeImpl() {
            // TODO Auto-generated method stub
            String filename = "assistenzaMarketnews";
            System.out.println("***********");

        try {
            //inizializza a l'ambiente Kettle
                KettleEnvironment.init(true);

                //acceso al repository
                RepositoriesMeta repositoriesMeta = new RepositoriesMeta();
                repositoriesMeta.readData();
                RepositoryMeta repositoryMeta =
                                repositoriesMeta.findRepository("stage");
                KettleFileRepository repository = new KettleFileRepository();
                repository.init(repositoryMeta);
                repository.connect("admin", "admin");

                RepositoryDirectoryInterface directory =
                                        repository.loadRepositoryDirectoryTree();
                directory = directory.findDirectory("penthao");

                //istanzia il job
                        JobMeta jobMeta = repository.loadJob(filename, directory,
                                        null, null);
                Job job = new Job(repository, jobMeta);

                //passaggio variabili
                job.initializeVariablesFrom(null);
                job.getJobMeta().setInternalKettleVariables(job);
                job.copyParametersFrom(job.getJobMeta());

                job.setParameterValue("MrketNewFile",
                        "C:\\Users\\deto\\workspace\\stage\\information.xml");
                job.setParameterValue("VeronaMercatoFile",
                        "C:\\Users\\deto\\workspace\\stage\\veronaMercato.xml");
                job.activateParameters();

            job.setLogLevel(LogLevel.ROWLEVEL);

            //fa partire il job
                job.execute(0, null);
                job.waitUntilFinished();

                if (job.getErrors()!=0) {
                        System.out.println("Error encountered!");
                }
        } catch (KettleException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
}
```

*Codice **4.6: classe Job02***

4.1.4.3 Classe Job03

Essenzialmente si comporta uguale alla classe Job02 ma chiama il Job assistenzaArticoli più volte. Ogni volta lo esegue passandogli parametri diversi, ed infine anche qui notifica se c'è stato un errore

```
public class Job03 extends AbstractJob {

        @Override
        public Date getNextExecutionDate() {
                // TODO Auto-generated method stub
                return null;
        }

        @Override
        protected void executeImpl() {
                // TODO Auto-generated method stub
                String filename = "assistenzaArticoli";

            try {
                //inizializza a l'ambiente Kettle
                        KettleEnvironment.init(true);

                        //accesso al repository
                        RepositoriesMeta repositoriesMeta = new RepositoriesMeta();
                        repositoriesMeta.readData();
                        RepositoryMeta repositoryMeta =
                                repositoriesMeta.findRepository("stage");

                        KettleFileRepository repository = new KettleFileRepository();
                        repository.init(repositoryMeta);
                        repository.connect("admin", "admin");

                        RepositoryDirectoryInterface directory =
                                                repository.loadRepositoryDirectoryTree();
                        directory = directory.findDirectory("penthao");

                        //istanzia il job
                        JobMeta jobMeta = repository.loadJob(filename, directory, null,
                                        null);
                        Job job = new Job(repository, jobMeta);

                        //passaggio variabili
                        job.initializeVariablesFrom(null);
                        job.getJobMeta().setInternalKettleVariables(job);
                        job.copyParametersFrom(job.getJobMeta());

                        job.setParameterValue("file",
                                "C:\\Users\\deto\\workspace\\stage\\fruitnet.txt");
                        job.activateParameters();

                        //fa partire il job
                        job.execute(0, null);
                        job.waitUntilFinished();

                        //cambio variabili
                        job.setParameterValue("file",
                                "C:\\Users\\deto\\workspace\\stage\\freshplaza.txt");
                        //fa partire il job
                        job.execute(0, null);
                        job.waitUntilFinished();

                        //cambio variabili
```

```
                        job.setParameterValue("file",
                                "C:\\Users\\deto\\workspace\\stage\\freshplazaUS.txt");
                        //fa partire il job
                        job.execute(0, null);
                        job.waitUntilFinished();

                        if (job.getErrors()!=0) {
                                System.out.println("Error encountered!");
                        }
            } catch (KettleException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }
}
```

Codice 4.7: classe Job03

4.1.5 Interfaccia IPlugin

Questa interfaccia viene implementa dai plugin per il download e la generazione dei file di dati. Principalmente si usa per poter creare una lista di essi e andare ad eseguire il metodo execute di tutti in un ciclo for.

```
public interface IPlugin {

        public String getPluginName();
        public boolean execute();

}
```

Codice 4.8: interfaccia IPlugin

4.1.5.1 Classe DownloadMarketnewsBasePlugin

Questa è la classe base per il download dei dati dal sito https://www.marketnews.usda.gov/, da questa ereditano 3 classi che sono MarketNewsUSDACostaEstFromChile, MarketNewsUSDACostaOvestLosAngLongBeachFromChile, MarketNewsUSDACostaOvestLosAngLongBeachFromItaly.

Queste modificano le variabili templateURL e outputFileName, che contengono rispettivamente l'URL da cui scaricare l'XML e il file su cui salvarlo. Il metodo execute di questa classe crea un URL formato da templateURL, la data di oggi meno una settimana e la data di oggi, il quale serve per andare sulla pagina che ha i dati inerenti all'ultima settimana. Dopo di che legge i byte da questo url e li scrive su outputFileName, infine se non ci sono errori ritorna True

```
public class DownloadMarketnewsBasePlugin implements IPlugin{

        protected String templateURL = "https://www.marketnews.usda.gov/mnp/fv-report-top-
                riceDafilters&commAbr=KIWI&shipNavClass=&portal=fv&repType=shipPily&movNav
                Class=&Go=Go&locName=ITALY+IMPORTS+-+PORTS+OF+ENTRY+PHILADELPHIA+AREA+AND+
                NEW+YORK+CITY+AREA&type=shipPrice&locAbrAll=&navClass=FRUITS&organic=&navT
                ype=byComm&environment=&varName=HAYWARD&locAbr=ITPANYPOE&volume=&stateID=&
                commName=KIWIFRUIT&termNavClass=&repDate=%s&endDate=%s&format=xml&rebuild=
                false&_=1458076985404";
        protected String outputFileName = "information.xml";

        public DownloadMarketnewsBasePlugin(){

        }

        @Override
        public String getPluginName() {
                return "DownloadMarketnews";
        }

        @Override
        public boolean execute(){
                try{
                        DateFormat dateFormat = new SimpleDateFormat("MM/dd/yyyy");

                        Date fine = new Date();

                        Calendar cal = Calendar.getInstance();
                        cal.setTime(fine);
                        cal.add(Calendar.DATE, -7);
                        Date inizio = cal.getTime();

                        String startDate = dateFormat.format(inizio);
                        String endDate = dateFormat.format(fine);

                        System.out.println("***********");
                        System.out.println(String.format("*** TEMPLATE: %s",
                                                        templateURL));
                        System.out.println(String.format("*** START_DATE: %s",
                                                        startDate));
                        System.out.println(String.format("*** END_DATE: %s", endDate));


                        String stringUrl = String.format(templateURL, startDate, endDate);

                        URL url = new URL(stringUrl);
                        ReadableByteChannel rbc = Channels.newChannel(url.openStream());
                        FileOutputStream fos = new FileOutputStream(outputFileName);
                        fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);

                }catch(Exception e){
                        e.printStackTrace();
                        return false;
                }finally {
                        System.out.println("***********");
                }

                return true;
        }
}
```

Codice 4.9: Classe DownloadMarketnewsBasePlugin

4.1.5.2 Classe DownloadVeronamercato

Questa è la classe per il download dei dati dal sito http://www.veronamercato.it/, nessuna classe eredita da lei, ed usa la libreria esterna di JTidy. Per prima cosa il metodo execute recupera il numero di listino da inserire nell'URL per scaricare i dati inerenti a oggi, dopo lo passa a JTidy che serve per rendere well-formed l'XML, non essendolo di default. Ciò che esce dal parser poi viene scritto su outputFileName, il quale viene riaperto in scrittura e nella prima riga viene inserito ' " " ' in modo che sia well-formed al cento per cento. Per recuperare il numero di listino ho estratto, grazie a jsoup, dal seguente URL www.veronamercato.it/common/ , il numero di listino recuperandolo dalla riga: "<input type="hidden" name="id_listino" value="5008">". Per farlo ho recuperato l'attributo value dal tag < input > e in un ciclo for le ho inserite all'interno di un Vector, poi ho preso il secondo elemento del vettore che contiene il numero.

```
public class DownloadVeronamercato implements IPlugin {

	protected String templateURL = "http://www.veronamercato.it/common";
	protected String outputFileName = "veronaMercato.xml";

	@Override
	public String getPluginName() {
		// TODO Auto-generated method stub
		return "DownloadVeronamercato";
	}

	@Override
	public boolean execute() {
		// TODO Auto-generated method stub
		//recupero il numero di listino di oggi
		URL url;
		try {
			org.jsoup.nodes.Document list = Jsoup.connect(templateURL).get();
			//estraggo i tag = input
			Elements links = list.select("input");
			Vector<String> info = new Vector<String>();
			for(Element link : links){
				//li salvo su un vettore
				info.add(link.attr("value"));
			}
			//accedo alla pagina di oggi
			String listino = info.elementAt(1);
			templateURL = templateURL + "/query_results2.php?action=1&id_
				listino="+ listino +"&descrizione=0&classe_prod=0";
			//la rendo un xml well-formed con tidy
			Tidy tidy = new Tidy();
			tidy.setMakeClean(true);
			tidy.setXmlOut( true);

			url = new URL(templateURL);

			//lo scrivo su file
```

```
                        FileOutputStream fileOutputStream = new
                                        FileOutputStream(outputFileName);

                        Document doc = tidy.parseDOM( url.openStream(), fileOutputStream);

                        RandomAccessFile file = new RandomAccessFile(outputFileName,"rw");
                        file.skipBytes(62);
                        file.writeBytes(" \" \"> <html> ");
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                        return false;
                }
                return true;
        }
}
```

Codice 4.10: Classe DownloadMarketnewsBasePlugin

4.1.5.3 Classe DownloadArticleBasePlugin

Questa è la classe per il download dei link e della descrizione di essi dalle Homepage dei siti: http://www.fruitnet.com/eurofruit, http://www.freshplaza.us/, http://www.freshplaza.it/, da questa ho fatto ereditare 2 classi; FreshplazaUs, Fruitnet. Anche qui come prima le classi modificano solo le 2 variabili sopra citate; per questa classi ho avuto vari problemi a reperire un file XML visto che JTidy non riusciva a fare il parser, perciò ho deciso di ripiegare sulla generazione di un txt. Per fare ciò ho sfruttato la libreria esterna jsoup, che permette il parser di un file XML su una determinata espressione, grazie a questa ho estratto dall'URL i tag a[href] e ho estratto i value abs:href che rappresentano i link. Poi ho scritto il metodo trim che recupera la descrizione del link, infine ho scritto su un txt, per ogni riga, il link + TAB + la descrizione.

```
public class DownloadArticleBasePlugin  implements IPlugin {

        protected String templateURL = "http://www.freshplaza.it/";
        protected String outputFileName = "freshplaza.txt";

        @Override
        public String getPluginName() {
                return "Downloadfreshplaza";
        }

        @Override
        public boolean execute(){
                try {
                        System.out.println("***********");
                        Document doc = Jsoup.connect(templateURL).get();
                        //estrae i link href dal sito
                        Elements links = doc.select("a[href]");

                        FileWriter html = new FileWriter(outputFileName);
```

```
                        for(Element link : links) {
                                //scrive i link e il testo sul file di testo
                                html.write(link.attr("abs:href") + "     " +
                                                trim(link.text(), 500) + "\r\n");
                                }
                                html.close();
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return true;
        }
        private static String trim(String s, int width) {
                if (s.length() > width)
                        return s.substring(0, width-1) + ".";
            else
                    return s;
            }
}
```

Codice 4.11: Classe DownloadMarketnewsBasePlugin

4.2 Trasformazioni e Job

Di seguito andremo ad analizzare i Job e le trasformazioni che sono state create in questo progetto. Si sono creati 2 job, *assistenzaMarketnews* e *assistenzaArticoli*, i quali utilizzano le trasformazioni che siamo andati a definire. Il primo utilizza 2 trasformazioni, *caricaMarketnews* e *VeronaMercato*, mentre il secondo ne usa solo una che è *caricaArticoli*. Ho deciso di usare i Job perché le trasformazioni hanno un'esecuzione parallela dei passi. Noi abbiamo bisogno che prima i dati vengano caricati nella tabella di appoggio, e solo successivamente venga chiamata la procedure di allineamento.

4.2.1 Job assistenzaMarketnews

Questo Job ha lo scopo di andare a inserire i dati all'interno dello schema "PRODOTTI" ed è composto, come si può notare nella figura 4.2, da due percorsi paralleli. Quello superiore ha il compito di inserire i dati nella tabella "MARKETNEWS_ARCHIVIO", mentre quello inferiore nella tabella "VERONAMERCATO_ARCHIVIO". I due percorsi hanno la stessa struttura, per prima cosa vanno ad eseguire la trasformazione, che inserisce i dati all'interno della tabella di appoggio. Dopo, se non ci sono stati errori, va ad eseguire il codice SQL per chiamare la procedure di allineamento. Se per qualche motivo c'è un errore all'interno di una o entrambe le trasformazioni, ho fatto in modo che venga inviata una mail al gestore del sistema, nella quale viene allegato il file di log che contiene l'errore.

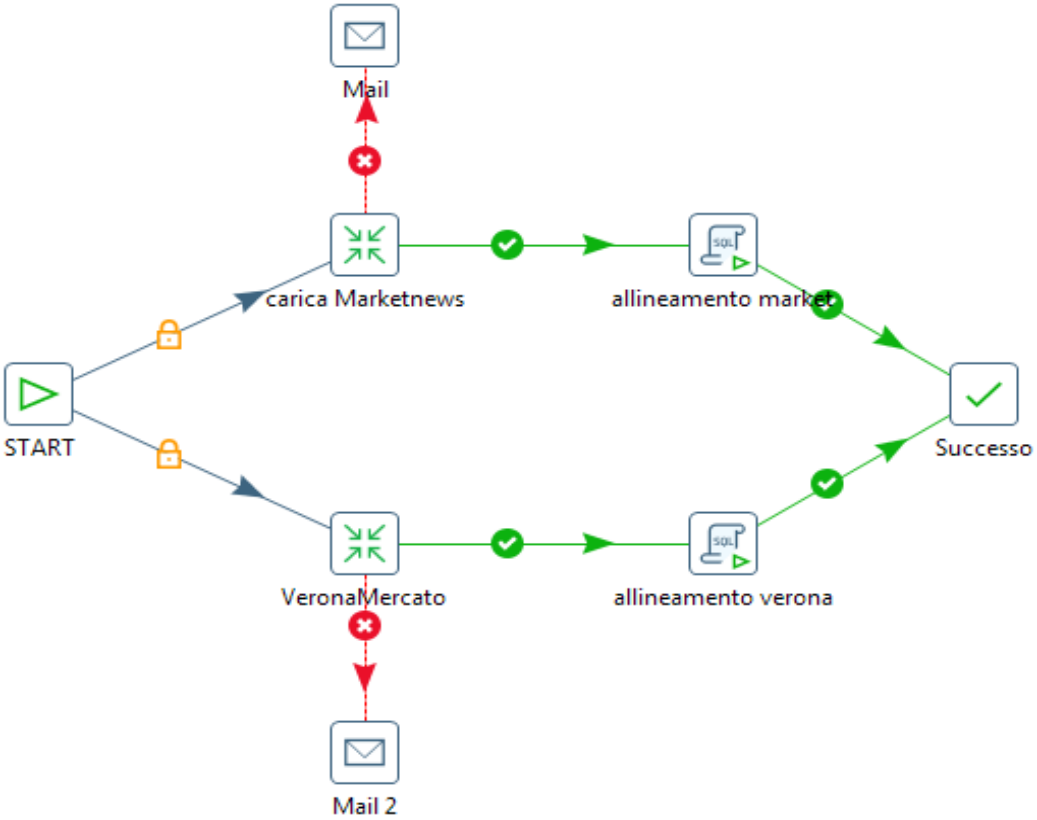

Figura 4.2 Job assistenzaMarketnews

4.2.1.1 Trasformazione caricaMarketnews

È la trasformazione che estrae ed inserisce i dati nella tabella MARKETNEWS_TEMP dagli xml e crea la chiave hash. È composta dai seguenti passi:

- *file*: recupera il nome del file in cui sono memorizzati i dati che viene passato come parametro alla trasformazione
- *estrazione dati*: è un get data from xml con il seguente XPath "/report/item" ed estrae i campi della tabella MARKETNEWS_TEMP, tranne hash, dal file passato dal passo precedente
- *generazione chiave*: genera una chiave hash su tutti i campi in uscita dal passo precedente e la inserisce nel campo hash
- *sort row – unique row*: la composizione di questi 2 passi servono semplicemente per eliminare i dati duplicati, il primo ordina i dati sulla

chiave hash e il secondo, il quale ha bisogno di un input ordinato, elimina le righe duplicate

- *table output*: inserisce i dati ricevuti da passo precedente nella tabella MARKETNEWS_TEMP applicando prima un troncamento su di essa

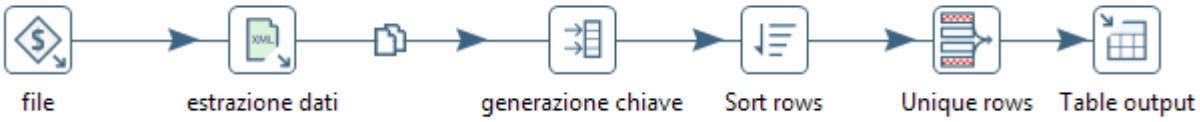

Figura 4.3 Trasformazione caricaMarketnews

4.2.1.2 Trasformazione VeronaMercato

È la trasformazione che estrae ed inserisce i dati nella tabella VERONAMERCATO_TEMP dal xml e crea la chiave hash. È composta dai seguenti passi:

- *file*: recupera il nome del file in cui sono memorizzati i dati che viene passato come parametro alla trasformazione
- *estrazione dati*: è un get data from xml con il seguente XPath "/html/body/table/tr/td/table/tr/td" ed estrae un singolo campo dal file passato dal passo precedente
- *Row flattener*: prende in ingresso le righe estratte nel passo precedente e le divide su 2 campi, categorie e dati, il primo valore lo inserisce in categorie, il secondo in dati, il terzo in categorie, ecc.
- *Selected values*: rimuove il campo categorie dallo stream, questo ci serve per il passo successivo in modo da avere solo i dati
- *Row flattener2*: prende il campo in uscita dal precedente passo e divide le righe distribuendole, come sopra, nei 10 campi della tabella VERONAMERCATO_TEMP, escludendo hash e data
- *Get System Info*: recupera la data di esecuzione della trasformazione e la aggiunge allo stream
- *generazione chiave*: genera una chiave hash su tutti i campi in uscita dal passo precedente e la inserisce nel campo hash

- *Table output*: inserisce i dati ricevuti da passo precedente nella tabella VERONAMERCATO_TEMP applicando prima un troncamento su di essa

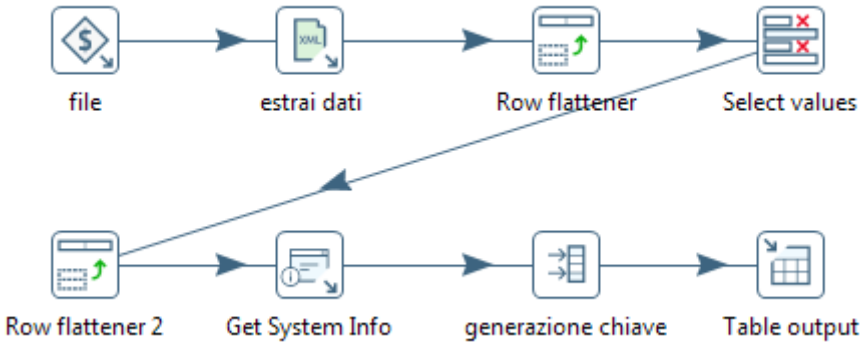

Figura 4.4 Trasformazione VeronaMercato

4.2.2 Job AssistenzaArticoli

Questo Job ha il compito di andare ad inserire i dati all'interno dello schema "ARTICOLI". Per prima cosa esegue la trasformazione caricaArticoli che carica i dati all'interno della tabella "ARTICOLI_TEMP" che è quella di appoggio. Poi, dopo che la trasformazione ha terminato, va ad eseguire la procedure di allineamento dei dati e ad inserire i nuovi dati all'interno della tabella "ARTICOLI_ARCHIVIO". Anche qui se per qualche motivo ci dovesse essere un errore all'interno della trasformazione, ho fatto in modo che venga inviata una mail al gestore del sistema, alla quale viene allegato il file di log che contiene l'errore.

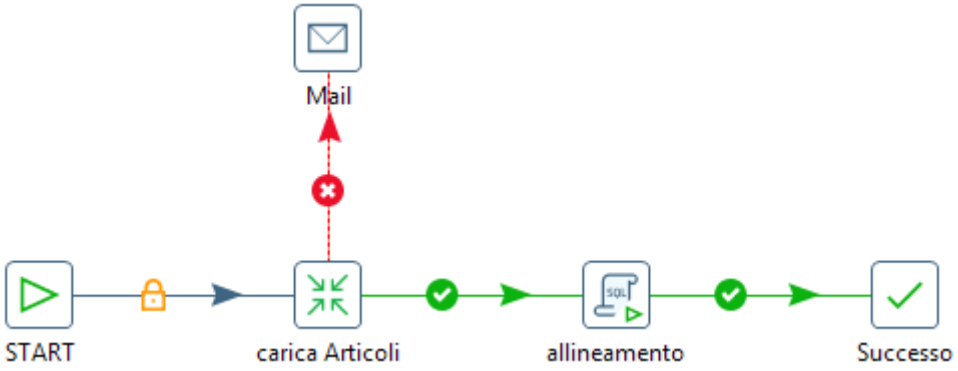

Figura 4.5 Job AssistenzaArticoli

4.2.2.1 Trasformazione caricaArticoli

È la trasformazione che estrae ed inserisce i dati nella tabella ARTICOLI_TEMP dai txt. È composta dai seguenti passi:

- *file*: recupera il nome del file in cui sono memorizzati i dati che viene passato come parametro alla trasformazione
- *estrazione dati*: è un imput CSV con delimitatore rappresentato da un "TAB" ed i campi link e descrizione
- *valuta dati*: guarda quali link hanno un riscontro con l'espressione regolare "^.+/article.+$" e salva il risultato nel campo result
- *selezione articoli*: tiene solo quelle righe che hanno il campo result a True, cioè che hanno avuto una corrispondenza nel passo precedente
- *ordina dati – rimozione duplicati*: come nel passo spiegato sopra elimina i duplicati se c'è ne sono
- *Get System Info*: recupera la data di esecuzione della trasformazione e la aggiunge allo stream
- *Table output*: inserisce i dati ricevuti da passo precedente nella tabella ARTICOLI_TEMP applicando prima un troncamento su di essa

Figura 4.6 Trasformazione caricaArticoli

4.3 Codice SQL

In questa sezione andremo a mostrare, il codice SQL per la generazione delle tabelle e delle procedure di allineamento dei dati.

Schema PRODOTTI:

```
CREATE SCHEMA "PRODOTTI";

CREATE TABLE "PRODOTTI"."MARKETNEWS_ARCHIVIO"
(
"commName"                varchar(200),
"cityName"                varchar(200),
"organic"                 varchar(200),
"packageDesc"             varchar(200),
"varietyName"             varchar(200),
"subvarName"              varchar(200),
"gradeDesc"               varchar(200),
"reportDate"              varchar(200),
"lowPriceMin"             varchar(200),
"highPriceMax"            varchar(200),
"mostlyLowMin"            varchar(200),
"mostlyHighMax"           varchar(200),
"season"                  varchar(200),
"itemSizeDesc"            varchar(200),
"color"                   varchar(200),
"envDesc"                 varchar(200),
"unitSale"                varchar(200),
"quality"                 varchar(200),
"condition"               varchar(200),
"appearance"              varchar(200),
"storage"                 varchar(200),
"shelled"                 varchar(200),
"purchases"               varchar(200),
"section"                 varchar(200),
"importExportDesc"        varchar(200),
"supplyTone"              varchar(200),
"demandTone"              varchar(200),
"basisOfSale"             varchar(200),
"marketTone"              varchar(200),
"priceComment"            varchar(200),
"commenti"                varchar(200),
"reportingCity"           varchar(200),
"hash"                    DOUBLE PRECISION,
primary key("hash")
);

CREATE TABLE "PRODOTTI"."MARKETNEWS_TEMP" AS SELECT * FROM
"PRODOTTI"."MARKETNEWS_ARCHIVIO";
ALTER TABLE "PRODOTTI"."MARKETNEWS_TEMP" ADD primary key("hash");

CREATE TABLE "PRODOTTI"."VERONAMERCATO_ARCHIVIO"
(
"hash"                            DOUBLE PRECISION,
"Codice_articolo"         varchar(200),
"Gruppo"                  varchar(200),
"Prodotto"                varchar(200),
"Natura_prodotto"         varchar(200),
"Origine_prodotto"        varchar(200),
"Confezione"              varchar(200),
"Caratteristiche_generali"varchar(200),
"Categoria"               varchar(200),
"Unit_di_misura"          varchar(200),
"Prezzo_in_euro"          varchar(200),
"data"                    varchar(200),
primary key("hash")
);
```

```
CREATE TABLE "PRODOTTI"."VERONAMERCATO_TEMP" AS SELECT * FROM
"PRODOTTI"."VERONAMERCATO_ARCHIVIO";
ALTER TABLE "PRODOTTI"."VERONAMERCATO_TEMP" ADD primary key("hash");

-----------------------------------------------------------------------------------------------------------

CREATE FUNCTION "PRODOTTI"."ALLINEAMENTO_MARKETNEWS"()
RETURNS void AS $$
BEGIN

	INSERT INTO "PRODOTTI"."MARKETNEWS_ARCHIVIO" select * from
"PRODOTTI.MARKETNEWS_TEMP" where hash not in
		(select hash from "PRODOTTI"."MARKETNEWS_ARCHIVIO");
END;
$$
LANGUAGE plpgsql;


CREATE FUNCTION "PRODOTTI"."ALLINEAMENTO_VERONAMERCATO"()
RETURNS void AS $$
BEGIN

	INSERT INTO "PRODOTTI"."VERONAMERCATO_ARCHIVIO" select * from
"PRODOTTI"."VERONAMERCATO_TEMP" where hash not in
		(select hash from "PRODOTTI"."VERONAMERCATO_ARCHIVIO");
END;
$$
LANGUAGE plpgsq;
```

Codice 4.12: Schema PRODOTTI

Schema ARTICOLI:

```
CREATE SCHEMA "ARTICOLI";

CREATE TABLE "ARTICOLI"."ARTICOLI_ARCHIVIO"
(
"link"		varchar(500),
"descrizione"	varchar(500),
"data_"		varchar(200),
primary key("link")
);

CREATE TABLE "ARTICOLI"."ARTICOLI_TEMP" AS SELECT * FROM "ARTICOLI"."ARTICOLI_ARCHIVIO";
ALTER TABLE "ARTICOLI"."ARTICOLI_TEMP" ADD primary key("link");

-----------------------------------------------------------------------------------------

CREATE FUNCTION "ARTICOLI"."ALLINEAMENTO_ARTICOLI"()
RETURNS void AS $$
BEGIN

	INSERT INTO "ARTICOLI"."ARTICOLI_ARCHIVIO" select * from "ARTICOLI"."ARTICOLI_TEMP"
where link not in
		(select link from "ARTICOLI"."ARTICOLI_ARCHIVIO");
END;
$$
LANGUAGE plpgsql;
```

Codice 4.13: Schema ARTICOLI

Capitolo 5

Test del sistema

In questo capitolo vengono messi in luce i test avvenuti per verificare il corretto funzionamento del sistema.
Gli unici test che si è andati ad operare sono quelli per testare il coretto funzionamento dei job, andando ad eseguirli e analizzando il log restituito da Kettle il quale mostra lo stato di escuzione e eventuali errori. È possibile impostare il livello di dettaglio del log.
Di segito mostro il log di base del job assistenza Articoli:

2017/07/05 12:13:37 - Spoon - Avvio job...

2017/07/05 12:13:37 - assistenzaArticoli - Inizio dell'esecuzione del job

2017/07/05 12:13:37 - assistenzaArticoli - Inizio entry [carica Articoli]

2017/07/05 12:13:38 - caricaArticoli - Spedizione iniziata per la trasformazione [caricaArticoli]

2017/07/05 12:13:38 - Table output.0 - Connected to database [assistenteVendite] (commit=1000)

2017/07/05 12:13:38 - estrazione dati.0 - Riga d'intestazione saltata nel file 'C:\Users\deto\workspace\stage\fruitnet.txt'

2017/07/05 12:13:38 - estrazione dati.0 - Elaborazione terminata (I=495, O=0, R=0, W=494, U=0, E=0)

2017/07/05 12:13:38 - valuata dati.0 - Elaborazione terminata (I=0, O=0, R=494, W=494, U=0, E=0)

2017/07/05 12:13:38 - selezione articoli.0 - Elaborazione terminata (I=0, O=0, R=494, W=113, U=0, E=0)

2017/07/05 12:13:38 - ordina dati.0 - Elaborazione terminata (I=0, O=0, R=113, W=113, U=0, E=0)

2017/07/05 12:13:38 - rimozione duplicati.0 - Elaborazione terminata (I=0, O=0, R=113, W=55, U=0, E=0)

2017/07/05 12:13:38 - Get System Info.0 - Elaborazione terminata (I=0, O=0, R=55, W=55, U=0, E=0)

2017/07/05 12:13:38 - Table output.0 - Elaborazione terminata (I=0, O=55, R=55, W=55, U=0, E=0)

2017/07/05 12:13:38 - assistenzaArticoli - Inizio entry [allineamento]

2017/07/05 12:13:38 - assistenzaArticoli - Inizio entry [Successo]

2017/07/05 12:13:38 - assistenzaArticoli - Terminata jobentry [Successo] (risultato=[true])

2017/07/05 12:13:38 - assistenzaArticoli - Terminata jobentry [allineamento] (risultato=[true])

2017/07/05 12:13:38 - assistenzaArticoli - Terminata jobentry [carica Articoli] (risultato=[true])

2017/07/05 12:13:38 - assistenzaArticoli - L'esecuzione del job è terminata

2017/07/05 12:13:38 - Spoon - Il job ha terminato.

2017/07/05 12:14:01 - Spoon - Salva il file o cartella...

2017/07/05 12:14:01 - Spoon - Salva nel repository...

2017/07/05 12:14:02 - DBCache - We wrote 4 cached rows to the database cache!

2017/07/05 12:14:02 - Spoon - Trasformazione aperta.

2017/07/05 12:14:02 - Spoon - Esecuzione trasformazione [caricaArticoli]...

2017/07/05 12:14:02 - Spoon - Esecuzione trasformazione avviata.

2017/07/05 12:14:03 - Spoon - La trasformazione è terminata!

Conclusione e sviluppi futuri

Nell'ambito di questa tesi è stato raggiunto l'obiettivo prefissato.

È stato realizzato un sistema in grado di estrarre i dati richiesti, trasformarli e infine mandarli a salvare, all'interno di un supporto di memorizzazione, cercando di ridurre al minimo la perdita di informazioni.
Inoltre il sistema è automatizzato, cioè dopo che viene avviato non ha più bisogno dell'ausilio dell'utente per funzionare, in quanto viene schedulato in automatico giornalmente.

Alcuni sviluppi futuri per il progetto possono comprendere:

- la creazione di un sistema in grado di visualizzare i dati, in modo da permettere all'utente di poter usare i dati estratti per i propri scopi
- creazione di indici sulle tabelle in modo da rendere il recupero dei dati più facile
- ampliamento fonti da dove andare a reperire i dati.

Bibliografia e Sitografia

[1, 5] Penthao: Matt Casters, Roland Bouman, Jos van Dongen
"Pentaho® Kettle Solutions"(Wiley publishing, Inc.2010)

[2]XPhath : https://www.w3.org/TR/xpath/

[3] HTML Tidy: http://www.html-tidy.org/documentation/

[4] JSoup : https://jsoup.org/