POLITECNICO DI MILANO

V Facoltà di Ingegneria
Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

Architetture per l'importazione automatica di dati genomici e proteomici in un data warehouse integrato

Relatore: Prof. Marco Masseroli, Ph.D

Correlatore: Ing. Giorgio Ghisalberti

Tesi di Laurea di:
Davide Di Stefano
Matr. n. 717856

Anno Accademico 2009-2010

POLITECNICO DI MILANO

V Facoltà di Ingegneria
Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

Architetture per l'importazione automatica di dati genomici e proteomici in un data warehouse integrato

Laureando: _____
(Davide Di Stefano)

Relatore: _____
(Prof. Marco Masseroli)

Anno Accademico 2009-2010

Ringraziamenti

Desidero innanzitutto ringraziare il Prof. Masseroli per le numerose ore dedicate alla mia Tesi. Inoltre, ringrazio sentitamente l'Ing. Ghisalberti che è stato sempre disponibile a dirimere i miei dubbi durante la stesura di questo lavoro. Intendo poi ringraziare l'Istituto Politecnico di Milano per avermi fornito testi e dati indispensabili per la realizzazione della Tesi.
Infine, ho desiderio di ringraziare con affetto i miei genitori per il sostegno e il grande aiuto ricevuti durante il mio percorso universitario.

Indice

Indice delle illustrazioni

Indice delle tabelle

Glossario

Data warehouse (DW): termine inglese traducibile con magazzino di dati, consiste in un archivio informatico contenente i dati di un'organizzazione. I DW sono progettati per consentire di produrre facilmente relazioni e analisi.

Fenotipo: la effettiva, totale manifestazione fisica di un organismo, in opposizione al suo genotipo (le istruzioni ereditate che porta, che possono essere o non essere espresse). Questa distinzione genotipo-fenotipo fu proposta da Wilhelm Johannsen nel 1911 per rendere chiara la differenza tra l'eredità di un organismo e cosa l'eredità produce.

Framework: nella produzione del software rappresenta una struttura di supporto su cui un software può essere organizzato e progettato. Alla base di un framework vi è sempre una serie di librerie di codice utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software, come ad esempio un IDE, un debugger, o altri strumenti ideati per aumentare la velocità di sviluppo del prodotto finito.

Genoma: tutte le sequenze di acido nucleico che costituiscono il patrimonio genetico completo di un organismo.

Proteoma: l'insieme delle proteine di un organismo o di un sistema biologico.

Workflow: termine inglese traducibile con flusso di lavoro, è inteso come la sequenza delle operazioni da svolgere nel tempo.

1 Sommario

Lo sviluppo delle tecnologie informatiche e delle biotecnologie ha contribuito alla nascita di discipline come la bioinformatica che sono in continua evoluzione. La bioinformatica costituisce l'ambizioso tentativo di utilizzare le tecnologie computazionali per analizzare e descrivere, in maniera integrata, sistemi biologici complessi al fine di formulare ipotesi sui processi molecolari della vita.
L'espansione di questa disciplina medico-tecnologica ha portato alla conseguente crescita di informazioni derivanti dalle sperimentazioni bio-medico-molecolari; la disponibilità di informazioni genomiche apre nuovi orizzonti e nuove opportunità per le strategie di ricerca, ma per rendere vantaggiosa tale quantità di informazioni è fondamentale disporre di strumenti che ne permettano l'analisi e l'interpretazione.
Lo scopo della bioinformatica è di organizzare in banche dati e analizzare le conoscenze acquisite sul genoma e proteoma al fine di conservare, recuperare e valutare in modo efficace la quantità d'informazioni oggi a disposizione. Negli ultimi anni si è assistito alla nascita di numerose banche dati sul Web, che consentono la consultazione online di tali dati e permettono la possibilità di scaricarli liberamente.
Le informazioni messe a disposizione dei biologi, dei medici e ricercatori risultano molto eterogenee e distribuite e non permettono una visione d'insieme; nasce quindi la necessità di disporre di strumenti informatici per effettuare ricerche incrociate sulle varie sorgenti dati e acquisire informazioni che non si potrebbero ottenere trattando le fonti dati singolarmente.
In tal senso presso il Politecnico di Milano si sta lavorando alla realizzazione di un progetto denominato Genomic and Proteomic Data Warehouse (GPDW), che consiste nella creazione di un data warehouse che integri le informazioni distribuite dalle principali fonti di dati genomici e proteomici mondiali, sulla base di una struttura concettuale che relaziona entità biomolecolari e caratteristiche (feature) biomediche.
La presente Tesi ha pertanto come primo scopo la realizzazione di un'architettura software e delle procedure automatiche d'importazione dei dati applicabili in modo generalizzato, e come secondo obiettivo descrivere e implementare le operazioni necessarie per l'integrazione delle informazioni fornite da alcune nuove banche dati considerate nel progetto GPDW, utilizzando l'architettura software e applicando le procedure create.

Dopo il presente sommario, nel secondo capitolo della Tesi vi è un'introduzione al significato concettuale della bioinformatica, in cui sono mostrati cenni storici sulla nascita e lo sviluppo di questa disciplina e sono evidenziati i suoi compiti principali e gli obiettivi cui mira. In seguito è proposto un breve excursus degli ambiti genomici e proteomici, mettendo in risalto il contributo che le tecnologie dell'informazione portano alla ricerca bio-medico-molecolare. Inoltre è illustrata una panoramica sulle principali entità che

saranno trattate all'interno della Tesi, cromosomi, geni, proteine, DNA e sulle loro interazioni. Infine è presentata una panoramica delle banche dati biomolecolari disponibili on line e dei tipi e formati di dati che forniscono.

Nel terzo capitolo sono illustrati gli obiettivi che ci si è proposti di raggiungere con lo sviluppo della presente Tesi.

Nel quarto capitolo sono descritti gli strumenti e le metodologie tecnologiche utilizzate per il raggiungimento degli scopi enunciati, indicando i software che sono stati utilizzati per la realizzazione della Tesi. In seguito si descrive il framework che gestisce l'intero processo di creazione del data warehouse GPDW, le parti che lo compongono e le attività necessarie per l'importazione dei dati. Il capitolo si conclude con la descrizione delle attività di importazione e integrazione dei dati, mostrandone i limiti progettuali.

Nel quinto capitolo e nel sesto capitolo sono illustrate le scelte implementative che hanno portato alla realizzazione dell'architettura software e di nuove procedure automatiche per l'importazione e l'integrazione dei dati, descrivendone il funzionamento attraverso diagrammi di workflow che sintetizzano i processi da eseguire. In particolare nel quinto capitolo è descritta la progettazione dei componenti dediti all'importazione dei dati, mentre, nel sesto capitolo, è discussa l'implementazione della procedura utilizzata per la gestione delle tuple duplicate.

Nel settimo capitolo sono presentate le banche dati considerate nella Tesi, esponendo alcune informazioni di carattere generale, storico e statistico; inoltre si forniscono informazioni sui tipi di file dati forniti che sono stati qui trattati.

Nell'ottavo capitolo sono discusse le fasi della progettazione della base di dati, illustrando, per ogni banca dati trattata, i corrispondenti diagrammi entità relazione e gli schemi logici realizzati, nel quale sono descritte le tabelle create dalle procedure d'importazione implementate.

Nel nono capitolo sono descritte l'architettura software e le metodologie implementate per l'importazione automatica dei dati dai file considerati, illustrando i contenuti di tali file, le scelte progettuali e le strategie adottate per realizzare un'importazione corretta e coerente.

Nel decimo capitolo si evidenziano gli errori e le inconsistenze riscontrate nei dati importati grazie alle procedure di controllo implementate e sono esposti alcuni risultati quantitativi relativi ai dati importati e ai tempi impiegati per importarli.

La sintesi degli obiettivi raggiunti è inserita nell'undicesimo capitolo, mentre il dodicesimo capitolo propone alcuni suggerimenti riguardanti i possibili scenari futuri di utilizzo ed estensioni implementabili.

La Tesi termina con il tredicesimo capitolo contenente la bibliografia referenziata e il quattordicesimo capitolo nel quale è descritto il mapping tra i campi dei file dati considerati e i campi delle tabelle della base dati creata.

2 Introduzione

La bioinformatica[1] è una disciplina scientifica dedicata alla risoluzione di problemi biologici a livello molecolare con metodi informatici.
Essa costituisce un tentativo di descrivere dal punto di vista numerico e statistico i fenomeni biologici fornendo ai risultati tipici della biochimica e della biologia molecolare un corredo di strumenti analitici e numerici. Vengono coinvolte, oltre all'informatica, la matematica applicata, la statistica, la chimica, la biochimica e nozioni di intelligenza artificiale.
Gli aspetti fondamentali di cui si occupa la bioinformatica sono principalmente:

- fornire modelli statistici validi per l'interpretazione dei dati provenienti da esperimenti di biologia molecolare e biochimica al fine di identificare tendenze e leggi numeriche;
- generare nuovi modelli e strumenti matematici per l'analisi di sequenze di DNA, RNA e proteine al fine di creare un corpus di conoscenze relative alla frequenza di sequenze rilevanti, la loro evoluzione ed eventuale funzione;
- organizzare le conoscenze acquisite a livello globale su genoma e proteoma in basi di dati, al fine di rendere tali informazioni accessibili a tutti e ottimizzare gli algoritmi di ricerca dei dati stessi per migliorarne l'accessibilità.

L'evoluzione storica della bioinformatica, che inizialmente si occupava principalmente dello studio del DNA e del RNA, ha portato a un vasto uso dell'informatica in molti settori della biologia a tal punto che è stato coniato il nuovo termine, ormai universalmente accettato, di Biologia Computazionale che esplicita con maggior chiarezza e precisione i reali e più vasti contenuti scientifici e disciplinari del connubio tra informatica e biologia nel XXI secolo.
Nel corso degli ultimi decenni i progressi importanti nel campo della biologia molecolare accoppiati con l'evoluzione delle tecnologie genomiche, hanno portato a una crescita esponenziale delle informazioni biologiche generate dalla comunità scientifica. Questo enorme quantitativo d'informazioni genomiche ha reso necessario adottare database computerizzati per archiviare, organizzare, indicizzare i dati e ha richiesto lo sviluppo di strumenti specializzati per consultare e analizzare i dati registrati.
Una banca dati biologica è un grande "corpo" organizzato di dati persistenti, di solito associati a un software progettato per aggiornare, interrogare e recuperare i dati memorizzati all'interno del sistema. Un semplice database potrebbe essere costituito da un unico file contenente molti record, ognuno dei quali include lo stesso set di informazioni. Ad esempio, un record associato a un database di sequenze dei nucleotidi contiene in genere informazioni quali il nome del contatto, la sequenza di ingresso con una descrizione del tipo di molecola, il nome scientifico dell'organismo sorgente da cui è stato isolato, e spesso, citazioni di letteratura associati alla sequenza.

In questo senso entra in gioco la bioinformatica, quella disciplina della scienza, in cui biologia e la scienza dei computer e delle tecnologie dell'informazione, l'informatica, si fondono per formare una sola disciplina. L'obiettivo finale della bioinformatica è scoprire la ricchezza d'informazioni biologiche nascoste nella massa di dati e ottenere una visione più chiara sulla biologia degli organismi fondamentali. Questa nuova conoscenza potrebbe avere un impatto profondo in svariati campi quali la salute umana, l'agricoltura, l'ambiente, l'energia e la biotecnologia.

All'inizio della "rivoluzione genomica", un primo compito della bioinformatica è stato la creazione e il mantenimento di database per memorizzare le informazioni biologiche, come ad esempio le sequenze di nucleotidi e aminoacidi. Lo sviluppo di questo tipo di database ha coinvolto non solo le questioni di progettazione delle basi dati, ma lo sviluppo di interfacce complesse con cui i ricercatori possano accedere ai dati esistenti, nonché proporre dati nuovi o rivedere quelli presenti.

In definitiva, tutte queste informazioni devono essere combinate per formare un quadro completo delle normali attività cellulari in modo che i ricercatori possano studiare come queste attività siano alterate in differenti stati di malattia. Pertanto, il campo della bioinformatica si è evoluto in modo tale che il compito più urgente coinvolge l'analisi e l'interpretazione dei vari tipi di dati, inclusi i nucleotidi, le sequenze di amminoacidi, i domini proteici e la struttura delle proteine.

L'effettivo processo di analisi e interpretazione dei dati si riferisce alla biologia computazionale.

Importanti sotto-discipline della biologia computazionale includono:

- lo sviluppo e l'attuazione di strumenti che consentono un accesso efficiente per la gestione dei vari tipi di informazioni;
- lo sviluppo di nuovi algoritmi (formule matematiche) e le statistiche con cui valutare i rapporti tra i membri di grandi insiemi di dati, i metodi per individuare un gene all'interno di una sequenza, prevedere la struttura di proteine e la loro funzione.

La prima sfida che la comunità bioinformatica deve affrontare è l'archiviazione intelligente ed efficace di questa massa di dati, e quindi la responsabilità di fornire un accesso facile e affidabile a questi dati. Il dato in sé è privo di senso prima dell'analisi e rende impossibile, anche per un biologo addestrato, l'interpretazione manuale. Pertanto, devono essere sviluppati strumenti informatici chiari per consentire l'estrazione d'informazioni biologiche significative.

Vi sono tre processi biologici intorno ai quali gli strumenti bioinformatici devono essere sviluppati:

- la sequenza del DNA determina la sequenza della proteina;
- la sequenza della proteina determina la struttura delle proteina;
- la struttura della proteina determina la funzione delle proteina.

L'integrazione delle informazioni apprese su questi processi biologici fondamentali dovrebbe permettere di raggiungere l'obiettivo a lungo termine della completa comprensione della biologia degli organismi.

2.1 Ricerca genomica e proteomica

La genomica è una branca della biologia molecolare che si occupa dello studio del genoma degli organismi viventi. In particolare concentra il suo studio su struttura, contenuto, funzione ed evoluzione del genoma. È una scienza che si basa sulla bioinformatica per l'elaborazione e la visualizzazione dell'enorme quantità di dati che produce.
Il passo successivo rispetto alla genomica ha portato all'identificazione di una nuova disciplina che prende il nome di proteomica. La proteomica è una disciplina scientifica che studia il proteoma, essa mira a identificare le proteine e ad associarle uno stato fisiologico in base all'alterazione del livello di espressione fra controllo e trattato[2]. Permette di correlare il livello di proteine prodotte da una cellula o tessuto e l'inizio o la progressione di uno stato di stress.
Il termine proteoma è stato coniato nel 1994 da Mark Wilkins, esso descrive l'insieme delle proteine di un organismo o di un sistema biologico, in altre parole le proteine prodotte dal genoma. Rappresenta l'insieme di tutti i possibili prodotti proteici espressi in una cellula, incluse tutte le isoforme. Il proteoma è dinamico nel tempo, varia in risposta a fattori esterni e differisce sostanzialmente tra i diversi tipi cellulari di uno stesso organismo. La proteomica riguarda lo studio su grande scala della proteina, in particolare delle sue strutture e funzioni. Mentre il genoma è un'entità pressoché costante, il proteoma differisce da cellula a cellula ed è in costante evoluzione nelle sue continue interazioni con il genoma e l'ambiente. Un organismo ha espressioni proteiche radicalmente diverse secondo le varie parti del suo corpo, nelle molteplici fasi del suo ciclo di vita e nelle varie condizioni ambientali.
L'area di ricerca genomica e proteomica riguarda attività basate sull'indagine molecolare dei trascritti e delle proteine espresse in un comparto cellulare. Essa è basata sulla separazione di centinaia di prodotti proteici, sulla quantizzazione di prodotti d'espressione specifici, e sulla ricerca dei meccanismi alla base di processi biologici includendo sia ricerche a carattere metodologico che tecnologico. Quest'area è altamente multidisciplinare e richiede l'integrazione di conoscenze biochimiche, bioanalitiche, bioinformatiche e biomolecolari. Le ricerche riguardano la definizione di prodotti d'espressione candidati e le proteine espresse in specifici comparti cellulari avvalendosi di modelli umani e animali. Tale studio globale è finalizzato alla comprensione dei meccanismi biologici in particolari condizioni fisiologiche e patologiche a carico degli organi in esame. L'obiettivo è l'identificazione dei targets molecolari coinvolti e la comprensione dei meccanismi sottesi all'adattamento o alla progressione della malattia. Lo studio proteomico richiede il continuo sviluppo di metodi per il miglioramento delle capacità separative, della sensibilità e delle possibilità

d'interpretazione dei dati correlati ai segnali biologici. I risultati delle ricerche sono: nuovi targets e nuove metodologie.
Alla base della genomica sono i metodi della biologia molecolare, quali i metodi di clonaggio dei geni e di sequenziamento del DNA. Conoscere l'intero genoma degli organismi permette di assumere nei confronti della ricerca biologica un approccio nuovo in silico, vale a dire consentire la riproduzione in una simulazione matematica al computer per indicare fenomeni di natura chimico-biologica, invece che in provetta o in un essere vivente.
Questo presenta molti vantaggi: un esempio è la maggior facilità del trasferimento delle conoscenze su un organismo ad un altro, tramite la ricerca di geni omologhi. Un altro vantaggio è chiaramente osservabile per esempio in campo biomedico: molte malattie sono complesse, determinate da molti geni, come i tumori, e la conoscenza dell'intero genoma permette di identificare con più facilità i geni coinvolti e osservare come questi interagiscono nel loro background genetico. Grazie alle scoperte scaturite dal sequenziamento del genoma umano è nata una nuova branca definita genetica personalizzata, derivante dalle applicazioni delle conoscenze genetiche in medicina e nella pratica clinica. Ora è, infatti, possibile eseguire degli studi predittivi sull'incidenza di una data patologia su un campione o su un individuo rispetto alla popolazione generale per definire il rischio di sviluppare quella patologia.
Tra gli obiettivi che si pone la genomica vi è dunque l'allestimento di complete mappe genetiche e fisiche del DNA degli organismi viventi, proseguendo con il suo completo sequenziamento. La sequenza del DNA viene poi annotata, ovvero vengono identificati e segnalati tutti i geni e le altre porzioni di sequenza significative, insieme a tutte le informazioni conosciute su tali geni.
In questo contesto, l'informatica si propone di fornire strumenti e metodi di analisi, indispensabili per orientarsi in un ambito che attualmente contempla enormi quantità di dati sperimentali e informazioni. Per guidare le condizioni in cui gli esperimenti sono stati effettuati e per aiutare lo scienziato a valutarne i risultati sono stati creati strumenti efficaci ed efficienti in grado di organizzare la conoscenza biologica. Tali strumenti sono i vocabolari controllati e le annotazioni.

2.2Vocabolari controllati, ontologie e annotazioni funzionali

Nel proseguimento della corrente Tesi si tenderà a limitare i termini quali gene o proteina, in quando gli argomenti trattati varranno per entrambi gli oggetti; sarà quindi preferita l'espressione entità biomolecolare per includere entrambi i termini in un unico soggetto.
Un vocabolario controllato fornisce una metodologia per organizzare la conoscenza in maniera ordinata, facilmente leggibile e preferibilmente in modo univoco affinché sia utilizzabile semplicemente e rapidamente da un calcolatore elettronico. Un vocabolario controllato è composto di un insieme di termini preselezionati e autorizzati dal progettista dello stesso; ciò è in contrasto con il linguaggio naturale in quanto in quest'ultimo non vi

sono restrizioni sul vocabolario utilizzato e quindi è resa possibile la presenza di omografie, sinonimi e polisemie. Tali fattori, che aumenterebbero di fatto le ambiguità delle informazioni, sono assenti, o quasi, nei vocabolari controllati.
Ogni termine presente in un vocabolario controllato, identificato univocamente da un codice alfanumerico, rappresenta un particolare concetto o caratteristica.
Un esempio di vocabolario controllato è Medical Subject Heading (MeSH)[3] il cui scopo è indicizzare articoli di riviste e libri per favorire la categorizzazione e la ricerca di documentazione scientifica in ambito medico.
In ambito di annotazione genomica, vi sono numerosi vocabolari controllati che possono essere suddivisi in due categorie:

- vocabolari controllati flat o terminologie: i termini presenti nel vocabolario non sono strutturati o collegati tramite relazioni;
- vocabolari controllati strutturati o ontologie: i termini presenti nel vocabolario sono strutturati gerarchicamente, dal più generico (radice) ai più specifici (foglie), tramite relazioni caratterizzate da un particolare significato semantico.

Per entità biomolecolare s'intenderà quindi qualsiasi elemento che possa essere descritto attraverso termini specifici appartenenti a un vocabolario controllato.
Mentre negli anni passati l'utilizzo di ontologie era raro, attualmente gli strumenti che operano con queste sono aumentati esponenzialmente, perciò la loro struttura è di fondamentale importanza. Tanto è vero quanto il fatto che molti vocabolari originariamente flat si stanno trasformando anch'esse in ontologie. Ad ogni modo, tra i vocabolari controllati flat è possibile citare il vocabolario di OMIM (Online Mendelian Inheritance in Man)[4] per descrivere malattie genetiche oppure il vocabolario di Reactome[5] per la descrizione di pathway.
Tra i vocabolari controllati ontologici è possibile citare ChEBI (Chemical Entities of Biological Interest) concentrato su componenti chimici, System Biology Ontology (SBO) concentrata sulla modellazione computazionale in ambito biologico, Medical Subject Heading (MeSH) concentrata sull'indicizzazione di articoli scientifici oppure ancora Gene Ontology (GO)[6]. Quest'ultima è probabilmente la più famosa e utilizzata ontologia per annotare nuove entità biomolecolari[7], corredata da un vasto insieme di strumenti che su di essa si basano per effettuare applicazioni di text mining[8] o ancora per applicazioni cliniche[9].
I vocabolari presi singolarmente hanno poco significato intrinseco e le ontologie, fino a pochi anni fa, potevano essere considerate degli esercizi di stile. Negli ultimi anni si è assistito a un'inversione di mentalità. Sempre più organizzazioni si concentrano sui vocabolari controllati, i quali termini sono utilizzati per descrivere le conoscenze riguardanti le entità biomolecolari. Tale descrizione è detta annotazione. Viene quindi definita annotazione un'associazione di uno specifico termine di un vocabolario controllato ad una particolare entità biomolecolare; tale entità sarà quindi descritta dal significato attribuito al termine associato.

2.3 Banche dati biomolecolari

A partire dalla nascita dei primi calcolatori e soprattutto negli ultimi quindici anni l'informatica ha conosciuto uno sviluppo esponenziale, in accordo su quanto afferma la legge di Moore: "Le prestazioni dei processori, e il numero di transistor ad esso relativo, raddoppiano ogni 18 mesi.", come si nota dal grafico in figura 1.

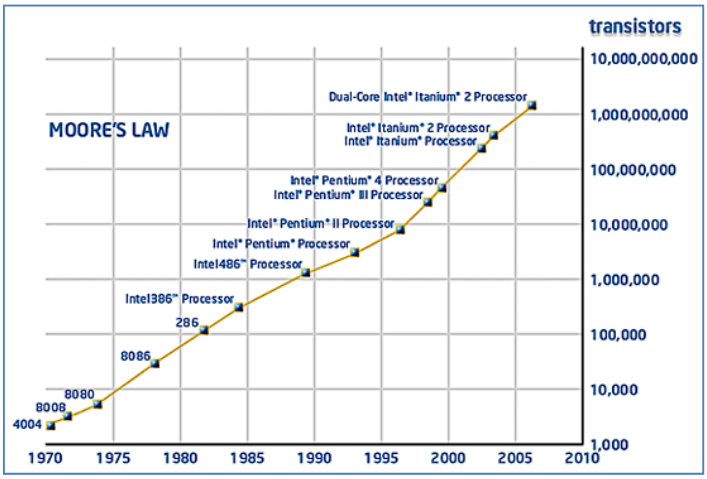

Figura 1 – Legge di Moore

Parallelamente e conseguentemente a questo sviluppo anche Internet ha subito un'evoluzione in maniera ugualmente repentina (figura 2). La grande crescita di Internet ha contribuito allo sviluppo della bioinformatica.

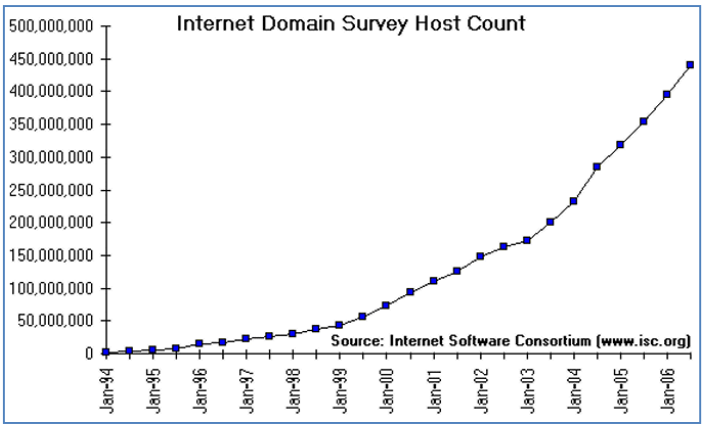

Figura 2 – Diffusione degli Internet servey negli ultimi anni

Prima dell'avvento del PC le informazioni venivano memorizzate su supporti fisici quali la carta, la memorizzazione dei dati era sequenziale e non permetteva un ordinamento specifico e nemmeno delle ricerche mirate, oltre a considerare il fatto che qualsiasi operazione presupponeva un dispendio di risorse umane, economiche e temporali consistenti. Con l'avvento dell'informatica si sono iniziati a sviluppare i database; i termini database, banca dati, base dati, indicano tutti un archivio strutturato in modo tale da

consentire la gestione dei dati, da parte di applicazioni software. Il database è un insieme d'informazioni, di dati che vengono suddivisi per argomenti in un ordine logico(tabelle), in seguito tali argomenti vengono suddivisi per categorie (campi)[10][11] .
Esistono essenzialmente due modi differenti per la gestione dei dati in un database:

- database flat file: tutti i dati sono memorizzati in un unico file. Il file può essere strutturato in due tipologie differenti:
  - formato testuale: un database può essere memorizzato semplicemente in un file di testo. Tutti i record sono scritti in modo seriale, separati tra loro da particolari spaziatori, con i relativi dati (campi) scritti al loro interno;
  - formato tabella: ogni riga rappresenta un record e ogni colonna rappresenta un campo. I nomi delle colonne rappresentano i nomi dei campi dei record.
- database relazionale: i dati sono memorizzati in più tabelle collegate tra loro, inoltre i database relazionali necessitano di particolari programmi di gestione (Database Management System o DBMS) , che siano in grado di saltare da una tabella all'altra e di interpretare le relazioni ed i vincoli ad esse associati. Devono inoltre occuparsi di gestire l'aggiunta, la modifica e la gestione degli indici.

L'utilizzo di queste strutture da parte dei biologi ha portato alla nascita delle banche dati biologiche.
Le banche dati biologiche[12] sono archivi di dati coerenti, memorizzati in modo uniforme ed efficiente. Questi database contengono dati provenienti da un ampio spettro di settori della biologia molecolare e si suddividono in due gruppi:

- Basi di dati primarie o archiviate che contengono informazioni e annotazioni di sequenze di DNA e proteine, le strutture delle proteine, del DNA e dei profili di espressione proteica. Alcuni esempi sono:
  - GenBank - http://www.ncbi.nlm.nih.gov/Genbank/
  - EMBL - http://www.ebi.ac.uk/embl/
  - DDBJ - http://www.ddbj.nig.ac.jp/
- Basi di dati secondarie o basi di dati derivate, sono così chiamate perché contengono i risultati delle analisi sulle risorse primarie, comprese le informazioni sui modelli di sequenza, le varianti, le mutazioni e le relazioni evolutive. Informazioni provenienti dalla letteratura, contenute nelle banche dati bibliografiche. Tra questi i più noti sono:
  - OMIM - http://www.ncbi.nlm.nih.gov/sites/entrez?db=omim
  - PUBMED - http://www.ncbi.nlm.nih.gov/PubMed/

È essenziale che queste banche dati siano facilmente accessibili e che sia fornito un intuitivo sistema d'interrogazioni per consentire ai ricercatori di ottenere informazioni specifiche su un particolare argomento biologico.
Sono state create banche dati specializzate per particolari soggetti ad esempio:

- Banche dati di letteratura scientifica;

- Banche dati di tassonomia;
- Banche dati di nucleotidi;
- Banche dati genomiche;
- Banche dati di proteine;
- Banche dati di microarray.

Come mostra la figura 3, le banche dati hanno avuto un largo incremento negli ultimi anni, a oggi se ne possono contare 1330.

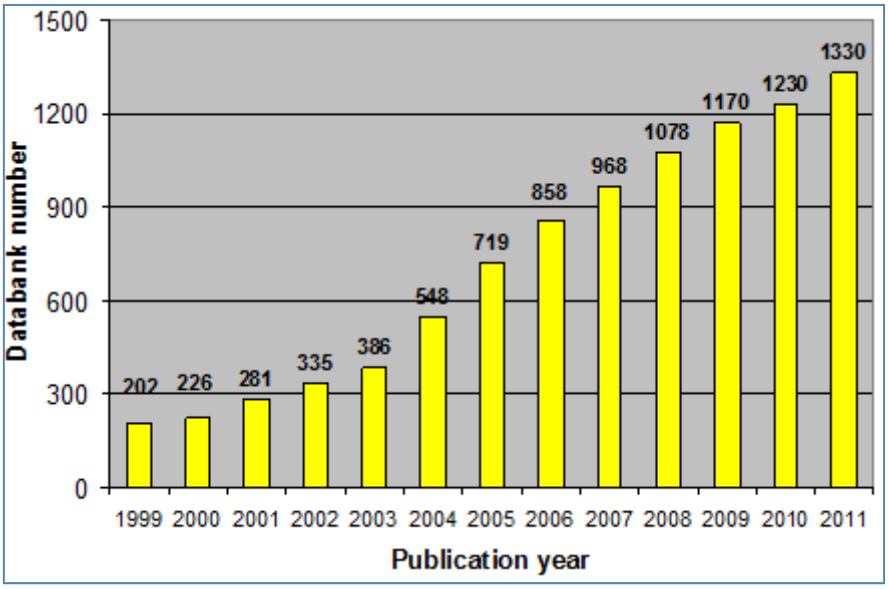

Figura 3 – Rilevazione statistica delle banche dati mondiali

Le diverse banche dati presenti online forniscono differenti modi per l'accesso ai dati:

- Accesso attraverso l'interfaccia Web (pagine HTML o XML): le informazioni fornite risultano non strutturate, i dati vengono proposti tramite interfacce eterogenee, inoltre i risultati delle query sono proposti per singola sequenza e sono principalmente restituiti in formato HTML, richiedendo molto tempo per avere risposte esaurienti.
- Accesso tramite Web service: questo servizio è disponibile solo per poche banche dati con un numero limitato di elementi ed è rivolto a persone che possiedono una certa abilità informatica.
- L'accesso tramite server FTP: richiede di avere le tecnologie necessarie e le risorse umane per la re-implementazione della banca dati a livello locale.
- Accesso diretto: questo viene raramente concesso per questioni di sicurezza e propone linguaggi di interrogazione differenti tra le varie banche dati, in più si evidenzia la mancanza di un vocabolario comune.

In figura 4 sono presentate le maggiori banche dati presenti nel WEB.

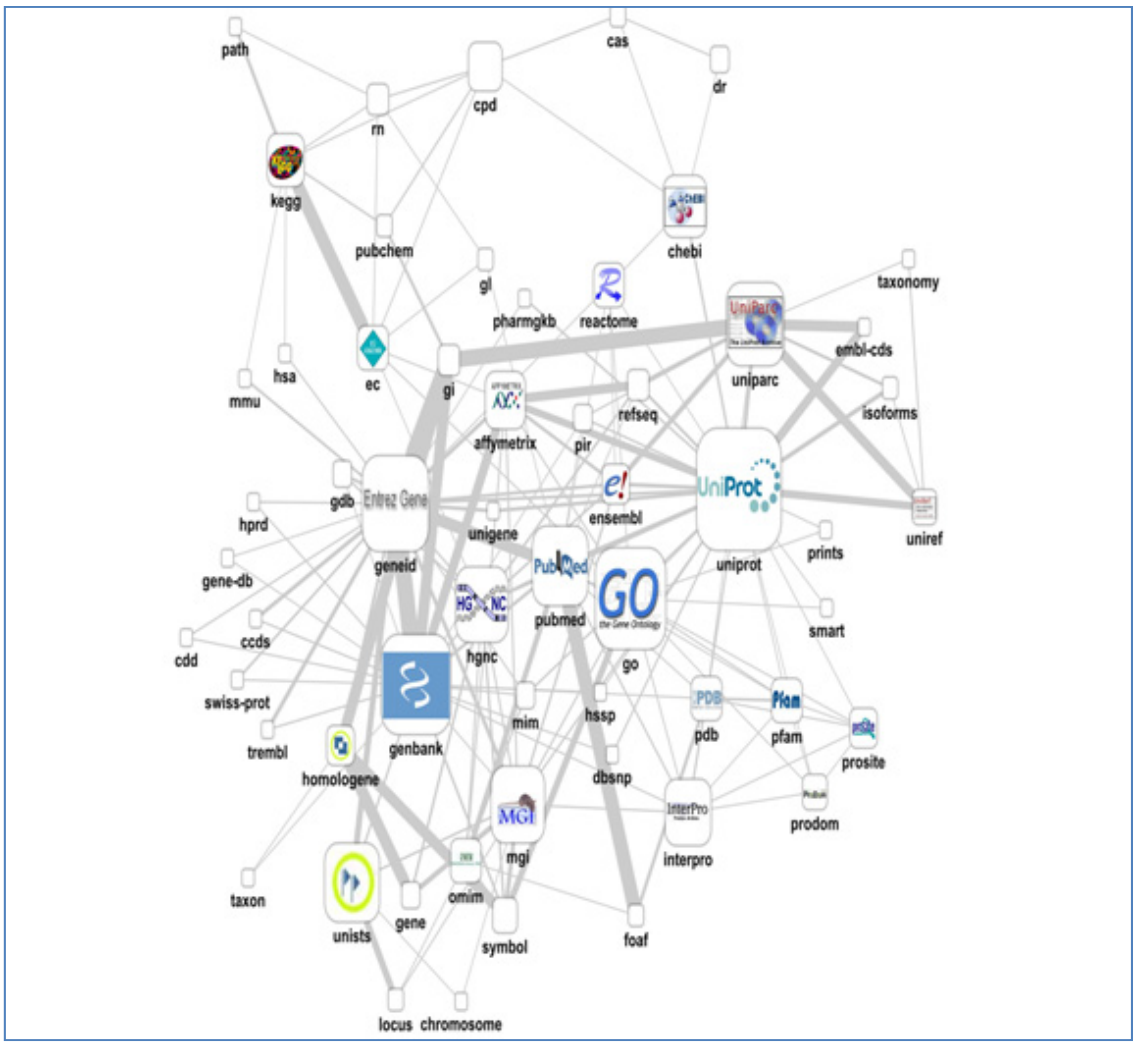

Figura 4 – Banche dati mondiali

Ogni banca dati risulta spesso poco integrata con le altre sorgenti, il che rende necessario un lavoro di integrazione di dati come quello prodotto dal progetto GPDW[13] oggetto di questa Tesi.

2.4 Formati di file dati

Le banche dati possono fornire gli stessi dati in formati diversi, non esiste uno standard comune per il tipo di file e il formato con cui rappresentare le informazioni[14]; i dati genomici sono generalmente forniti con tipologie di file diverse, come mostra la figura 5.

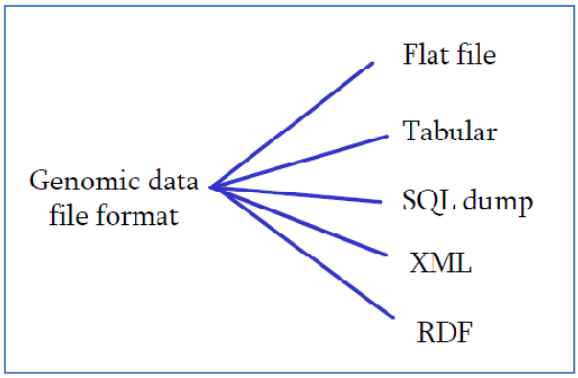

Figura 5 –Tipi di formato dati dei file genomici

Il tipo flat file è definito come un file di testo contenente valori strutturati tra di loro, nel campo dei dati genomici un flat file è un file di testo in cui ogni riga ha una semantica diversa e la semantica è definita dall'etichetta inserita all'inizio della riga. E' possibile che in una stessa riga ci possano essere due o più etichette successive in cui l'etichetta che segue specifica la semantica di quella precedente; se all'inizio di una riga non è indicata alcuna etichetta, la riga eredita la semantica della precedente riga.
In generale, si può affermare che un file testuale è in formato tabellare quando i dati contenuti sono organizzati in righe e colonne separate da uno o più separatori In questo caso il valore semantico di un dato dipende dalla sua posizione di riga e colonna. La figura 6 mostra le diverse tipologie di file tabellari esistenti.

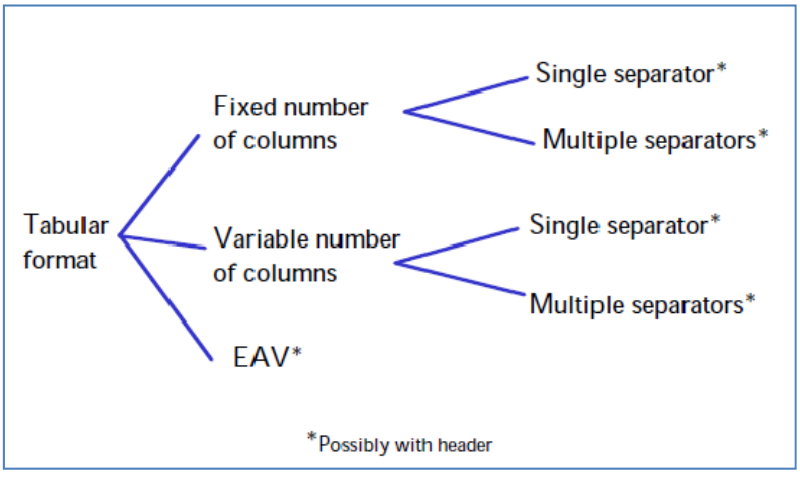

Figura 6 – Tipi di file tabellari

Spesso le intestazioni presenti in questa tipologia di file sono significative per la comprensione del contenuto del file, altre volte ne sono solo a corredo e forniscono informazioni di carattere statistico.
Alcune banche forniscono i propri dati come file dump SQL: sono DBMS specifici e spesso DBMS diversi o versioni precedenti dello stesso DBMS non riescono ad importarli correttamente.

Ad esempio la banca dati della Gene Ontology (GO) fornisce un file dump SQL per il DBMS MySQL, contenente l'ontologia e le annotazioni riguardanti geni e proteine. La banca dati InterPro, invece, fornisce sia il dump SQL per Oracle che per MySQL.
A volte si possono trovare dump SQL generici, non per un DBMS specifico, ma spesso si incontrano problemi d'importazione, generalmente dovuti alla mancanza di chiavi esterne nel dump, mancano infatti i vincoli di integrità referenziale e le relazioni tra le tabelle sono descritte solo graficamente nello schema del DB, che non sempre è fornito.
L'XML o eXtensible Markup Language, è un metalinguaggio creato e gestito dal World Wide Web Consortium (W3C) utilizzato per creare nuovi linguaggi atti a descrivere documenti strutturati e come mezzo per l'esportazione di dati tra sorgenti eterogenee. Per tali caratteristiche molte banche dati mettono a disposizione i propri dati in formato XML. La struttura dell'istanza di un documento XML può essere descritta mediante Document Type Definition (DTD) o XML-Schema.
Alcune banche dati genomiche forniscono oltre al formato XML generico un altro formato XML standard per il Web: RDF.
Il Resource Description Framework (RDF) è lo strumento base proposto da W3C per la codifica, lo scambio e il riutilizzo di metadati strutturati e consente l'interoperabilità tra applicazioni che si scambiano informazioni sul Web.
È costituito da due componenti:

- RDF Model and Syntax che espone la struttura del modello RDF e descrive una possibile sintassi;
- RDF Schema che espone la sintassi per definire schemi e vocabolari per i metadati.

2.5 Difficoltà nell'efficace utilizzo delle informazioni biomolecolari disponibili

Riprendendo il discorso sviluppato nel sottocapitolo dedicato alle banche biomolecolari, si possono riassumere alcune delle principali problematiche che nascono dall'utilizzo di tali conoscenze.
In primo luogo vi è una molteplicità di banche dati distribuite geograficamente che possono includere anche dati duplicati. Il lavoro eseguito negli ultimi anni riguarda soprattutto l'esecuzione di un'integrazione di tali dati in un'unica struttura centrale che sia in grado di gestire e mantenere aggiornato il contenuto stesso. In quest'ottica è stato sviluppato GPDW e altri sistemi d'integrazione di dati. In relazione a tale punto vi è, in primo luogo, il problema di gestire la grossa quantità eterogenea di dati in modo integrato e riuscire a offrire all'utente del sistema una vista il più possibile omogenea.
In secondo luogo esiste la problematica riguardante i vocabolari controllati; nonostante dichiarino di essere il più possibile ortogonali tra loro, vi sono relazioni tra termini di un vocabolario controllato e un altro, soprattutto se tali vocabolari sono gestiti da organizzazioni diverse tra loro.

Infine il problema riguardante la bontà delle annotazioni presenti nelle banche dati[15], la difficoltà affrontata dai curatori di validare nuove annotazioni e la difficoltà nella gestione e mantenimento di tali annotazioni. A titolo indicativo il GO Consortium, dal completamento del sequenziamento[16] terminato nel 2003 da Celera-Genomics e da Homo Genome Project (HGP) a oggi, ha annotato solamente 19920 geni sui circa 25000 presenti nell'essere umano, evidenziando che circa il 20% non risulta ancora caratterizzato; inoltre delle 159423 annotazioni presenti, circa il 58% risultano inferite elettronicamente ovvero non confermate da un curatore e pertanto poco attendibili.
Questa Tesi cerca di dare una risposta, in modo efficiente ed efficace, ad alcuni di questi problemi tramite il data warehouse sviluppato presso i laboratori del Politecnico di Milano.

3 Obiettivi della Tesi

L'integrazione dei dati biomolecolari è un importante aspetto della ricerca bioinformatica, sia per le sfide che impone di superare, sia per il ruolo che ricopre nella ricerca scientifica nell'ambito delle scienze della vita. Tramite la ricerca biomolecolare in particolare, si può rispondere alle varie domande d'interesse analizzando complessivamente i vari tipi di dati e le conoscenze disponibili, in modo da ottenere evidenze a supporto dei risultati ottenuti. La grande quantità di dati biomolecolari presenti in forma strutturata o semi-strutturata rende imprescindibile la loro integrazione e analisi automatizzata. Le descrizioni controllate (annotazioni) delle entità biomolecolari (geni e loro prodotti proteici) sono di fondamentale importanza per gli scienziati e i ricercatori perché questi dati possono sostenere in maniera efficace l'interpretazione biomedica dei risultati di screening biomolecolari. Tali risultati concorrono alla creazione della conoscenza che può essere utilizzata per formulare e validare ipotesi, ed eventualmente scoprire nuova conoscenza biomedica. Integrare le informazioni fornite da molteplici fonti è quindi fondamentale per la ricerca biomolecolare. Tuttavia la dispersione dei dati genomici e proteomici in molte risorse distribuite, l'alta varietà e la grande quantità di dati rappresentano una sfida importante da affrontare per l'integrazione e l'utilizzo efficiente ed efficace dei dati disponibili. A questa sfida si rivolge il progetto GPDW (Genomic and Proteomic Data Warehouse), di cui fa parte questa Tesi. Il progetto GPDW si propone di creare un data warehouse locale che integri le annotazioni provenienti da diverse banche dati, mantenendovi aggiornate le informazioni li integrate.
Questa Tesi si prefigge come scopo principale la generalizzazione delle procedure presenti nel framework GPDW, ristrutturando in modo congiunto i file di configurazione XML e le procedure automatiche per l'importazione e l'integrazione dei dati, e, come secondo obiettivo, utilizzare tali modifiche per importare e integrare le informazioni messe a disposizione da alcune banche dati non ancora considerate nel progetto GPDW. In particolare, la realizzazione di questo secondo obiettivo, prevede la descrizione e l'implementazione delle operazioni necessarie per l'importazione e l'integrazione dei dati forniti dalle nuove sorgenti considerate. Tale obiettivo si suddivide nei seguenti sotto-obbiettivi metodologici:

- modellizzazione concettuale e logica dei dati forniti dalle banche dati considerate;
- implementazione del processo d'importazione dati attraverso la creazione di procedure automatizzate di parsing di dati standard e specifiche, secondo il tipo di file dati considerato, e d'importazione nel data warehouse dei dati estratti dai file dati;
- configurazione delle procedure automatiche per l'importazione e l'integrazione dei dati nel data warehouse e segnalazione di anomalie presenti nei dati.

4 Strumenti e metodologie tecnologiche

Il progetto Genomic and Proteomic Data Warehouse (GPDW) è stato sviluppato per la creazione di una struttura dati che consenta l'integrazione delle informazioni genomiche e proteomiche disponibili e scaricabili via Internet, e permetta l'evoluzione nel tempo a fronte dell'integrazione di nuove fonti dati. Inoltre, grazie alla realizzazione di un sistema automatico per l'aggiornamento dei dati in esso contenuti, sarà possibile verificare eventuali problemi e inconsistenze dovuti all'inserimento di nuovi dati. Questa è un'attività molto importante perché a causa della continua evoluzione dei dati e del formato in cui essi vengono forniti, si potrà monitorare costantemente il loro andamento. L'approccio utilizzato per l'integrazione di una nuova fonte dati all'interno del data warehouse prevede una serie di attività schematizzate in figura 7.

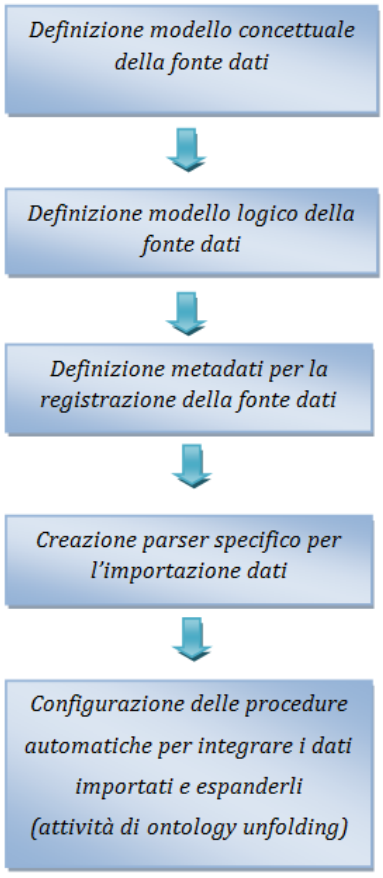

Figura 7 – Attività d'importazione

Nel seguito della Tesi sarà usato il termine feature per indicare indistintamente entità biomelocolari e feature biomediche, poiché dal punto di vista informatico rappresentano lo stesso oggetto.

4.1 Software utilizzati

Lo sviluppo di GPDW è stato implementato in ambiente Java, la piattaforma utilizzata è Eclipse, di tipo open-source composta da una struttura estendibile e da strumenti e runtime per la creazione, la distribuzione e la gestione del software in tutto il suo ciclo di vita[17][18].
Per la struttura dati è stato utilizzato PostgreSQL, un database relazionale ad oggetti che utilizza il linguaggio SQL, con più di 15 anni di sviluppo attivo e di architettura testata che gli ha permesso di conseguire un'ottima reputazione per l'affidabilità, l'integrità dei dati e la precisione. È compatibile con tutti i maggiori sistemi operativi e completamente conforme al modello ACID dei database.
La progettazione delle strutture del database e dei diagrammi logici e concettuali è stata implementata tramite l'utilizzo di Microsoft Visio ®.

4.2 Il framework GPDW

Il framework GPDW gestisce tutto il processo d'integrazione e importazione, partendo dalla generazione della base dati di supporto contenente i metadati necessari al funzionamento dell'applicazione stessa.
Il metodo per l'integrazione dei dati eterogenei raccolti dal Web è diviso a due macrofasi:

- importazione dei dati dalle loro differenti fonti in un primo livello, definito livello dati importati;
- integrazione dei dati importati.

Per eseguire automaticamente queste operazioni, è stata realizzata l'architettura di software, le cui componenti principali sono identificate nella figura 8.

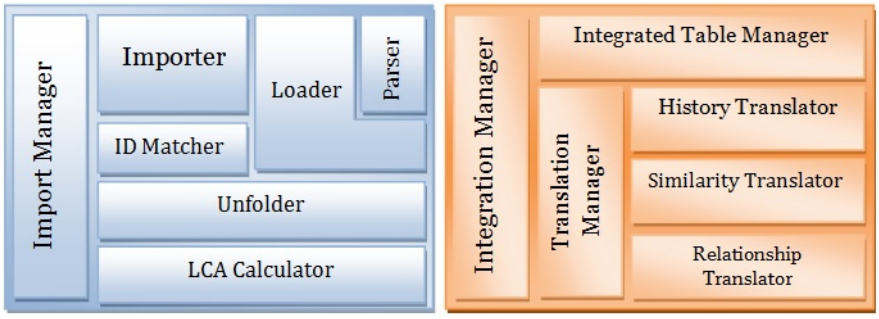

Figura 8 – Struttura del framework GPDW

Il run dell'applicazione genera una base di dati composta da quattro schemi:

1. *public*: in questo schema sono create le tabelle nelle quali vengono memorizzati i dati direttamente importati dai file di sorgente (tabelle di livello importato) e le tabelle di livello integrato ad esse associate, generate durante la fase d'integrazione dei dati;

2. *flag*: contiene le tabelle in cui sono memorizzati i valori dei campi codificati nelle tabelle dello schema *public*;
3. *metadata*: vi sono memorizzate le tabelle contenenti informazioni relative le sorgenti importate, i file analizzati e le features cui si riferiscono i dati importati;
4. *log*: in questo schema sono memorizzate le tabelle di appoggio create dalle procedure automatiche nel caso di operazioni particolarmente complesse; sono, inoltre, presenti le tabelle contenenti tuple eliminate da tabelle dello schema *public* (ad esempio tuple duplicate o aventi valori inconsistenti), in modo da tenere traccia di tutti i dati importati e rilevare eventuali anomalie.

Le procedure d'integrazione e importazione dei dati descritte nel seguito sono state definite prima del mio lavoro di Tesi. Dato che presentavano alcune limitazioni, ho proceduto alla loro ridefinizione, come illustrato nei capitoli successivi.

4.3 Metodologia d'importazione dei dati

Le operazioni d'importazione dati richiedono l'esecuzione dei seguenti processi standard e in particolare richiedono la registrazione della fonte dati nel file di definizione *data_source.xml*.
I componenti principali interessati dalle operazioni d'importazione sono quattro:

- ImportManager;
- Importer;
- Parser;
- Loader.

L'ImportManager è unico per tutta la fase d'importazione, il suo obiettivo è istanziare, configurare ed eseguire gli Importer dichiarati nel file di configurazione *data_sources.xml*. Un altro compito dell'ImportManager è creare gli indici sul database alla fine della fase d'importazione.
Per ciascuna fonte dati considerata viene definito un Importer, il cui obiettivo è quello di gestire ed eseguire i Parser e i Loader per tale origine dati.
L'Importer riceve dall'ImportManager l'elenco dei file dati da importare, quindi configura, istanzia e gestisce i Loader. Per ogni file sorgente viene istanziato un Loader che estende il Parser adeguato per ciascun formato di file specifico. Il Loader riceve dal Parser i dati estratti dal file e li memorizza nelle tabelle del database.
I Parser sono classi che si occupano di estrarre i dati dal file; sono presenti vari Parser specifici per ogni tipo di formato di file:

- TabularFileParser: utilizzato per gestire i file di tipo tabellare con separatore di campo singolo;
- TabularFileWithHeaderParser: utilizzato per gestire i file tabellare con separatore di campo singolo aventi header;

- TabularFileParserMultipleSeparator: utilizzato per gestire i file di tipo tabellare con separatori di campo multipli;
- TabularFileWithHeaderParserMultipleSeparator: utilizzato per gestire i file di tipo tabellare con separatori di campo multipli aventi header.

Il Loader estende un Parser in quanto deve implementarne il metodo astratto *onNextLine()* che si occupa di processare la singola riga estratta dal file tabellare. Spesso la distinzione tra Parser e Loader non è ben definita, perché il Loader esegue alcune operazioni di analisi, o viceversa.

Se il Parser per il file dati da importare è già presente all'interno delle classi del framework, esso viene utilizzato, in caso contrario è necessario creare un nuovo Parser. È inoltre necessario implementare l'Importer specifico per ogni sorgente, mentre l'ImportManager non viene modificato.

In figura 9 sono schematizzate le operazioni necessarie a completare il processo di importazione e le interazioni tra i componenti precedentemente descritti.

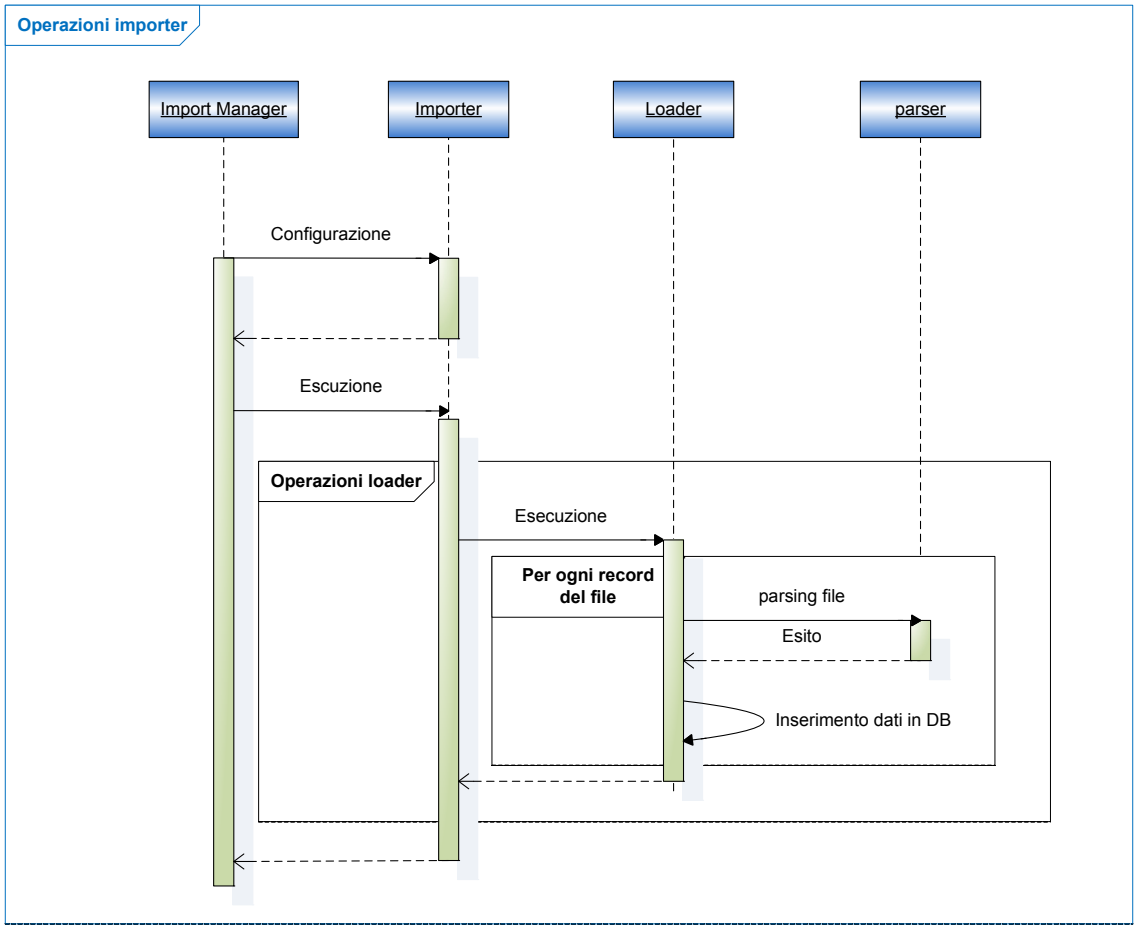

Figura 9 – Scenario di una tipica fase d'importazione

È evidente come tale architettura presenti dei limiti poiché vi sono alcuni elementi non sufficientemente generalizzati. Ciò comporta, oltre ad un'inutile duplicazione del codice, la creazione di classi specifiche che spesso non seguono uno stesso standard processuale.

Inoltre eventuali modifiche strutturali del framework richiederebbero la ridefinizione di tutte le classi implementate in precedenza. Ho, quindi, proceduto con un'analisi approfondita delle procedure presenti all'interno del progetto, con lo scopo di definire un formalismo comune che consentisse di astrarle ulteriormente. Questi aspetti saranno maggiormente approfonditi nel capitolo successivo.

L'importazione è stata progettata per essere molto flessibile e consentire l'aggiunta di nuove fonti in modo semplice e procedurale. Per raggiungere quest'obiettivo, il processo è guidato da un file di configurazione XML, *data_sources.xml* nel quale è necessario definire l'elenco e tutte le caratteristiche di alto livello delle sorgenti dati da importare e le relazioni che intercorrono tra di esse.

Inoltre, quando viene importata una nuova feature è necessario definirne le caratteristiche all'interno di un altro file XML, *feature_definition.xml*, andando a specificare se si tratta di entità biomolecolare o di feature biomedica, se è ontologica, se presenta dati di similarità o di history e con quale altre features viene associata.

Ovviamente, la riprogettazione delle procedure automatiche d'importazione, descritta in precedenza, ha richiesto in maniera congiunta una ristrutturazione dei file di configurazione XML, all'interno dei quali deve essere definito tutto quanto risulta essere specifico per la singola sorgente.

Ogni sorgente identifica i propri record attraverso un identificativo; prima di importare i dati all'interno del data warehouse, è necessario verificare che tale ID sia conforme con l'espressione regolare definita all'interno del file di configurazione *data_sources.xml*. Il componente che si occupa di eseguire il match è definito all'interno della classe *RelationshipMatcher.java*, in realtà creata per importare le informazioni di associazione tra coppie di features. È stato quindi necessario implementare una nuova classe dedita a validare gli ID importati con le espressioni regolari, mentre la classe *RelationshipMatcher.java* dovrà occuparsi solamente del popolamento delle tabelle di associazione. Quest'aspetto sarà maggiormente approfondito in seguito nel quinto capitolo.

L'applicazione assegna a ogni "data record" importato un OID univoco rispetto tutte le entries del data warehouse. Per assicurare la correttezza dei dati importati, è stato definito un insieme di regole che consentono al componente ID Matcher di identificare il tipo di dato associato a ciascun ID, in modo da inserire le informazioni nelle tabelle appropriate del data warehouse. Durante questo processo, ogni tupla viene associata con i metadata relativi la propria sorgente dati, in modo da tenere traccia della sua provenienza.

Il processo d'importazione è in grado di completare autonomamente il lavoro nonostante siano presenti nel file anomalie, quali la presenza o la mancanza di colonne rispetto a quanto definito nel file XML, di rilevare (e ricordare) le modifiche e segnalare la presenza di tali anomalie.

Al termine dell'importazione dei dati sono abilitate le chiavi primarie o esterne, i vincoli di unicità e gli indici, in modo da rilevare possibili duplicazioni e incongruenze, e migliorare i tempi di acceso.

L'applicazione prevede che sia abilitato un indice univoco sui campi *source_name* e *source_id* delle tabelle importate, in modo da identificare ogni singola tupla fornita da una sorgente. L'importazione della banca dati OMIM è risultata conflittuale con tale definizione. A differenza delle sorgenti finora importate, le quali fornivano per ogni loro identificativo un solo data record, OMIM fornisce diversi file di sorgente all'interno dei quali sono memorizzate tuple riferite allo stesso ID. È stato, quindi, necessario non solo tenere traccia della sorgente che ha fornito l'ID ma anche del file dal quale quest'ultimo è stato estratto, tramite l'aggiunta di un nuovo campo *reference_file*. Inoltre vi era la necessità di eliminare eventuali tuple duplicate e di aggregare informazioni riguardanti una stessa feature memorizzate in record diversi tra loro. Ciò ha portato alla realizzazione di un nuovo modulo dedito all'eliminazione/merge delle tuple duplicate, le cui scelte procedurali e implementative saranno descritte con maggiore dettaglio nel sesto capitolo.

4.4 Metodologia d'integrazione dei dati importati

Le principali operazioni svolte durante il processo d'integrazione dei dati importati possono essere raggruppate in due fasi: aggregazione e integrazione.
Durante la fase di aggregazione vengono create e popolate le tabelle integrate a partire dai dati importati. In seguito gli ID dei dati di history e similarità importati sono associati agli OID dei dati integrati nel data warehouse. Infine vengono traslati gli ID contenuti nelle tabelle di associazione tra coppie di features.
Durante la fase d'integrazione, attraverso un'analisi di similarità, si verifica se singole istanze fornite da sorgenti diverse rappresentano la stessa feature. In questo caso sono associate a un nuovo "concept OID".
Al termine dell'integrazione dei dati sono abilitate le chiavi primarie o esterne, i vincoli di unicità e gli indici, in modo da rilevare possibili duplicazioni e incongruenze, e migliorare i tempi di acceso.

5 Astrazione delle procedure automatiche d'importazione dei dati

Nei capitoli precedenti è stata evidenziata la necessità di ottenere una base di dati che integri in maniera consistente le annotazioni provenienti da diverse banche dati accessibili online. Argomento principale della presente Tesi è la generalizzazione delle procedure automatiche d'importazione presenti all'interno del framework GPDW, in modo che possano essere applicate a una qualsiasi sorgente presente sul Web.

La necessità di realizzare procedure il più possibile astratte è dovuta a tre fattori:

- numerosità delle banche dati;
- varietà delle banche dati;
- frequenza di aggiornamento delle banche dati.

Per ottenere dati costantemente aggiornati e consistenti la procedura d'importazione dei dati deve essere realizzata in modo da riconoscere e adattarsi a eventuali modifiche dei file di sorgente.

L'architettura del framework descritta nel quarto capitolo prevedeva la realizzazione di un Importer di sorgente specifico per ogni banca dati considerata, e di un Loader specifico per ogni file importato. Vi erano quindi numerose parti di codice duplicate e procedure non standardizzate. Si è, quindi, reso necessario procedere con un'analisi approfondita delle procedure precedentemente implementate e delle strutture dei file messi a disposizione delle banche dati genomiche e proteomiche, in modo da poterne definire gli elementi comuni da trattare in modo standardizzato e nel contempo evidenziare eventuali errori progettuali presenti nella precedente versione del framework.

Nei sottocapitoli successivi saranno descritti i risultati di tale analisi e le scelte implementative adottate.

5.1 Ristrutturazione del file di configurazione data_sources.xml

Per procedere alla realizzazione delle procedure automatiche d'importazione è fondamentale definire ogni singola banca dati importata in una struttura XML facilmente compilabile, nella quale descrivere le informazioni da essa fornite. All'interno di tale struttura deve essere inserito tutto quanto risulta essere specifico per una singola sorgente, in modo da minimizzare la necessità di implementare nuovo codice Java.

In particolare, per ogni sorgente devono essere definiti:

1. i file da analizzare raggruppati per tipologie omogenee;
2. le features cui si riferiscono i dati importati, la struttura della relativa tabella di sorgente importata e le tipologie di informazioni fornite per ogni feature della sorgente;

3. le relazioni che intercorrono tra le features della sorgente considerata e le features delle altre banche dati importate.

Per evitare la duplicazione del codice, si è scelto di raggruppare i file con la stessa struttura che forniscono le medesime tipologie d'informazioni di feature, cioè quei file importabili tramite istanze diverse della stessa classe Loader. Per ogni gruppo di file è necessario definire a quali features si riferiscono i dati estratti, la tipologia d'informazione fornita per ogni feature (external reference, relationship, history e similarity) ed eventuali relazioni con features fornite da altre sorgenti. Tutte queste informazioni devono trovare riscontro con quanto definito in seguito nelle descrizioni delle singole features fornite dalla sorgente e delle relazioni con le altre sorgenti. In figura 10 è mostrato la definizione XML dei file forniti della sorgente ExPASy ENZYME.

```
-<data_source_definition handle="expasy_enzyme" name="expasy_enzyme" import="true" download="true">
  <description>Description of expasy_enzyme source</description>
  <importer>...ExpasyEnzymeImport</importer>
 -<download_uri>
  -<file_definition_list>
   -<file_definition loader="it.polimi.gfinder2.importer.expasyenzyme.EnzymeClassLoader">
    -<url handle="expasy_enzymeClass_DownUrl" name="ftp://.../enzclass.txt" download="true">
      <file handle="expasy_enzymeClass_File" import="true">expasy/enzclass.txt</file>
     </url>
    -<data>
     -<feature_list>
      -<feature name="enzyme">
       -<relationship name="expasy_enzyme_relationship">
         <relationship_type handle="is_a"/>
        </relationship>
       </feature>
      </feature_list>
     </data>
    </file_definition>
   -<file_definition loader="it.polimi.gfinder2.importer.expasyenzyme.EnzymeDataLoader">
    -<url handle="expasy_enzymeData_DownUrl" name="ftp://.../enzyme.dat" download="true">
      <file handle="expasy_enzymeData_File" import="true">expasy/enzyme.dat</file>
     </url>
    -<data>
     -<feature_list>
      -<feature name="enzyme">
       -<relationship name="expasy_enzyme_relationship">
         <relationship_type handle="is_a"/>
        </relationship>
       -<history name="expasy_enzyme_history">
         <history_type handle="replaced_by"/>
        </history>
       -<similarity name="expasy_enzyme_similarity">
         <similarity_type handle="alias"/>
         <match name="expasy2expasy"/>
        </similarity>
       </feature>
      </feature_list>
     -<feature_association_list>
      -<feature_association name="enzyme2protein">
        <match name="expasy2uniprot"/>
       </feature_association>
      -<feature_association name="enzyme2protein_fam_dom">
        <match name="expasy2prosite"/>
       </feature_association>
      </feature_association_list>
     </data>
    </file_definition>
   </file_definition_list>
  </download_uri>
 ...
```

Figura 10 – Esempio di definizione di file di sorgente

Come si può notare dalla figura 11, alla definizione dei file di sorgente, è definita la feature *enzyme* rappresentata nei dati forniti da ExPASy ENZYME. Per tale feature è necessario elencare:

- le espressioni regolari autoritative, cioè fornite dalla banca dati, utilizzate per verificare gli identificavi parsati;
- la tabella di sorgente *expasy_enzyme* nella quale memorizzare i dati di livello importato e le eventuali tabelle addizionali;
- relazioni di tipo gerarchico;
- relazioni di history;
- relazioni di similarità.

Non sono definite le relazioni di external reference poiché non presenti nei dati forniti dalla sorgente.

Infine sono definite le associazioni tra la feature *enzyme* fornita dalla sorgente ExPASy ENZYME e le features fornite dalle altre banche dati considerate all'interno del progetto GPDW.

```
-<feature_list>
 -<feature name="enzyme">
  -<regular_expression_list>
     <regular_expression type="id" description="regular expression for ...">([1-6])([.]([1-9]|[1-9][0-9])){0,2}</regular_expression>
     <regular_expression type="source" description="regular expression for ...">EXPASY ENZYME</regular_expression>
   </regular_expression_list>
   <definition name="enzyme" type="enzyme" main_table="expasy_enzyme"/>
  -<display_uri>
     <url prefix="http://expasy.org/enzyme/" display_class="main" description="Description of the expasy enzyme display_url"/>
     <url prefix="http://expasy.org/enzyme/" postfix=".-.-.-" display_class="class" description="Description of ..."/>
     <url prefix="http://expasy.org/enzyme/" postfix=".-.-" display_class="class" description="Description of ..."/>
   </display_uri>
  -<source table="expasy_enzyme">
    -<additional_tables>
       <additional_table name="expasy_enzyme_comment"/>
       <additional_table name="expasy_enzyme_alternative_name"/>
       <additional_table name="expasy_enzyme_action"/>
     </additional_tables>
   </source>
  -<relationship_list name="expasy_enzyme_relationship" unfold="true">
     <relationship handle="is_a"/>
   </relationship_list>
   <history name="expasy_enzyme_history"/>
  -<similarity name="expasy_enzyme_similarity">
    -<match name="expasy2expasy">
       <from_data_source name="expasy_enzyme"/>
       <to_data_source name="expasy_enzyme"/>
     </match>
   </similarity>
 </feature>
</feature_list>
-<feature_association_list>
 -<feature_association name="enzyme2protein" source_feature="enzyme" destination_feature="protein">
  -<match name="expasy2uniprot">
     <from_data_source name="expasy_enzyme"/>
     <to_data_source name="uniprot"/>
   </match>
 </feature_association>
 -<feature_association name="enzyme2protein_fam_dom" source_feature="enzyme" destination_feature="protein_fam_dom">
  -<match name="expasy2prosite">
     <from_data_source name="expasy_enzyme"/>
     <to_data_source name="prosite"/>
   </match>
 </feature_association>
</feature association list>
 ...
```

Figura 11 – Esempio di definizione di feature e di associazioni tra features

5.2 Importer di sorgente dati generico

La precedente versione del framework prevedeva la realizzazione di una classe Importer per ogni sorgente importata. L'aggiunta di una nuova sorgente da importare all'interno del data warehouse prevedeva quindi la duplicazione di codice precedentemente definito o nel peggiore dei casi la creazione di nuovo codice che non sempre risultava seguire uno standard comune per tutti gli Importer. Ciò è un limite progettuale poiché eventuali modifiche strutturali al core del sistema potevano richiedere la modifica di ogni singola classe definita ad hoc per l'importazione dei dati.
Le funzionalità di base di un Importer sono nell'ordine:

1. istanziare un oggetto della classe Loader, definita per eseguirne il parsing e l'importazione dei dati nel data warehouse per ogni file fornito dalla sorgente;
2. eseguire il *"run"* dei Loader istanziati, cioè eseguire l'importazione vera e propria dei dati nel data warehouse;
3. chiudere tutte le connessioni aperte, eseguire il *"commit"* delle modifiche o, nel caso in cui quest'ultimo fallisca, il "*rollback*".

Ho, quindi, creato una nuova classe *GenericImporter.java,* all'interno del package *it\polimi\gfinder2\importer\generic\,* nella quale sono definite in modo standardizzato le procedure appena descritte. Tale classe può essere direttamente utilizzata per richiamare i Loader realizzati per l'importazione di ogni singolo file di sorgente, può essere estesa da una nuova classe Importer, nella quale aggiungere eventuali procedure specifiche per la sorgente considerata, o nella quale ridefinire i metodi ereditati, nel caso in cui sia richiesto un trattamento diverso da quello standard.
In figura 12 è presentato il diagramma di workflow dell'Importer generico.

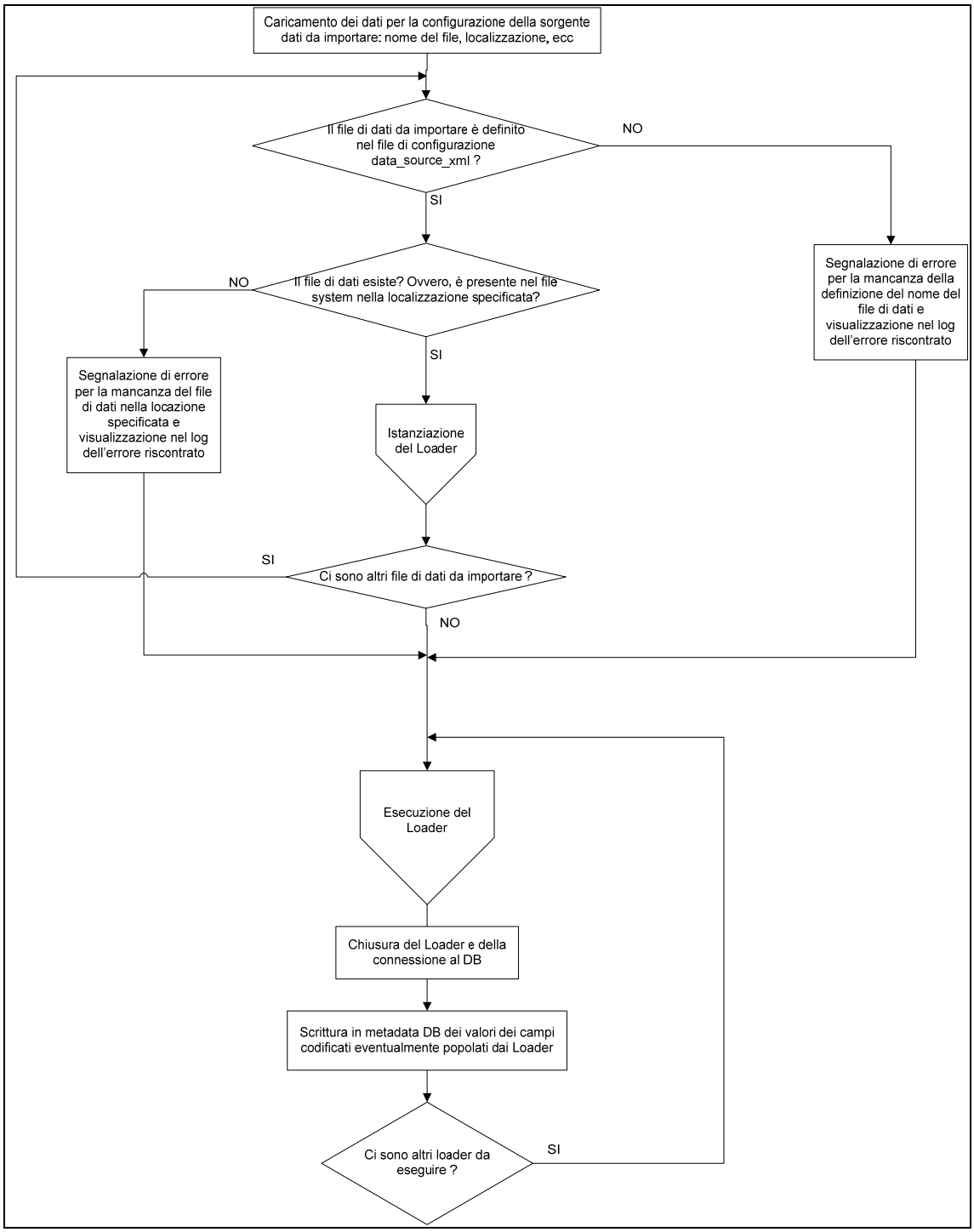

Figura 12 – Workflow Importer di sorgente generico

5.3 Loader dati generico

Il Loader è quel componente che si occupa di processare i dati estratti dai file forniti dalla banca dati e di inserirli nelle tabelle create nella base di dati.
Il funzionamento del Loader può essere riassunto in quattro fasi:

1. inizializzazione degli oggetti utilizzati per l'importazione dei dati nel data warehouse;
2. parsing del file;
3. inserimento nel data warehouse dei dati estratti tramite gli Importer di dati specifici;
4. chiusura delle connessioni aperte, esecuzione del *"commit"* delle modifiche ed esposizione delle statistiche.

Di queste quattro fasi solo una era eseguita in modo standardizzato, la fase di "*parsing*", durante la quale viene analizzato il file da importare e vengono estratte le informazioni in esso contenute. Le restanti operazioni dovevano essere implementate ad hoc per ogni singolo Loader. È evidente la necessità di definire in maniera astratta il comportamento desiderato del Loader, in modo che possano essere implementate genericamente tutte le sue funzionalità.
Tramite le informazioni definite nel file di configurazione *data_sources.xml,* riguardanti la tipologia dei dati contenuti nei file di sorgente, è possibile passare al Loader generico la struttura delle tabelle sorgenti e segnalare la presenza di relazioni di external reference, internal relationship, history, similarità e associazione tra features. Il Loader può, quindi, istanziare in modo automatico gli Importer di dati specifici (vedi sottocapitolo 5.5), le liste di entries atte a contenere i dati estratti ed eventuali vettori di parametri opzionali. Ogni oggetto istanziato è inserito all'interno di un HashMap, utilizzando come chiave il nome inserito nel file di configurazione XML per definire l'informazione, in modo che vi si possa accedere facilmente.
Nelle figure 13 e 14 è rappresentato il diagramma di workflow della fase di configurazione del Loader generico.

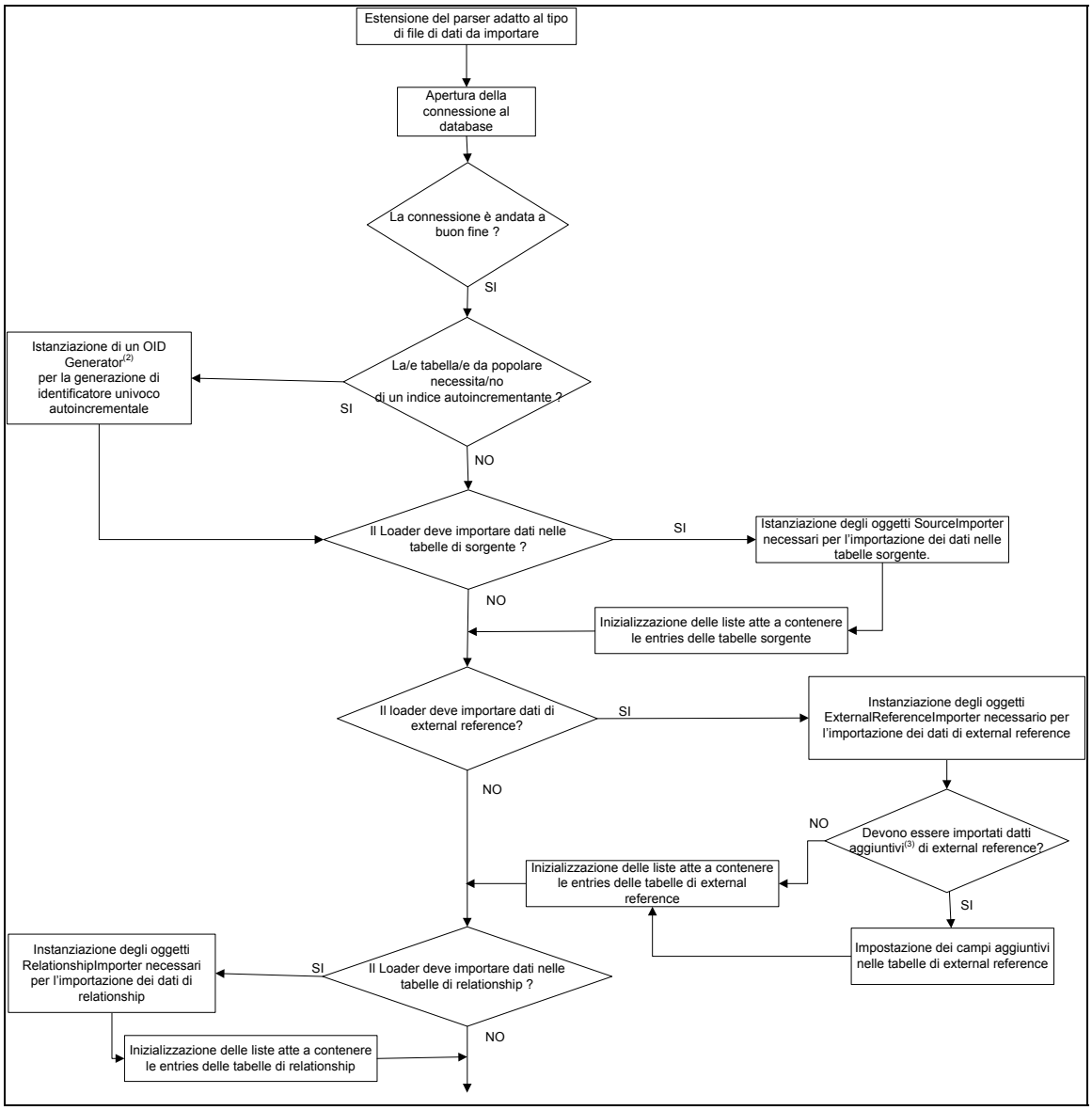

Figura 13 – Workflow Loader generico: configurazione (prima parte)

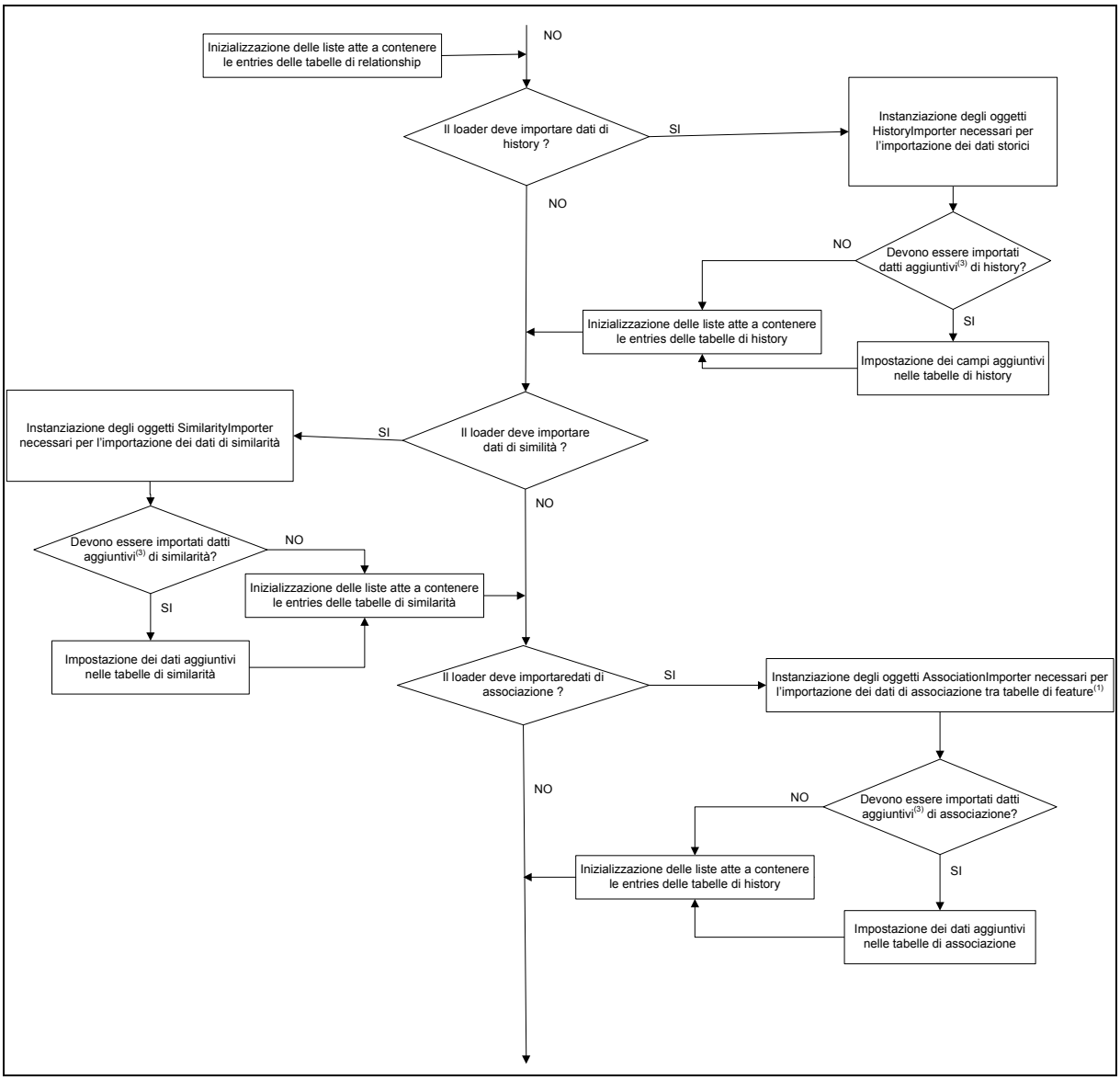

Figura 14 – Workflow Loader generico: configurazione (seconda parte)

In seguito, sarà eseguito il parsing del file utilizzando una delle classi generiche realizzate per tale scopo, o, nel caso in cui tale classe non sia già presente all'interno del framework, tramite una nuova classe Parser. Nel Loader specifico andrà solo definito l'inserimento dei dati estratti nelle liste di entries istanziate in modo automatico durante la fase precedente (vedi figura 15).

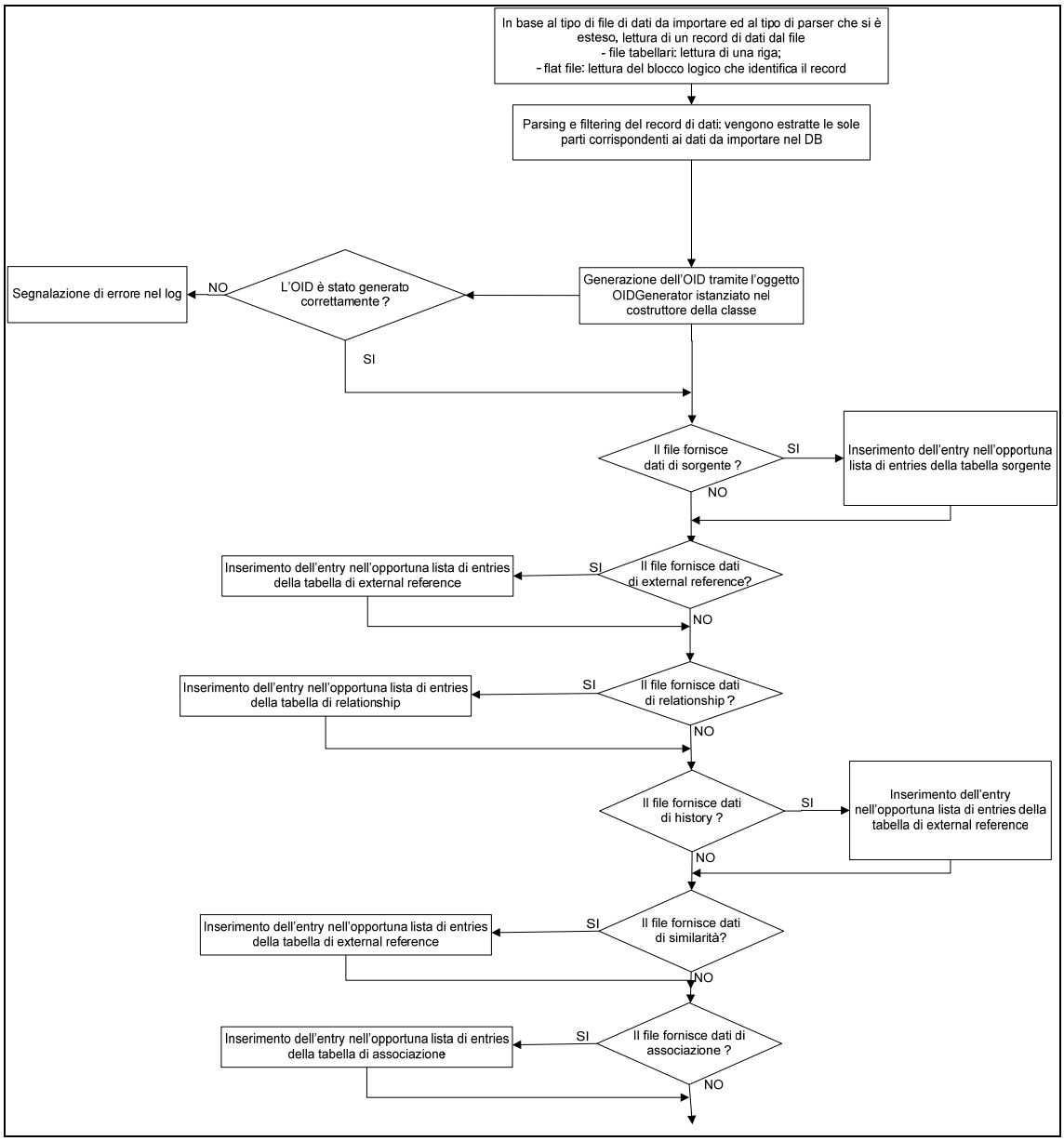

Figura 15 – Workflow Loader generico: parsing del file

Al termine del parsing del file, il Loader generico si occuperà di processare in modo automatico le liste di entries, di eseguire il popolamento delle tabelle del database richiamando i metodi *"insert"* degli Importer di dati specifici e genererà le informazioni di tipo statistico da mostrare nel log dell'applicazione (figura 16).

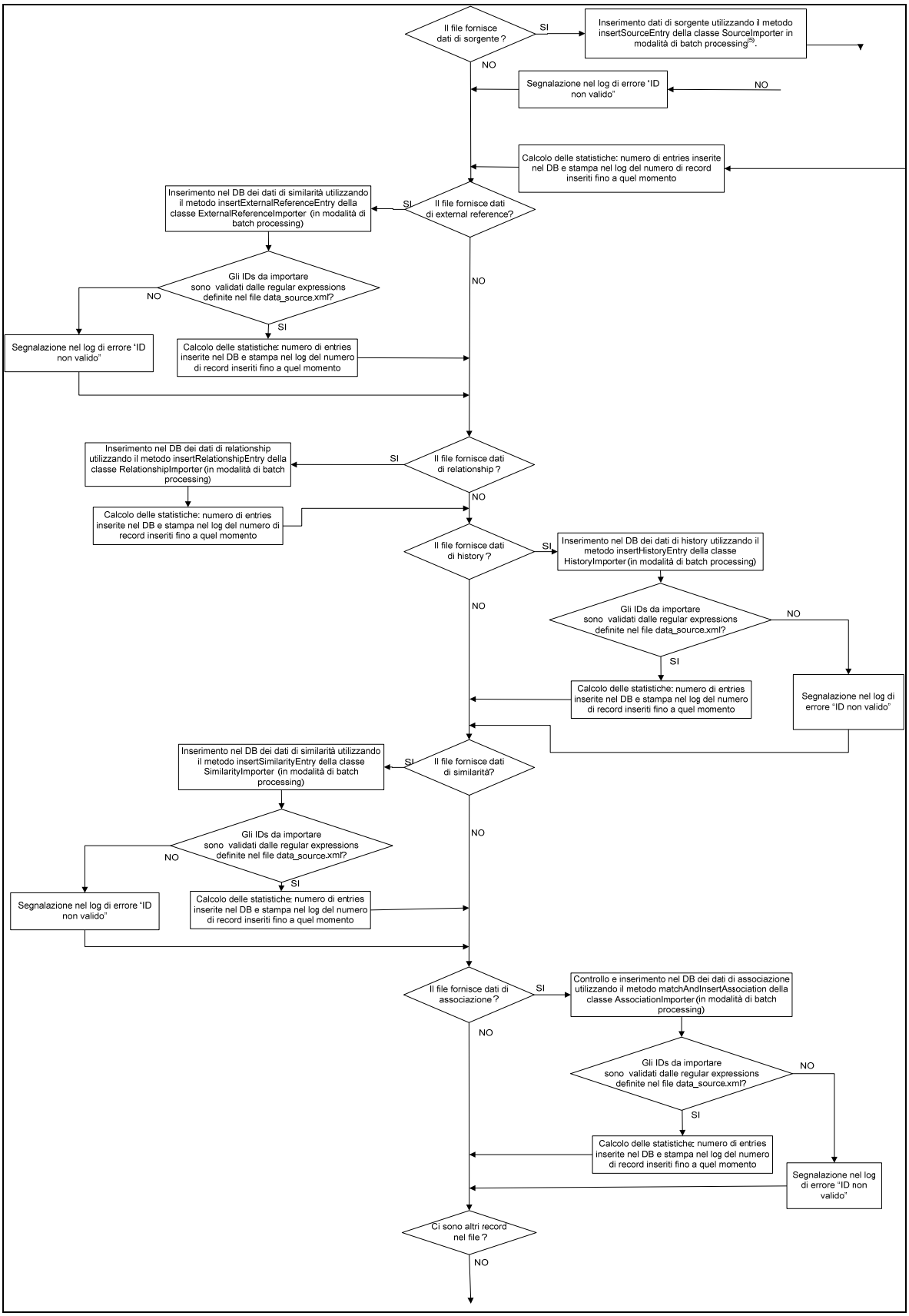

Figura 16 – Workflow Loader generico: inserimento dei dati estratti in DB

Infine, sempre in modo automatico, avverrà la chiusura di tutte le connessioni aperte, il *"commit"* delle modifiche e il calcolo dei dati statistici riguardanti l'importazione globale del file di sorgente, da inserire nelle tabelle dello schema *metadata* (figura 17).

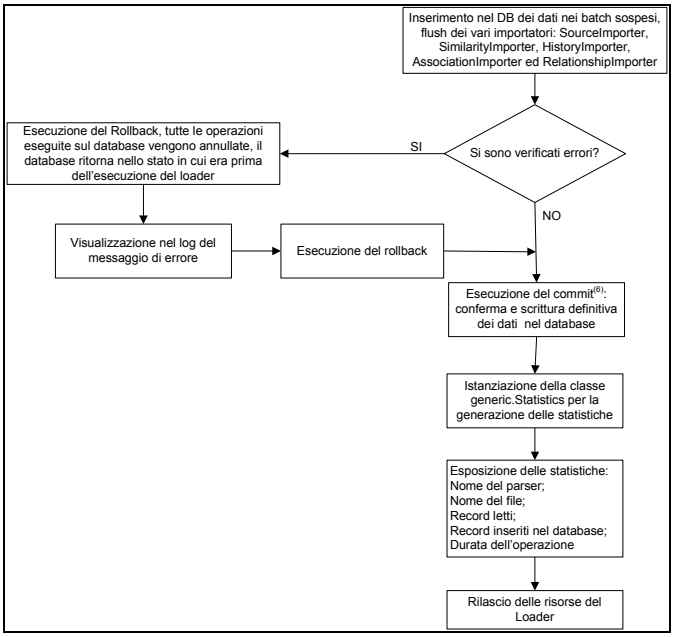

Figura 17 – Workflow Loader generico: chiusura connessione ed esposizione statistiche

5.4 Gestione delle espressioni regolari: IdentifierMatcher

Come descritto in precedenza nel quarto capitolo, una delle modifiche prioritarie del framework riguarda una diversa gestione delle espressioni regolari definite nel file di configurazione XML e il disaccoppiamento della logica di riconoscimento degli identificativi, estratti dai file di sorgente, dal componente RelationshipMatcher (rinominato AssociationImporter), il quale dovrà solo occuparsi del loro inserimento nelle tabelle di associazione tra features.

Si è scelto di definire per ogni feature le espressioni regolari autoritative, fornite dalla sorgente per riconoscere i propri identificativi, e separatamente, all'interno dei tag <*match*>, eventuali espressioni regolari non autoritative. Come si può notare nell'esempio riportato in figura 18, nel caso la sorgente utilizzi per rappresentare gli identificativi o il nome della sorgente coinvolta nell'associazione un formato diverso da quello autoritativo è possibile definire all'interno dei tag <*match_reg_expr_rule*> le espressioni regolari non autoritative e, all'interno del tag <*convert_reg_expr*>, il "replacement" da utilizzare per convertire la stringa dal formato non autoritativo a quello autoritativo.

```
-<feature_list>
  -<feature name="enzyme">
    -<regular_expression_list>
       <regular_expression type="id" description="...">([1-6])([.]([1-9]|[1-9][0-9])){0,2}</regular_expression>
       <regular_expression type="source" description="...">expasy_enzyme</regular_expression>
     </regular_expression_list>
      ...
   </feature>
 </feature_list>
-<feature_association_list>
  -<feature_association name="enzyme2protein_fam_dom" source_feature="enzyme" destination_feature="protein_fam_dom">
    -<match name="expasy2prosite">
       <from_data_source name="expasy_enzyme"/>
      -<regular_expression_list>
        -<regular_expression type="name" description="...">
           <match_reg_expr>(expasy)PROVA(enzyme)</match_reg_expr>
           <convert_reg_expr>$1_$2</convert_reg_expr>
         </regular_expression>
       </regular_expression_list>
       <to_data_source name="prosite"/>
     </match>
   </feature_association>
 </feature_association_list>
```

Figura 18 – Esempio di definizione di espressioni regolari e associazioni tra feature

Parallelamente alla ristrutturazione del codice è stato necessario modificare le procedure automatiche presenti nel framework. Ho creato una nuova classe *IdentifierMatcher.java,* presente all'interno del package *it.polimi.gfinder2.importer.generic*, nella quale sono definiti i metodi, utilizzati dagli Importer di dati specifici, descritti nel capitolo successivo, per estrarre dal file di configurazione *data_sources.xml* le espressioni regolari necessarie per eseguire il riconoscimento e l'eventuale conversione degli identificativi parsati prima del loro inserimento all'interno del data warehouse.

Tutte le espressioni regolari definite nel file di configurazione devono essere riportate nel formato PCRE[19] riconosciuto dalla classe *Matcher.java,* utilizzata per eseguire il confronto e la conversione.

Attualmente sono definite due tipologie di espressioni regolari, *"id"* e *"source"*, utilizzate per riconoscere rispettivamente gli identificativi forniti dalla sorgente e il nome della banca dati importata. Ovviamente tale definizione può essere facilmente estesa.

5.5 Importer di dati specifici

La ristrutturazione del codice descritta nei capitoli precedenti ha richiesto la ridefinizione degli Importer di dati specifici utilizzati dal Loader generico per memorizzare i dati di sorgente, external reference, relationship, history, similarity e associazione tra features. Per ogni tipologia di dato è stata definita una classe specifica atta a rappresentare l'entries da memorizzare. Sarà compito del Loader specifico istanziare gli oggetti di tipo *"entry"* opportuni con i dati estratti dai file di sorgente e inserirli nelle liste di entries utilizzate dal Loader generico per eseguire l'inserimento in tabella in modo automatico.

L'esecuzione di un Importer di dati è stata formalizzata e può essere riassunta in tre fasi principali:

1. configurazione dell'Importer: viene inizializzato l'oggetto della classe Importer; si apre la connessione verso il database, viene creata la tabella atta a contenere i dati

estratti e, se sono presenti campi codificati, vengono definiti i flag necessari. Inoltre, durante questa fase viene istanziato un nuovo IdentifierMatcher, il quale, grazie ai parametri definiti nel file di configurazione XML, recupererà tutte le espressioni regolari necessarie a identificare gli identificativi da memorizzare all'interno del data warehouse;

2. match degli ID e inserimento in tabella dell'entry: l'identificativo di ogni entry viene confrontato, utilizzando il componente IdentifierMatcher, con le espressioni regolari precedentemente estratte dal file di configurazione XML. Se il match ha esito negativo, l'errore è segnalato al Loader che provvederà a mostrare un opportuno messaggio nel log dell'applicazione. Viceversa, l'Importer inserisce l'entry in tabella e segnala al Loader la buona riuscita dell'operazione;
3. *"dispose"*: durante questa fase tutte le connessioni aperte dall'Importer sono chiuse, i dati di flag vengono memorizzati nel database e sono generati i dati di statistica.

In figura 19 è presentato un tipico scenario d'importazione di dati di associazione.

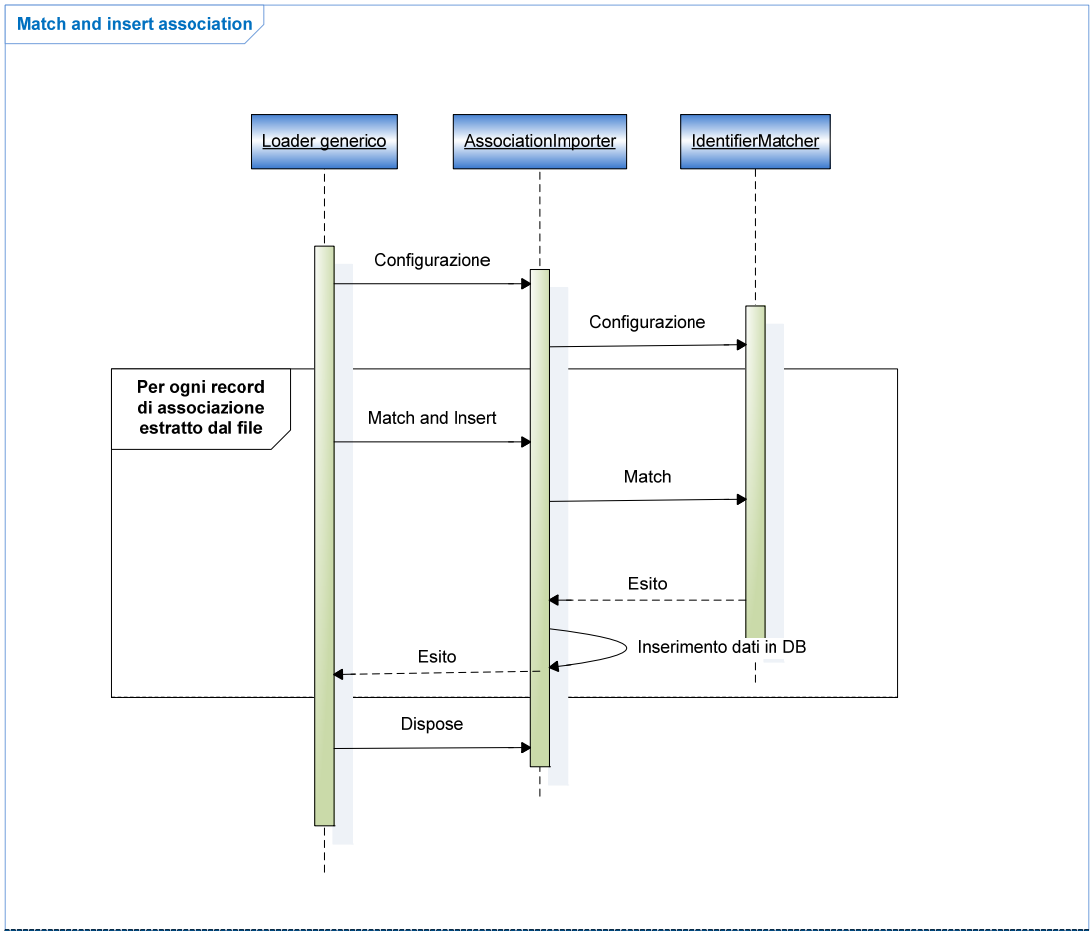

Figura 19 – Esempio di scenario d'importazione di entries di associazione

Il lavoro di ristrutturazione ha portato all'implementazione o ridefinizione di sei classi:

1. *SourceImporter.java*: crea la tabella di feature di livello importato, definisce i flag associati ai campi codificati, ed esegue, in modo automatico, l'inserimento dei dati

estratti dal file in tabella, curandosi di verificare che gli identificativi estratti siano conformi con le espressioni regolari definite nell'XML;

2. *ExternlReferenceImporter.java*: crea la tabella di external reference, definisce i flag associati ai campi codificati, esegue il confronto degli identificativi e dei nomi di sorgente estratti dal file con le espressioni regolari definite nel file di configurazione *data_sources.xml* (tramite un oggetto istanza della classe *IdentifierMatcher.java*) ed effettua l'inserimento dei dati nel database in modo automatico;
3. *RelationshipImporter.java*: crea la tabella di relationship, definisce i flag associati ai campi da codificare e inserisce i dati all'interno del data warehouse;
4. *HistoryImporter.java*: crea la tabella di history, definisce i flag associati ai campi da codificare ed esegue, in modo automatico, l'inserimento dei dati nella tabella creata nel caso gli identificativi da memorizzare siano conformi con le espressioni regolari definite nel file di configurazione XML*;*
5. *SimilarityImporter.java*: crea la tabella di similarità, definisce i flag associati ai campi da codificare e inserisce i dati estratti dal file all'interno del data warehouse in modo automatico, solo se gli identificativi da memorizzare sono validati dalle espressioni regolari definiti nel file *data_sources.xml* .

6 Eliminazione/merge delle tuple duplicate: DuplicatesChecker

Il componente *DuplicatesChecker* è stato realizzato per soddisfare l'esigenza di gestione delle tuple duplicate presenti nelle tabelle di livello importato (il problema non si pone per le tabelle di livello integrato, essendo queste derivate dalle tabelle di livello importato). Come anticipato nel quarto capitolo, ogni data record memorizzato nel data warehouse è identificato in modo univoco, sia a livello importato che integrato, attraverso due campi: l'identificativo fornito dalla banca dati, e il nome della sorgente dal quale è stato estratto. Nel caso in cui una sorgente fornisca diverse tuple riferite alla stessa feature, è evidente che non sarà possibile identificarla in modo univoco. La classe *DuplicatesChecker.java* dovrà quindi occuparsi di aggregare le informazioni disposte su tuple diverse, riferite a una stessa feature, in un'unica tupla, e nel caso in cui ciò non sia possibile eliminare i record duplicati. In seguito, dovrà aggiornare i valori dei campi chiave nelle tabelle legate a tuple tramite un vincolo di chiave esterna a tabelle sulle quali è stata eseguita la procedura.
Realizzando tale componente in modo parametrico è stato possibile non solo gestire le entries duplicate riferite a una stessa feature, ma applicare la procedura in modo automatico a tutte le tabelle per le quali è stato definito un vincolo di unicità.
Nel sottocapitolo successivo sono approfondite le scelte progettuali e le soluzioni adottate per la realizzazione del componente descritto.

6.1 Implementazione

La classe *DuplicatesChecker.java*, presente all'interno del package *it.polimi.gfinder2.importer.generic.extref*, è composta da quattro metodi:

- il *costruttore*;
- *configure( )*;
- *check( )*;
- *secondaryTableCheck( )*.

Il *costruttore* inizializza i parametri e controlla se la tabella,sulla quale eseguire il controllo, esiste. Riceve in input:

- lo schema nel quale è memorizzata la tabella da analizzare; parametro opzionale, nel caso in cui non sia passato lo schema di default è *public*;
- il nome della tabella sulla quale eseguire il controllo;
- lista dei campi appartenenti all'indice univoco;
- lista dei campi appartenenti alla PK (Primary Key), se presente;

- lista dei campi bitmap, per i quali l'operazione di merge dovrà essere eseguita tramite l'OR booleano (ad es. il campo *reference_file,* in modo da tenere traccia di tutti i file di sorgente che hanno fornito le informazioni memorizzate nella tupla).

In seguito all'istanziazione dell'oggetto, deve essere invocato il metodo *configure( )* che si occupa di costruire le query necessarie alla verifica. È possibile definire, in modo parametrico, un suffisso da utilizzare nella creazione delle tabelle generate dalla procedura nello schema *log,* utile, ad esempio, per chiamate ripetute della funzione sulla stessa tabella.
Il passo successivo consiste nell'esecuzione del metodo *check( )*; esso rappresenta il cuore della procedura, ha il compito di gestire le tuple con valori dei campi univoci duplicati. Può essere invocato in due modalità: merge delle entries duplicate o eliminazione/merge delle entries duplicate. Inizialmente, il metodo identifica e, se invocato in modalità eliminazione/merge duplicati, elimina le tuple aventi valori dei campi univoci duplicati per le quali non è possibile eseguire il merge, in quanto presentano valori discordanti dei campi non univoci. Le tuple eliminate sono inserite in una nuova tabella generata nello schema *log,* chiamata **_err* (il carattere * rappresenta il nome della tabella su cui è applicata la procedura).
Successivamente, sono rimosse dalla tabella sulla quale è applicata la procedura le entries aventi valori dei campi univoci duplicati per le quali è possibile eseguire il merge, cioè con valori dei campi non univoci uguali e/o *Null*. Tali entries vengono inserite nella tabella **_dup* generata nello schema *log*. A questo punto viene eseguito il merge di tali tuple, applicando la funzione sql *bool_or* ai campi booleani, *bit_or* ai campi bitmap e *MAX* agli altri campi (dato che hanno valori uguali e/o *NULL*, la funzione restituisce il valore non nullo se presente, altrimenti *NULL).* Il risultato di questa query viene memorizzato in una nuova tabella creata nello schema *log*, **_dist*, e reinserito nella tabella originale.
Le tabelle 1 e 2 mostrano il risultato dell'applicazione della procedura descritta su una tabella di test avente come chiave primaria il campo *gene_oid,* i campi *source_name* e *source_id* appartenenti all'indice univoco e il campo *reference_file* di tipo bitmap, da unire tramite OR booleano.

gene_oid	source_id	source_name	reference_file	taxonomy_id	symbol	creation_date	creation_author
1845	100500	0001	0001	9606		6/4/1986	McKusick
46	100640	0001	0001	9606	ALDH1A1	6/4/1986	McKusick
10973	100640	0001	0010	9606	ALDH1A1		
49	100650	0001	0001	9606	ALDH2	6/4/1986	McKusick
13120	100650	0001	0010	9606	ALDH2		
18404	100650	0001	0100	9606	ALDH2		
50	100660	0001	0001	9606	ALDH3A1	6/4/1986	McKusick
14906	100660	0001	0100	9606	ALDH3A1		

Tabella 1 – Tabella di test con entries duplicate

gene_oid	source_id	source_name	reference_file	taxonomy_id	symbol	creation_date	creation_author
1845	100500	0001	0001	9606		6/4/1986	McKusick
10973	100640	0001	0011	9606	ALDH1A1	6/4/1986	McKusick
18404	100650	0001	0111	9606	ALDH2	6/4/1986	McKusick
14906	100660	0001	0101	9606	ALDH3A1	6/4/1986	McKusick

Tabella 2 – Tabella di test risultante dall'applicazione della procedura DuplicatesChecker

La figura sottostante mostra le query create dalla procedura in seguito alla sua applicazione alla tabella d'esempio, nell'ordine di esecuzione.

```
- CREATE TABLE log.omim_gene_err AS (SELECT DISTINCT ON( a.omim_gene_oid, a.source_id,
a.source_name, a.reference_file, a.taxonomy_id, a.symbol, a.creation_date, a.creation_author)
a.omim_gene_oid, a.source_id, a.source_name, a.reference_file,a.taxonomy_id, a.symbol,
a.creation_date FROM public.omim_gene as a JOIN public.omim_gene as b ON a.omim_gene_oid <>
b.omim_gene_oid AND (a.source_id = b.source_id) AND (a.source_name = b.source_name) WHERE
(a.taxonomy_id <> b.taxonomy_id AND a.taxonomy_id IS NOT NULL AND b.taxonomy_id IS NOT NULL)
OR  (a.symbol <> b.symbol AND a.symbol IS NOT NULL AND b.symbol IS NOT NULL) OR
(a.creation_date <> b.creation_date AND a.creation_date IS NOT NULL AND b.creation_date IS NOT
NULL)  OR  (a.creation_author <> b.creation_author AND a.creation_author IS NOT NULL AND
b.creation_author IS NOT NULL)

-DELETE FROM public.omim_gene WHERE omim_gene_oid IN (SELECT omim_gene_oid FROM
log.omim_gene_err)

-CREATE TABLE log.omim_gene_dup AS (SELECT a.omim_gene_oid, a.source_id, a.source_name,
a.reference_file, a.taxonomy_id, a.symbol, a.creation_date, a.creation_author FROM
public.omim_gene as a JOIN public.omim_gene as b ON a.omim_gene_oid <> b.omim_gene_oid AND
(a.source_id = b.source_id) AND (a.source_name = b.source_name))

-CREATE TABLE log.omim_gene_dist AS ( SELECT  MAX(omim_gene_oid) AS omim_gene_oid,
MAX(source_id) AS source_id,  bit_or(source_name) AS source_name, bit_or(reference_file) AS
reference_file, MAX(taxonomy_id) AS taxonomy_id, MAX(symbol) AS symbol, MAX(creation_date) AS
creation_date, MAX(creation_author) AS creation_author FROM log.omim_gene_dup GROUP BY
source_id, source_name)

-DELETE FROM public.omim_gene WHERE omim_gene_oid IN (SELECT omim_gene_oid FROM
log.omim_gene_dup)

-INSERT INTO public.omim_gene (SELECT omim_gene_oid, source_id, source_name, reference_file,
taxonomy_id, symbol, creation_date, creation_author FROM log.omim_gene_dist)
```

Figura 20 – Esempio di query generate dalla procedura DuplicatesChecker

Infine, occorre invocare il metodo *secondaryTableCheck( )* sulle "tabelle secondarie" (tabelle legate alla tabella principale tramite un vincolo di chiave esterna), in modo da aggiornare i valori dei campi appartenenti alla FK (Foreign Key) riferiti a tuple modificate dalla procedura.

Questa metodo riceve in input il nome della tabella secondaria, la lista dei campi appartenenti al vincolo di FK e i campi della tabella principale cui fanno riferimento. Inoltre è possibile scegliere se limitarsi ad aggiornare i valori dei campi appartenenti alla chiave esterna oppure se eseguire l'aggiornamento ed eliminare le entries "disconnesse" dalla tabella principale (cioè le entries che anche dopo l'aggiornamento si riferiscono a valori non più presenti nella tabella principale).
Inizialmente sono aggiornati i valori riferiti a entries presenti nella tabella **_err*, generata dall'applicazione della procedura per l'eliminazione/merge duplicati sulla tabella principale. Tramite il *"join"* tra la tabella principale e la tabella **_err* sui campi appartenenti all'indice univoco, si ricavano i valori dei campi chiave esterna da aggiornare, in quanto si riferiscono a tuple eliminate dalla tabella principale dalla procedura di eliminazione/merge delle tuple duplicate, e i corrispondenti valori aggiornati, in quanto è presente nella tabella principale una tupla avente gli stessi valori dei campi dell'indice univoco della tupla eliminata, che quindi identifica lo stesso oggetto. A questo punto è possibile eseguire l'update dei valori dei campi appartenenti alla FK nella tabella secondaria. Lo stesso procedimento è in seguito applicato ai valori di chiave esterna riferiti ad entries presenti nella tabella **_dup* dello schema *log*: Tramite il *"join"* sui campi appartenenti all'indice univoco tra la tabella principale e la tabella **_dup* dello schema *"log"* si ricavano i valori dei campi chiave esterna da aggiornare, in quanto si riferiscono a tuple unificate tramite merge in una nuova tupla, e i corrispondenti valori aggiornati, cioè le chiavi primarie delle tuple risultato dell'operazione di merge.
Nelle figure sottostanti è proposto il diagramma in cui viene schematizzato il workflow della classe *DuplicatesChecker.java* appena descritto.

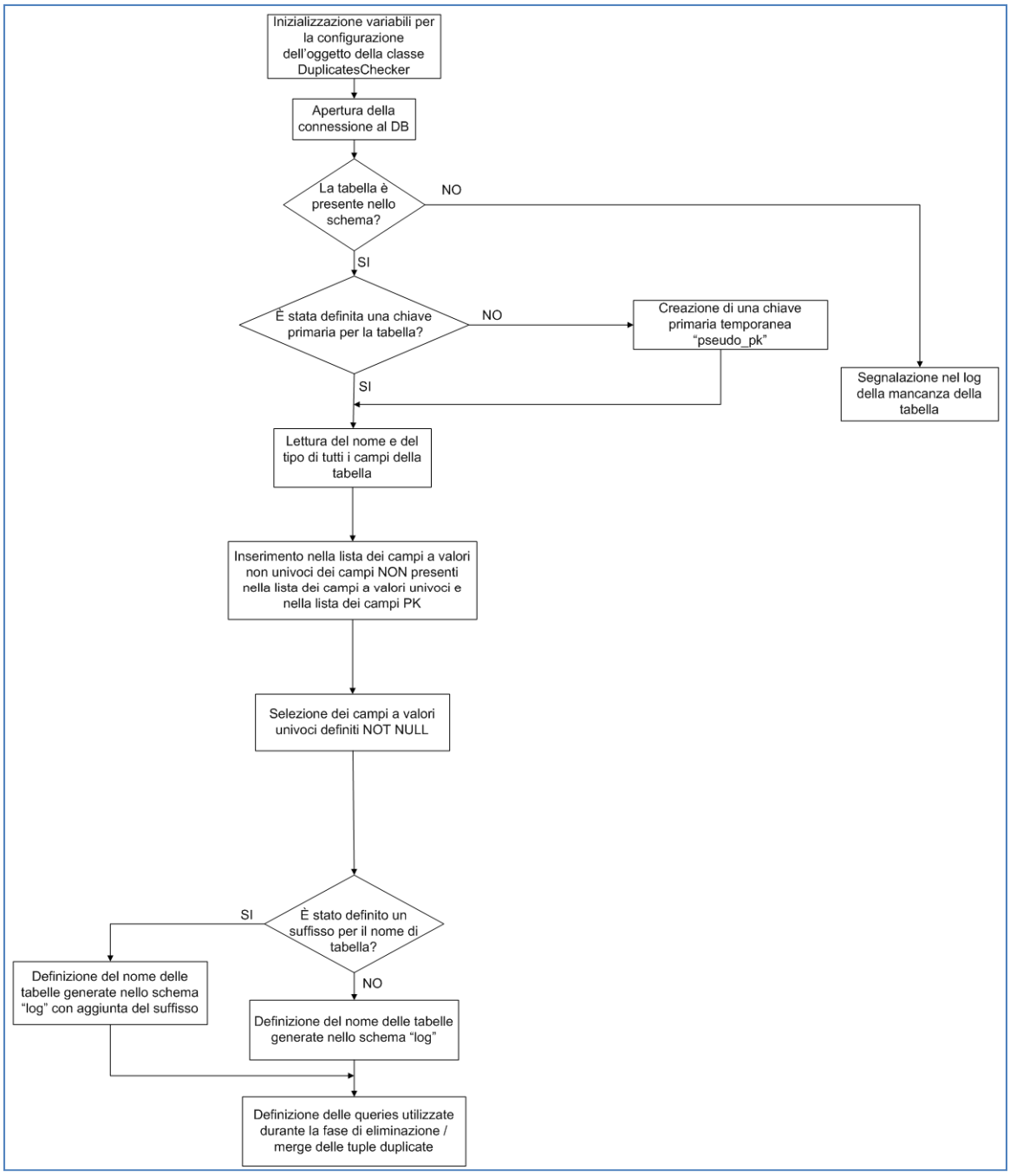

Figura 21 – Workflow DuplicatesChecker: inizializzazione e configurazione delle query

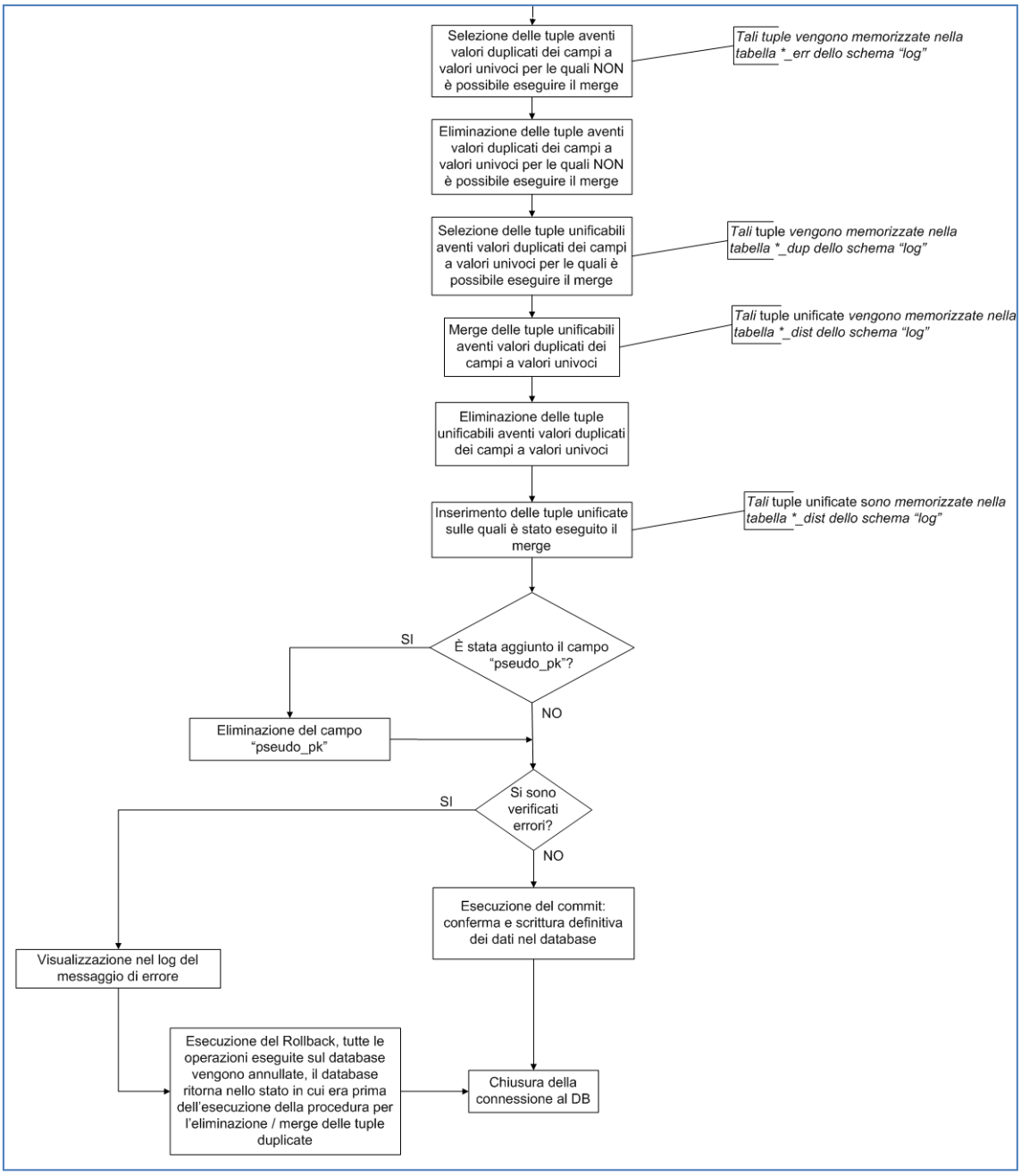

Figura 22 – Workflow DuplicatesChecker: eliminazione/merge delle tuple duplicate

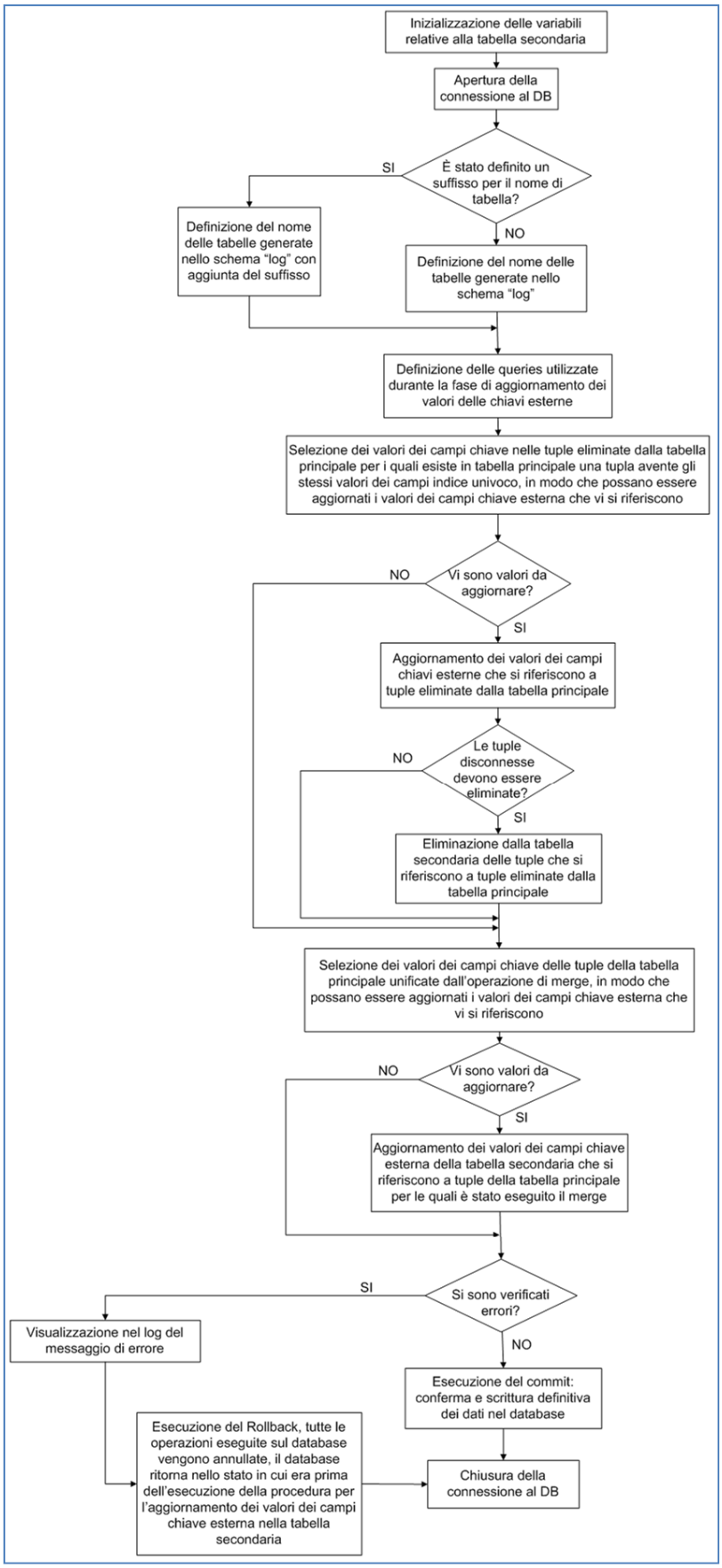

Figura 23 – Workflow DuplicatesChecker: aggiornamento valori di foreign key

7 Banche dati considerate

7.1 Banca dati ExPASy ENZYME – Expert Protein Analysis System ENZYME

ExPASy (Expert Protein Analysis System)[20] è il server per la proteomica del Swiss Institute of Bioinformatics (SIB).
Il SIB è una fondazione accademica nata nel 1988 al fine di promuovere la ricerca, lo sviluppo di banche dati e l'insegnamento della bioinformatica in Svizzera, in collaborazione con altri paesi.
ExPASy è un servizio per la comunità scientifica dedicato all'analisi della struttura e della sequenza delle proteine. E' stato creato nel 1993 ed è uno dei primi server dedicati alle scienze biologiche. Il sito permette l'accesso a molti strumenti analitici per l'identificazione di proteine, l'analisi della loro sequenza e previsioni sulla struttura terziaria.
In particolare, il server di ExPASy fornisce l'accesso verso una vasta varietà di basi di dati genomiche e proteomiche, ponendo particolare attenzione alla loro integrazione e interconnettività.
ExPASy svolge il ruolo di host principale delle seguenti basi di dati, sviluppate parzialmente o totalmente presso il SIB di Ginevra:

- SWISS-PROT
- SWISS-2DPAGE
- PROSITE
- ENZYME
- SWISS MODEL

In particolare, ENZYME è un archivio d'informazioni di nomenclatura di enzimi.
Tutte le basi di dati disponibili su ExPASy presentano numerosi cross-references verso altre banche di dati genetici e genomici e risorse utili.
I file della banca dati ExPASy ENZYME analizzati sono i seguenti:

- *enzclass.txt*: riporta in forma gerarchica la classificazione internazionale degli enzimi;
- *enzyme.dat*: elenco di enzimi ordinati per Enzyme Commission Number (EC number);

entrambi disponibili al seguente indirizzo Web:
ftp://ftp.expasy.org/databases/enzyme/.
Il file *enzclass.txt* è di tipo tabellare con intestazione e i campi sono suddivisi da singolo separatore, mentre il file *enzyme.dat,* il quale fornisce l'intero database di ExPASy Enzyme, è di tipo flat.

7.2 Banca dati OMIM – Online Mendelian Inheritance in Man

OMIM è una sorgente completa, autorevole e costantemente aggiornata di geni umani e fenotipi genetici.
I dati forniti da OMIM contengono informazioni su tutti i disturbi mendeliani noti e oltre 12.000 geni. OMIM si concentra sul rapporto tra genotipo e fenotipo. È aggiornato quotidianamente, e la ricchezza dei suoi dati è dovuta ai numerosi link presenti che rimandano ad altre risorse di informazioni genetiche.
Questo database è stato avviato nei primi anni 1960 dal Dott. Victor A. McKusick, inizialmente fu pensato come un catalogo dei caratteri mendeliani e dei disturbi, intitolato Mendelian Inheritance in Man (MIM).
Dal 1966 al 1998 sono state prodotte e pubblicate dodici edizioni di libri di MIM; nel 1985 è stata creata una versione online, che prende il nome di OMIM, nata da una collaborazione tra la National Library of Medicine e la William H. Welch Medical Library di Johns Hopkins.
Tale database online è stato reso disponibile su Internet dal 1987 e nel 1995 è stata sviluppata una versione per il World Wide Web dal NCBI, il National Center Biotechnology Information.
OMIM è scritto e modificato dal McKusick-Nathans Institute of Genetic Medicine e dalla Johns Hopkins University School of Medicine, sotto la direzione del Dott. Ada Hamosh.
La differenza principale tra i libri di MIM e OMIM consiste nella maggiore attualità della versione online. La banca dati online è aggiornata quotidianamente, mentre il libro contiene tutte le informazioni disponibili al momento della stampa.
L'obiettivo principale di OMIM è supportare il lavoro svolto da medici, ricercatori e studenti nel campo della scienza e della medicina.
I file della banca dati OMIM analizzati sono i seguenti:

- *omim.txt*: descrizione di geni umani e fenotipi genetici;
- *genemap*: elenco dei geni presenti in OMIM, disposti per ubicazione citogenetica;
- *morbidmap*: lista ordinata alfabeticamente delle malattie genetiche presenti in OMIM;
- *pubmed_cited*: elenco di riferimenti di pubblicazioni riguardanti geni e disturbi genetici;
- *genemap.key:* descrizione dei campi codificati contenuti nel file *genemap*;

tutti disponibili al seguente indirizzo Web: ftp://ftp.ncbi.nih.gov/repository/OMIM/.
I file *genemap*, *morbidmap* e *pubmed_cited* sono di tipo tabellare senza intestazione e i campi sono suddivisi da singolo separatore, mentre il file *genemap.key* e *omim.txt,* che fornisce l'intero database OMIM, sono di tipo flat.

8 Progettazione della base di dati

La progettazione del database è strutturata in tre passi:

- progettazione concettuale: consiste nella creazione dello schema concettuale o diagramma ER (entità relazione) che ha lo scopo di rappresentare le specifiche informali della realtà di interesse in termini di una descrizione formale completa ma indipendente dai criteri di rappresentazione utilizzati nel sistema di gestione della basi di dati. In queste rappresentazioni non si tiene conto né delle modalità con le quali queste informazioni verranno codificate né dell'efficienza dei programmi che utilizzeranno queste informazioni;
- progettazione logica: consiste nella traduzione dello schema concettuale, definito nella fase precedente, nel modello della rappresentazione dei dati adottato dal sistema di gestione della base di dati utilizzato. In questa fase si produce lo schema logico, il quale fa riferimento ad un modello logico dei dati; come nella fase precedente la rappresentazione risulta indipendente dai dettagli fisici. In questo caso le scelte progettuali si basano su criteri di ottimizzazione delle operazioni da compiere sui dati;
- progettazione fisica: in questa fase lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati, organizzazione dei file e degli indici. Questa fase fa riferimento a un modello fisico dei dati e dipende dallo specifico sistema di gestione dei dati scelto.

La prime due fasi prevedono la creazione degli schemi tramite l'utilizzo di software di progettazione; nel mio caso ho utilizzato Microsoft Visio ®, mentre la progettazione fisica è un'operazione automatica eseguita dal framework, grazie alle informazioni di sorgente e di feature definite nei file di configurazione XML.

8.1 Banca dati ExPASy ENZYME

8.1.1 Schema concettuale

In figura 24 è rappresentato il diagramma concettuale della banca dati ExPASy ENZYME, scaturito dall'analisi dei file *enzclass.txt* e *enzyme.dat*.
Alcuni attributi sono rappresentati in colore rosso a indicare la loro omissione in fase d'importazione.

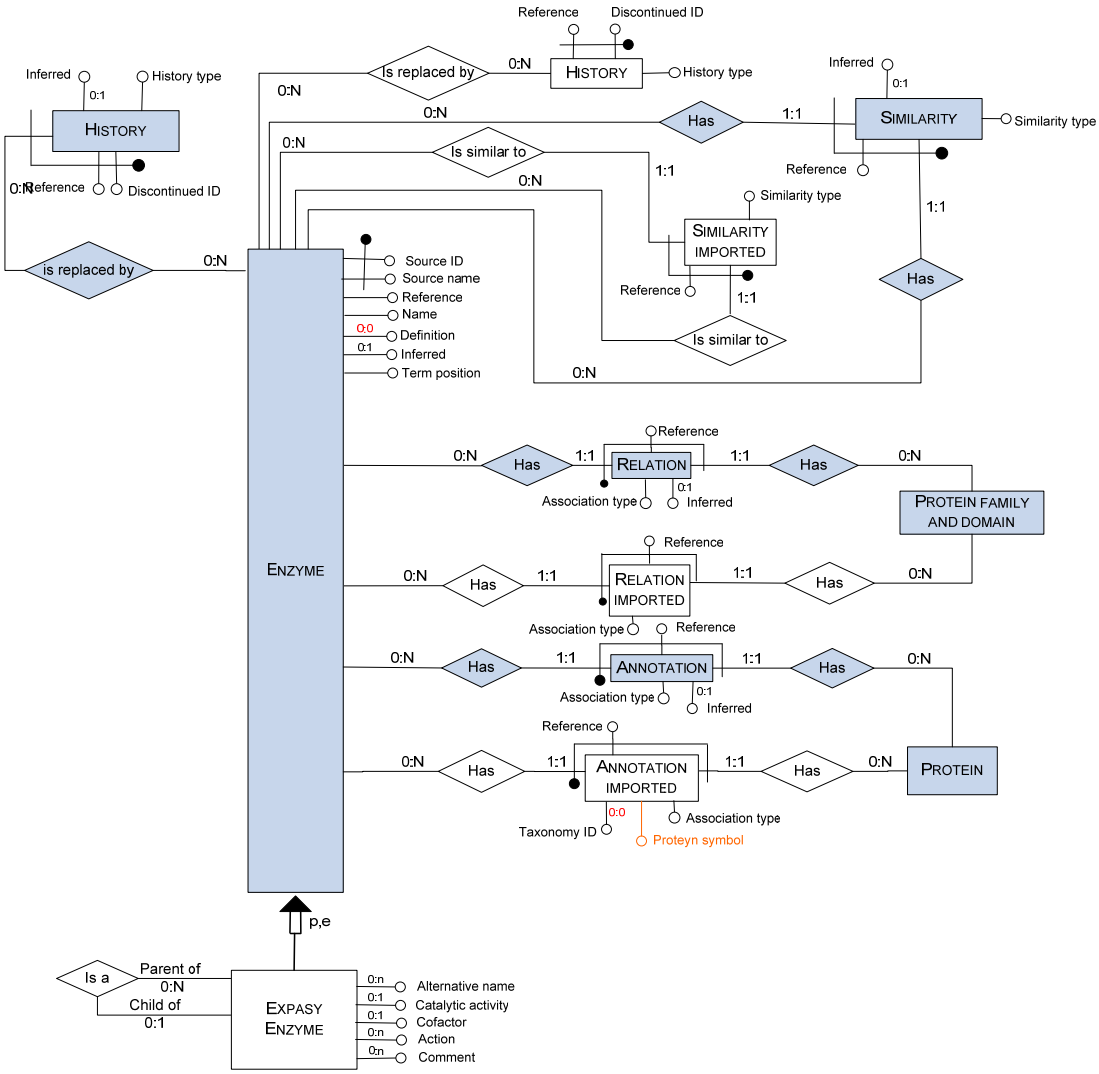

Figura 24 – Schema ER della banca dati ExPASy ENZYME

Dall'analisi è emersa la presenza di un'entità principale, *expasy_enzyme*, la quale presenta un'autorelazione di tipo gerarchico, e associazioni con le entità *protein* e *protein family and domain*. Inoltre, per *expasy_enzyme* sono forniti informazioni di similarità e di history.

8.1.2 Schema logico

Nella realizzazione dello schema logico sono state assunte delle convenzioni grafiche per poter identificare il tipo di tabella e il tipo di attributo delle tabelle stesse.
La tipizzazione delle tabelle è basata sul loro colore di sfondo come mostrato in tabella 3.

Colore tabella	Tipologia tabella
	Tabella di traduzione degli ID
	Tabella di associazione integrata
	Tabella di similarità o history integrata
	Tabella di unfolding integrata
	Tabella di autorelazione integrata
	Tabella di feature integrata
	Tabella di associazione importata
	Tabella di similarità o history importata
	Tabella di unfolding importata
	Tabella di autorelazione importata
	Tabella di feature importata

Tabella 3 – Legenda dei colori utilizzati per le tabelle degli schemi logici

Per quanto riguarda gli attributi valgono le seguenti regole.
I campi in grassetto sono campi obbligatori.
Il codice **PK** indica che il campo è una Primary Key, ovvero una chiave primaria della tabella.
Il codice **FK** indica che il campo è una Foreign Key o chiave esterna. Per chiave esterna si intende una colonna o combinazione di colonne utilizzata per stabilire e applicare un collegamento tra i dati di due tabelle.
Il codice **U1** indica che il campo appartiene a un indice univoco.
Il codice **I*n***, indica che il campo è un indice, dove *n* indica l'ordine di precedenza dell'indice. Ad esempio: I1,I2...ecc.
I campi preceduti dal carattere ***** sono codificati e i relativi valori sono memorizzati in tabelle dello schema *flag* della base di dati; i campi preceduti dal carattere **+** sono sempre campi codificati ma i relativi valori sono presenti in tabelle memorizzate nello schema *metadata*.

In figura 25 è rappresentato il diagramma logico della banca dati ExPASy ENZYME, ottenuto dalla trasformazione del diagramma ER descritto in precedenza.

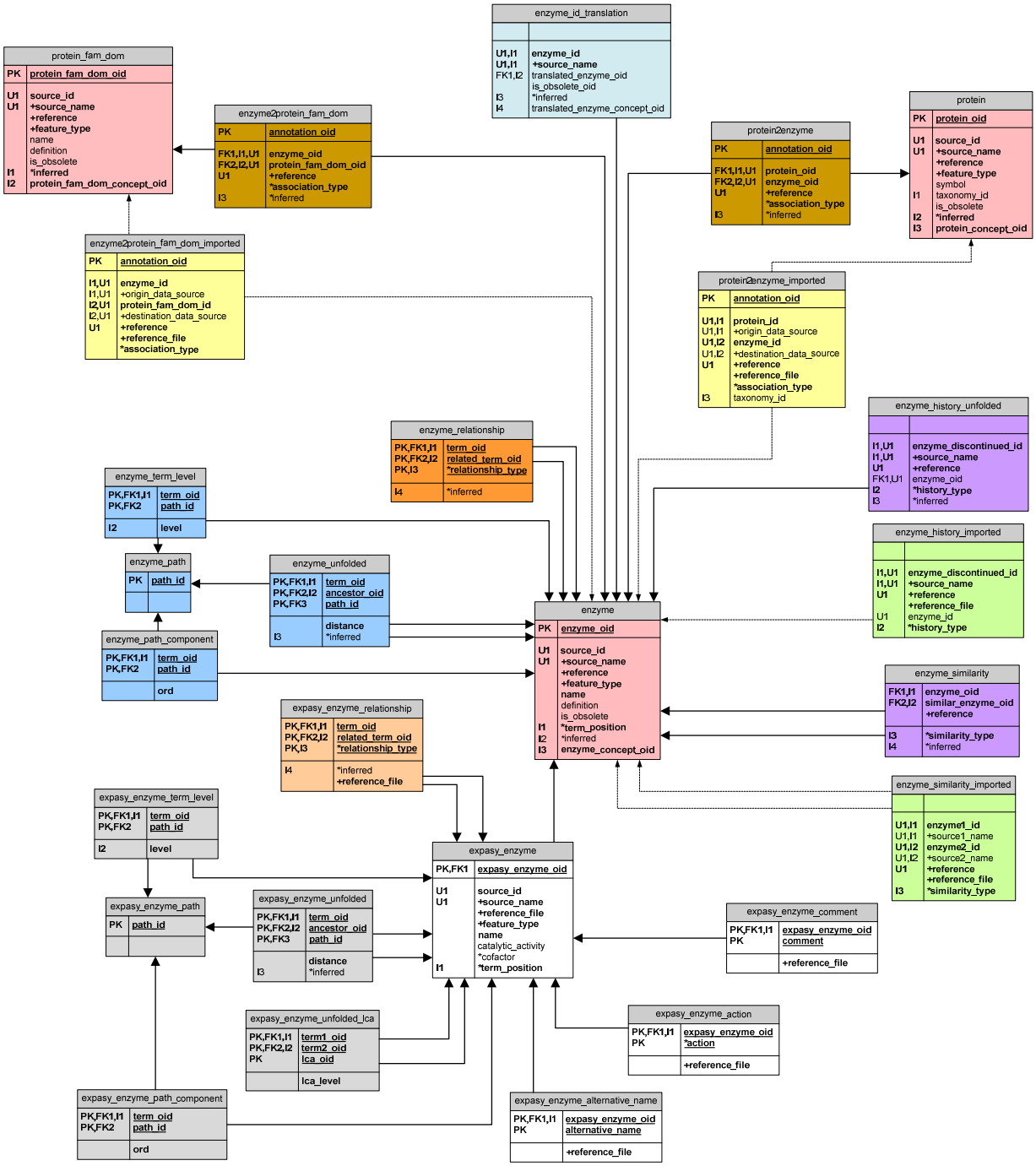

Figura 25 – Schema logico della banca dati ExPASy ENZYME

A livello importato, si può notare la presenza di una tabella principale, *expasy_enzyme*, alla quale corrisponde, a livello integrato, la tabella *enzyme*. Da tale schema risulta evidente come non tutte le informazioni importate vengano successivamente memorizzate nelle tabelle di livello integrato, ma solo quelle che in fase di progettazione del framework sono state definite "standard" per la feature in esame.

Essendo presente una relazione di tipo gerarchico, sono state inserite le tabelle atte a memorizzare la struttura di unfolding.

Le tabelle interessate dalla procedura d'importazione sono mostrate dalla figura 26 alla figura 30.

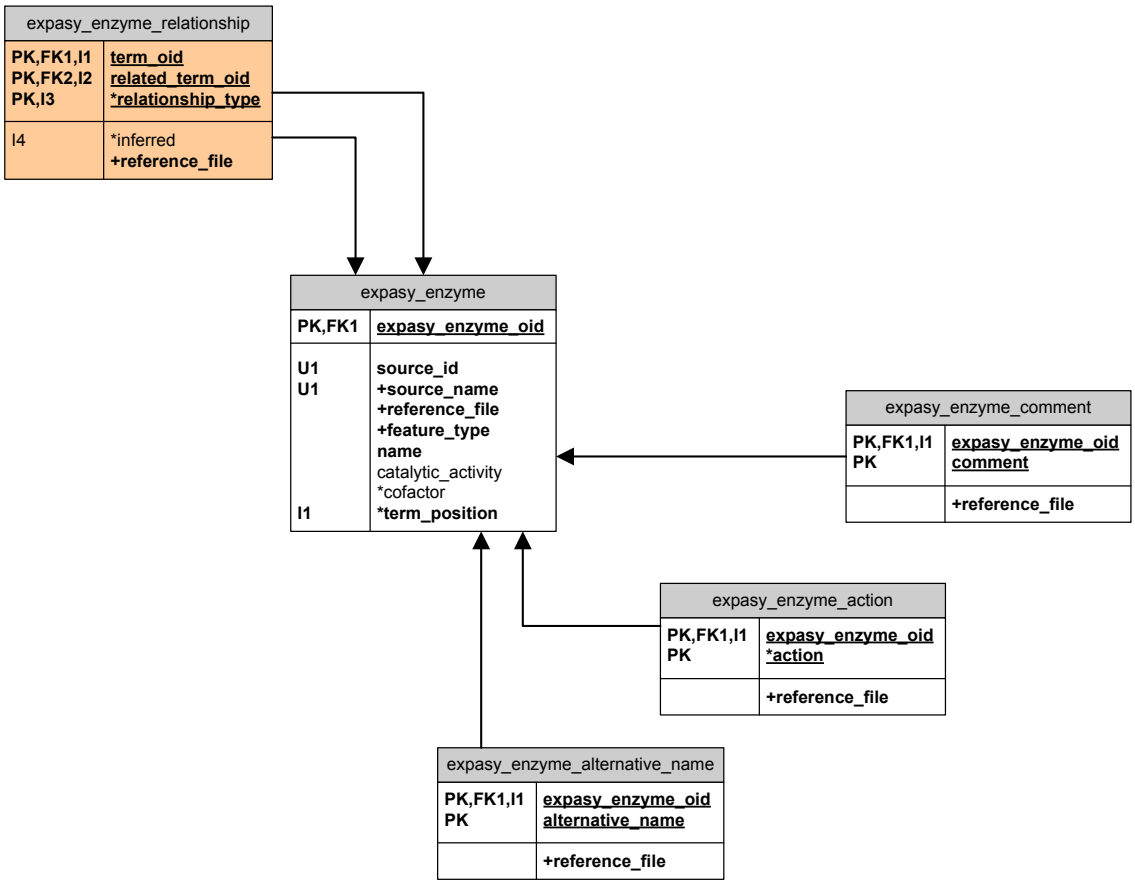

Figura 26 – Tabelle di sorgente e di relazione per gli enzimi

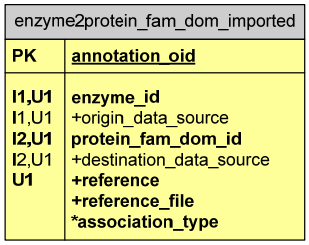

Figura 27 – Tabella di associazione enzyme2protein_fam_dom_imported

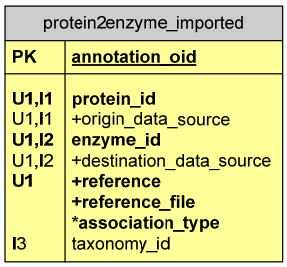

Figura 28 – Tabella di associazione protein2enzyme_imported

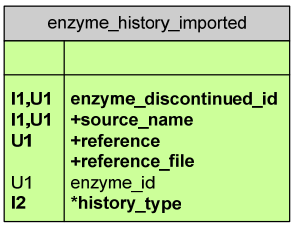

Figura 29 – Tabella di history enzyme_history_imported

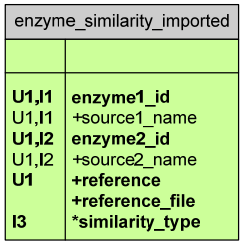

Figura 30 – Tabella di similarità enzyme_similarity_imported

8.2 Banca dati OMIM

8.2.1 Schema concettuale

Data la complessità del diagramma ER della banca dati OMIM non è possibile riportare lo schema completo ma solo le parti maggiormente significative.
Le figure 31 e 32 mostrano le entità principali contenute nel diagramma concettuale della banca dati OMIM, ottenuto dall'analisi dei file *omim.txt, genemap, morbidmap* e *pubmed_cited*.

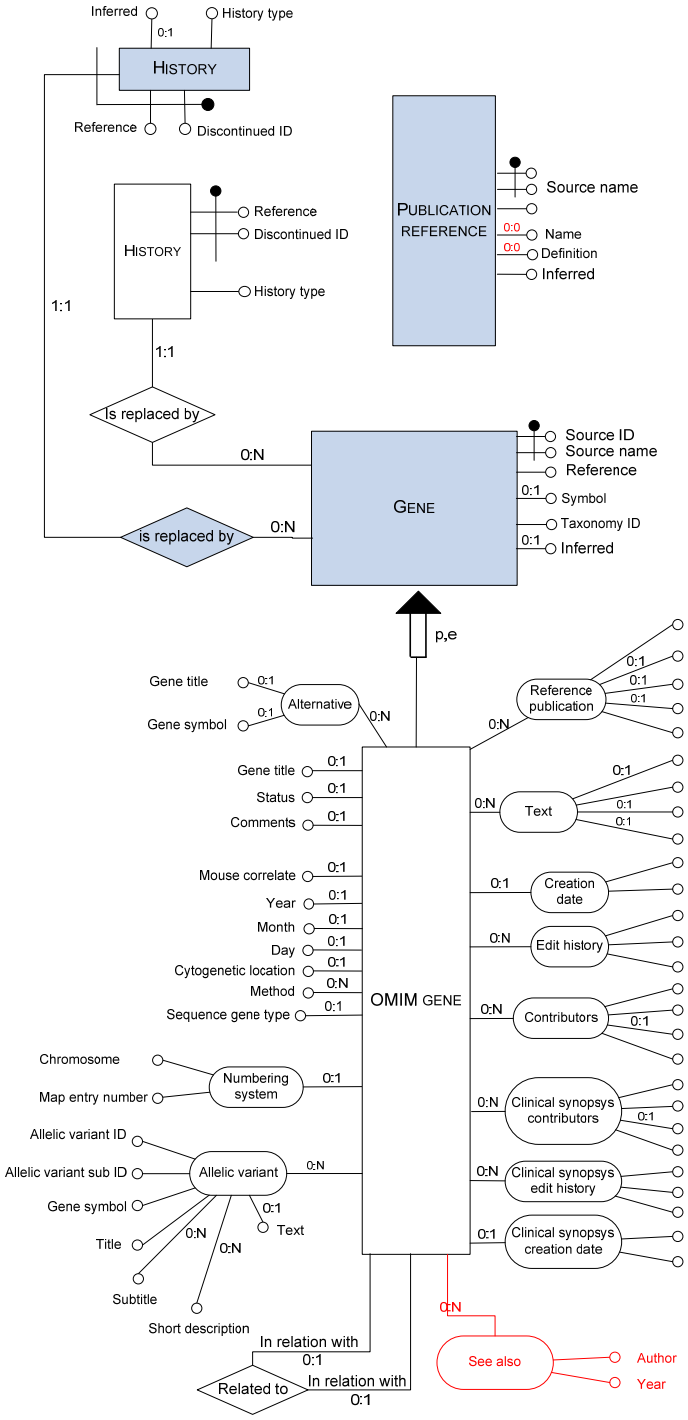

Figura 31 – Schema ER della banca dati OMIM (prima parte)

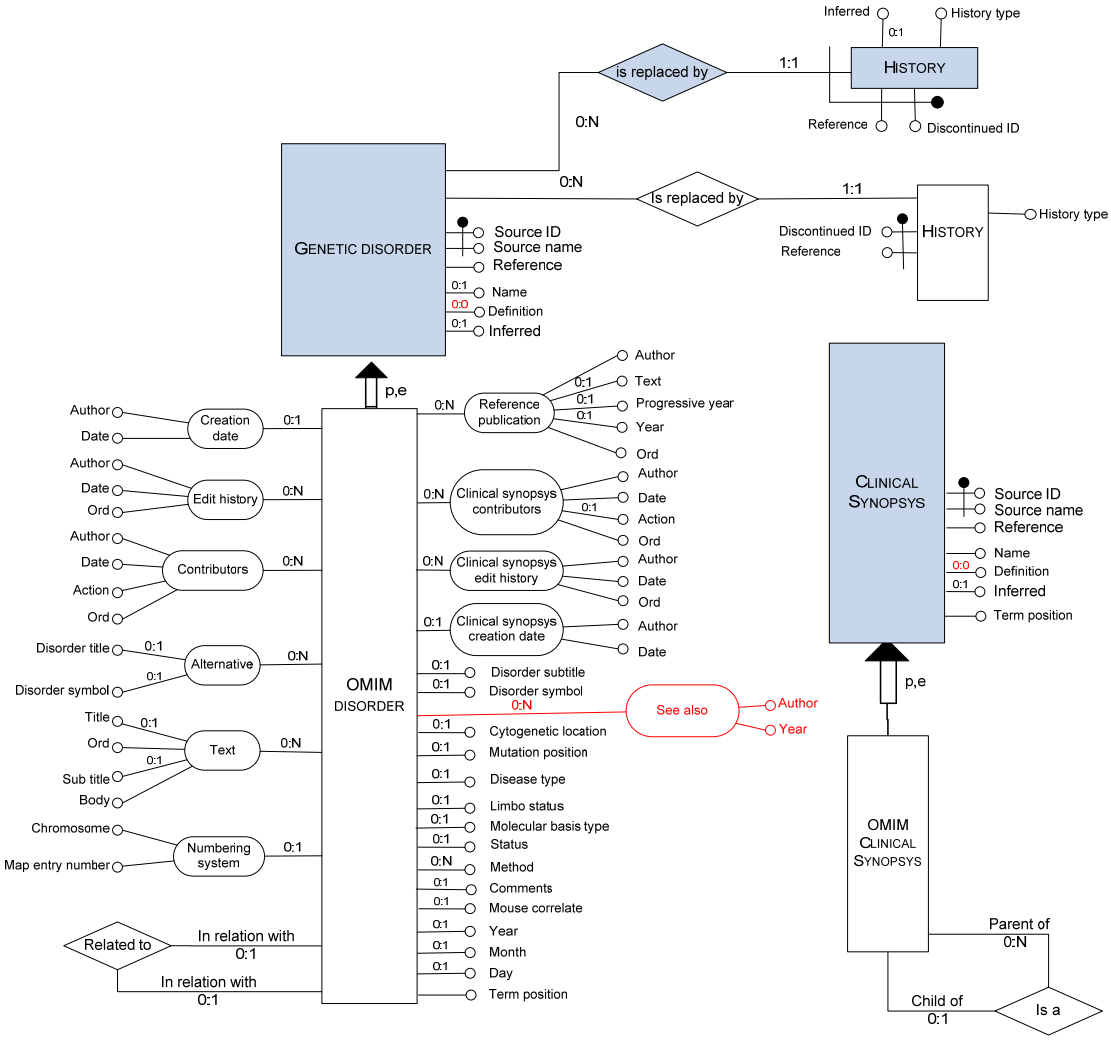

Figura 32 – Schema ER della banca dati OMIM (seconda parte)

Dall'analisi è emersa la presenza di due entità principali *omim_gene* e *omim_disorder*, ciascuna coinvolta in autorelazioni ma anche in relazioni che legano l'una con l'altra. Per tali entità sono fornite informazioni di history e riferimenti a pubblicazioni scientifiche (file *pubmed*). Inoltre vi è la presenza di un'altra entità, meno complessa, *omim_clinical_synopsys* per la quale è possibile definire un'autorelazione di tipo gerarchico e un legame con le due entità principali.

8.2.2 Schema logico

Data la complessità dello schema logico scaturito dalla trasformazione del diagramma ER della banca dati OMIM, dalla figura 33 alla figura 42 sono mostrate le solo tabelle interessate dalla procedura d'importazione.

Il campo *reference_file,* presente nelle tabelle di livello importato, risulta fondamentale per la corretta importazione dei dati forniti da questa sorgente. Le informazioni relative ciascun identificativo di OMIM non sono racchiuse in un unico file, ma sono presenti nei diversi file che la sorgente fornisce. È evidente come sia necessario tenere traccia non solo della banca dati che ha fornito le informazioni ma anche del file dal quale queste sono state prelevate.

La tabella *omim_clinical_synopsys_sub_group,* rappresentata in figura 35, è stata creata per suddividere le localizzazioni dei fenotipi in gruppi. Tale aspetto sarà maggiormente approfondito nel paragrafo 9.2.6.

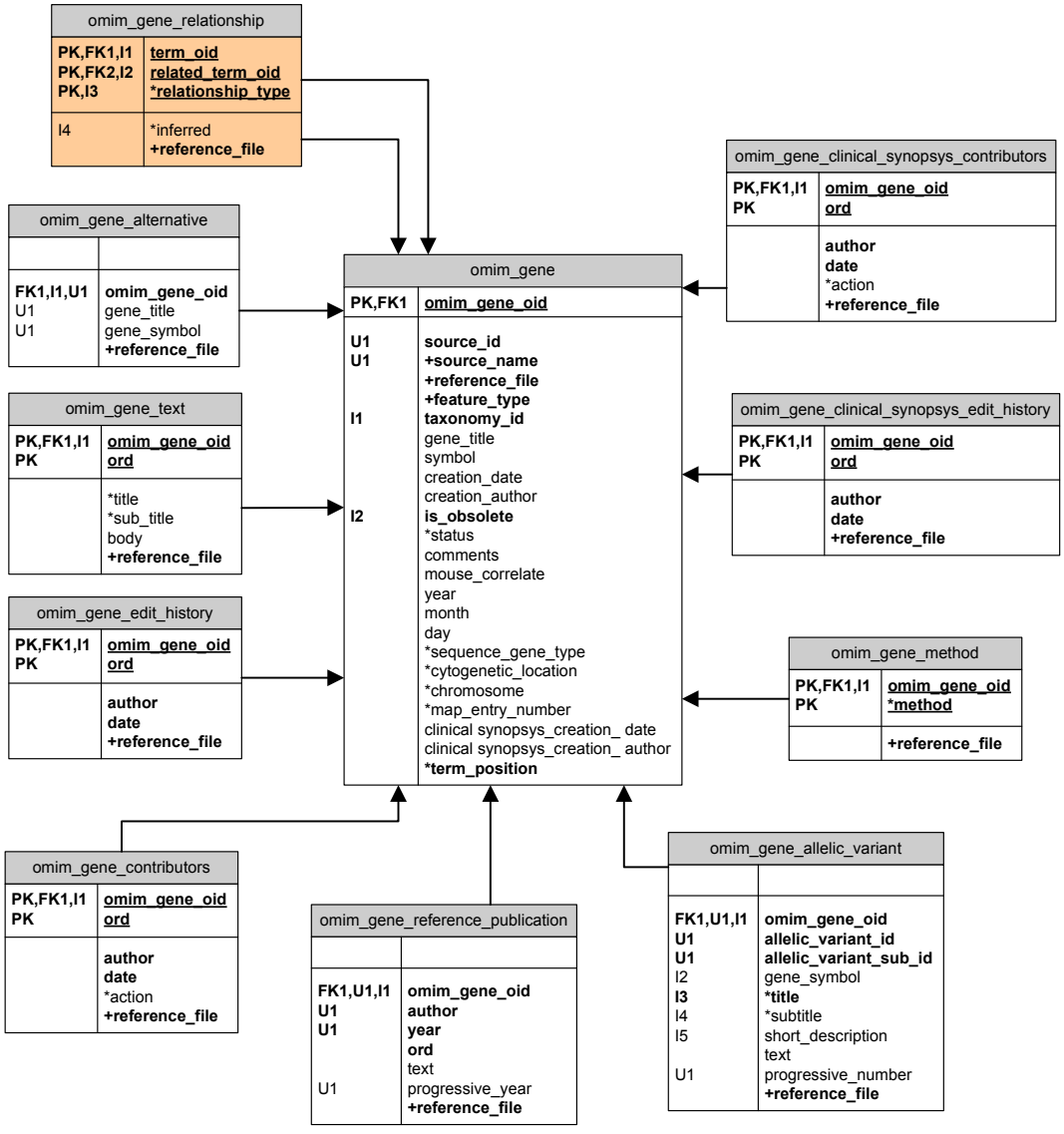

Figura 33 – Tabelle di sorgente e di relazione per i geni

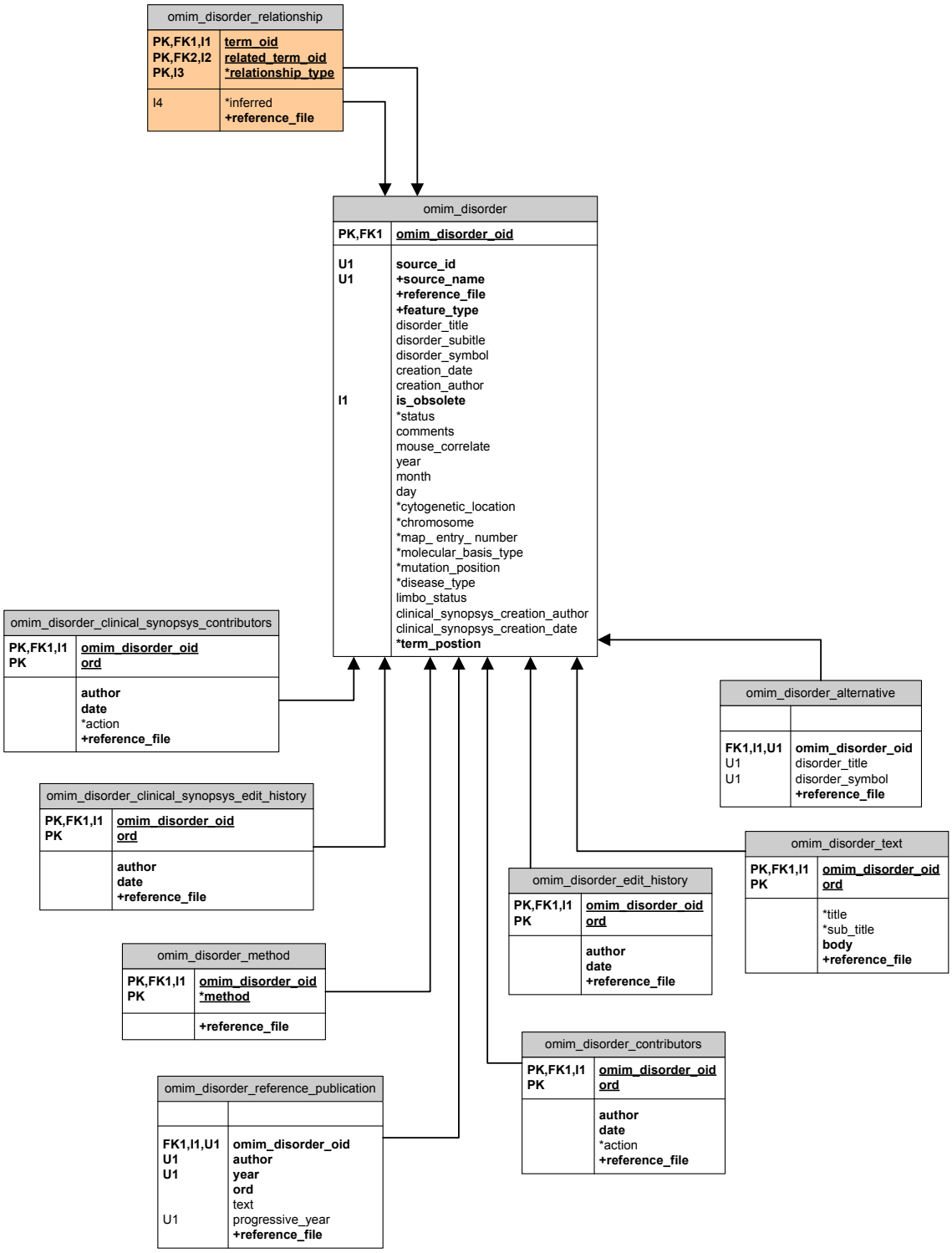

Figura 34 – Tabelle di sorgente e di relazione per i fenotipi

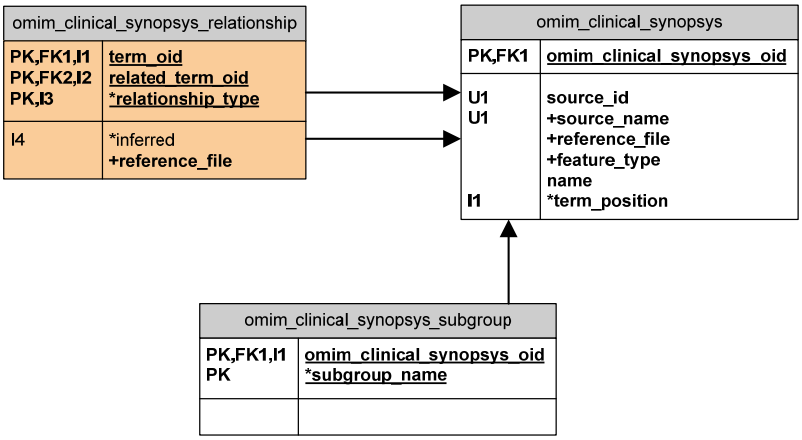

Figura 35 – Tabelle di sorgente e di relazione per clinical synopsys

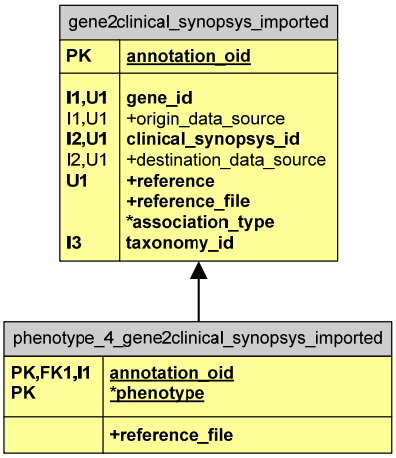

Figura 36 – Tabelle di associazione gene2clinical_synopsys_imported

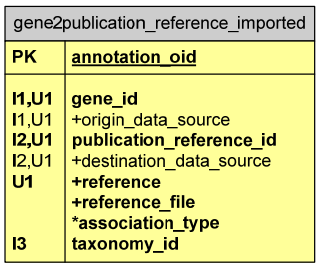

Figura 37 – Tabelle di associazione gene2publication_reference_imported

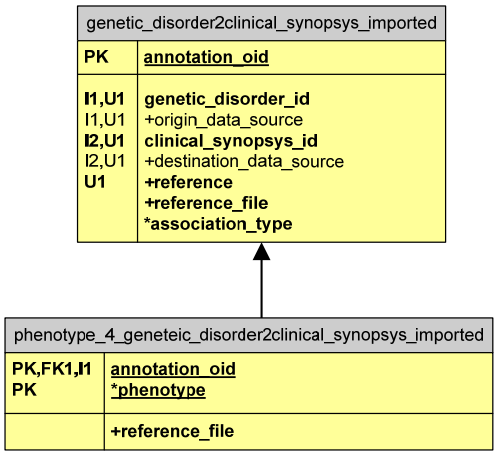

Figura 38 – Tabelle di associazione genetic_disorder2clinical_synopsys_imported

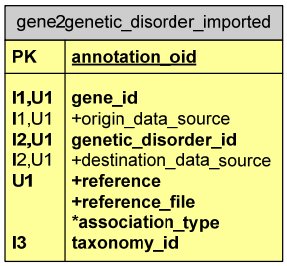

Figura 39 – Tabella di associazione gene2genetic_disorder_imported

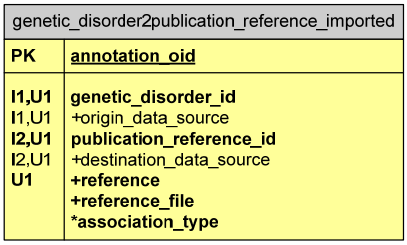

Figura 40 – Tabella di associazione genetic_disorder2publication__imported

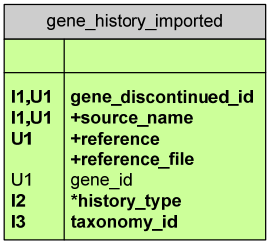

Figura 41 – Tabella di history gene_history_imported

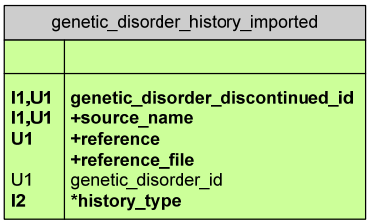

Figura 42 – Tabelle di history disorder_history_imported

9 Implementazione delle procedure automatiche d'importazione dati

In questa sezione sono illustrate le procedure realizzate e le scelte progettuali adottate per l'importazione dei dati forniti dalle sorgenti descritte nei capitoli precedebti, motivando le soluzioni tecniche utilizzate.
Per un maggior approfondimento, in appendice è mostrato il mapping tra i campi estratti dai file, descritti nel seguito del presente capitolo, e i campi delle tabelle del data warehosue.

9.1 Banca dati ExPASy ENZYME

La banca dati ExPASy ENZYME è raggiungibile al sito http://www.expasy.ch/enzyme/.
ExPASy ENZYME fornisce due tipi di file di testo:

- il file *enzclass.txt* di tipo tabellare con header di lunghezza fissa;
- il file *enzyme.dat* di tipo flat con header.

I Parser disponibili non permettono il processamento di file con header di lunghezza fissa; prima di procedere con l'importazione dei file forniti da ExPASy ENZYME ho, quindi, implementato il Parser generico per i file tabellari con header di lunghezza fissa; è stata realizzata una nuova classe *TabularFileWithFixLengthHeaderParser.java* che estende il Parser *TabularFileParser.java,* implementato in precedenza, eseguendo l'override del metodo *processLine*.
Il Parser riceve in input i seguenti parametri necessari alla scansione del contenuto del file:

- separatore di riga: carattere o sequenza di caratteri che indicano la fine di una riga di testo;
- separatore di campo: carattere o sequenza di caratteri che identifica la fine di un campo;
- numero di campi per riga;
- parametri di header: lista di stringhe che identifica le singole informazioni di header;
- lunghezza dell'header: numero di righe che compongono l'header;
- separatore di header: carattere o sequenza di caratteri usata per separare l'header dal resto del testo.

Il file viene letto una riga alla volta, grazie alla suddivisione fornita dal separatore di linea. Se confrontando la linea di testo estratta con il separatore di header, viene identificato l'inizio dell' header vengono passate al metodo *headerControl* le successive *m* righe, dove *m* indica la lunghezza dell'header passata come parametro. Tale metodo verifica che si tratti realmente di linee di header confrontandole con i parametri di header. In caso di esito

negativo del confronto viene segnalato l'errore, altrimenti il contenuto dell'header può essere gestito implementando l'override del metodo *onHeaderRead*.
Se, invece, non si tratta di una linea di header viene parsata con le stesse modalità dei Parser di tipo tabellare implementati in precedenza, separando i vari elementi tramite il separatore di campo.
È stato, inoltre, necessario realizzare il Parser generico per i file di tipo flat, in quanto nel framework non erano presenti classi per parsarli.
Ho implementato tre nuove classi:

- *it\polimi\gfinder2\importer\generic\FlatFileParser.java* utilizzata per parsare flat file senza header.
  Il Parser riceve in input i parametri necessari alla scansione del contenuto del file:
  - separatore di record: carattere o sequenza di caratteri che identifica un raggruppamento di campi appartenenti allo stesso elemento (record);
  - separatore di riga: carattere o sequenza di caratteri che indica la fine di una riga di testo;
  - etichette di linea: lista di stringhe contenente le etichette utilizzate per identificare i dati contenuti in ogni riga di testo.

  Il file viene letto una riga alla volta fino alla ricostruzione completa di un record, il quale viene successivamente analizzato dal metodo *processRecord*. Tale metodo confronta l'inizio di ogni linea del record con le etichette di linea in modo da identificare il tipo d'informazione, in caso non sia riconosciuta nessuna etichetta viene considerata quella della riga precedente. Il metodo *processRecord* restituisce ciascun record analizzato sotto forma di lista di *FlatFileLine*. Un oggetto di tipo *FlatFileLine* è composto da un'etichetta, che identifica il tipo dell'informazione, e da un valore.
- *it\polimi\gfinder2\importer\generic\FlatFileWithHeaderParser.java* utilizzata per parsare flat file con header, estende il Parser generico *it\polimi\gfinder2\importer\generic\FlatFileParser.java*.
  La gestione dell'header avviene con le stesse modalità con cui è gestito dal Loader tabellare preesistente nel framework . Il resto del file è gestito come descritto al punto precedente.
- *it\polimi\gfinder2\importer\generic\FlatFileWithFixLengthHeaderParser.java java* utilizzata per parsare flat file con intestazione di lunghezza fissa, estende il Parser generico *it\polimi\gfinder2\importer\generic\FlatFileParser.java*.
  La gestione dell'header avviene con le stesse modalità con cui è gestito dal Loader tabellare con header di lunghezza fissa descritto in questo capitolo. Il resto del file è gestito utilizzando i metodi ereditati dalla classe *FlatFileParser.java*.

Si è deciso di importare come prima sorgente il file *enzclass.txt* poiché contiene una suddivisione di tipo gerarchico delle tipologie di enzimi, utilizzata per strutturare i record presenti nel file *enzyme.dat*.

9.1.1 File enzclass.txt

Il file è di tipo tabellare con singolo separatore di campo e header di lunghezza fissa.
Il Loader che importa il file è *EnzymeClassLoader.java*, presente all'interno del package *gfinder2.importer.expasyenzyme*, ed estende il parser *TabularFileWithFixLengthHeaderParser.java.*
L'header contiene informazioni relative la versione e l'autore del file e altri dati che non sono considerati d'interesse per l'importazione; il blocco di header è identificato da una riga di caratteri *"-"*.
I due campi principali, classe dell'enzima e nome della classe, sono separati da una sequenza composta da due spazi bianchi**.**
In figura 43 è riportata la struttura del file.

```
----------------------------------------------------------------------
--
ENZYME nomenclature database
Swiss Institute of bioinformatics (SIB). Geneva, Switzerland
----------------------------------------------------------------------
--
Description:  Definition of enzymes classes, subclasses and sub-
Subclasses
Name:         ENZCLASS.TXT
Release of:   15-Jun-2010

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

1. -. -.-  Oxidoreductases.
1. 1. -.-   Acting on the CH-OH group of donors.
1. 1. 1.-    With NAD(+)or NADP(+) as acceptor.
1. 1. 2.-    With a cytochrome as acceptor.
```

Figura 43 – Esempio di struttura del file enzclass.txt

Come si può facilmente notare, le prime dieci righe rappresentano l'header, esse saranno dapprima confrontate con i parametri di header tramite il metodo *headerControl* e successivamente, grazie all'override del metodo *onHeaderRead,* non saranno considerate in fase di importazione.
Dalle righe successive si estraggono i campi che costituiscono il dato da integrare nel database.
Il Parser riceve in input la prima riga utile del file, per utile s'intende la prima riga non di header. La riga viene processata utilizzando il separatore di campo " ", ricavando una struttura a due campi, identificata dai seguenti valori:

- *EC number: 1. -. -.- ;*
- *Name: Oxidoreductases.*

Il file *enzclass.txt* fornisce una gerarchizzazione degli enzimi tramite struttura ad albero; ad esempio la sottoclasse di enzymi *"1. 1. -. -"* sarà figlia del nodo radice *"1. -. -. -"*.

Tramite l'utilizzo dei metodi per la gestione di stringhe si eseguono la pulizia e la formattazione del valore recuperato, dal quale si estrapola solo il valore d'interesse; viene scartata l'ultima parte di *"1. -. -.-"*, composta dalla sequenza ripetuta di caratteri ". -", e viene considerata solo la sottostringa "*Oxidoreductases*" del campo 2.
Inoltre la classe di enzimi estratta per essere effettivamente importata deve essere validata dal confronto con la seguente espressione regolare: *([1-6])([.]([1-9]|[1-9][0-9])){0,2}*.
Dal parsing di questo file si popolano le seguenti tabelle:

- *expasy_enzyme;*
- *expasy_enzyme_relationship.*

9.1.2 File enzyme.dat

Il file è di tipo flat con header di lunghezza fissa.
Il Loader che importa il file è costituito dalla classe *EnzymeDataLoader.java*, presente nel package *gfinder2.importer.expasyenzyme,* ed estende il Parser *FlatFileWithFixLengthHeaderParser.*
L'header contiene le informazioni riguardanti la versione e l'autore del file e altri dati che non sono considerati d'interesse per l'importazione. L'inizio e la fine del blocco di header sono identificati da una riga di caratteri *"-"*, mentre ciascuna linea di header è identificata dall'etichetta *"CC"*.
La struttura indicata in figura 44 riporta un tipico record con tutti i possibili campi che lo compongono.

```
//
ID     1.1.1.1
DE    Alcohol dehydrogenase.
AN    Aldehyde reductase.
CA    An alcohol + NAD(+) = an aldehyde or ketone + NADH .
CF     Iron or zinc.
CC     -!- Acts on primary or secondary alcohols or hemiacetals.
PR     PROSITE; PDOC00058;
DR    P07327, ADH1A_HUMAN;
```

Figura 44 – Esempio di struttura del file enzyme.dat

I record sono identificati dalla sequenza di caratteri *"//"* mentre ciascuna linea è identificata da un'etichetta di linea composta da due caratteri. La tabella 4 mostra le possibili etichette di linea presenti nel file (per cardinalità valore per record si intende il numero di valori estratti, non il numero di valori contenuti nel testo).

Identificativo campo	Nome campo	Cardinalità flags per record	Cardinalità valori per record
ID	Identification	1:1	1:1
DE	Description	1:n	1:1
AN	Alternate Name	0:n	0:1
CA	Catalytic Activity	1:n	0:1
CF	Cofactor	0:n	0:1

CC	Comment	0:n	0:n
PR	Prosite Reference	0:n	0:n
DR	Database Reference	0:n	0:n

Tabella 4 – Tipi di campo presenti nel file enzyme.dat

Di seguito si fornisce una descrizione dettagliata dei dati proposti da ogni campo corredata da esempi concreti.

- *ID*

```
ID 1.1.1.1
```

Figura 45 – Esempio di campo ID file enzyme.dat

Dal campo si estrae il valore *"1.1.1.1"* che rappresenta l'EC number associato all'enzima. Tale valore deve essere validato tramite l'espressione regolare *([1-6])([.]([1-9]|[1-9]([0-9]+)?)){0,3}*.

- *DE*

```
DE  Alcohol dehydrogenase.
```

Figura 46 – Esempio di campo DE file enzyme.dat

Campo di tipo obbligatorio, fornisce il nome raccomandato dalla NC-IUB (Nomenclature Committee of the International Union of Biochemistry) utilizzato per identificare l'enzima: *"Alcohol dehydrogenase"*. Il valore può estendersi su più righe di testo e la terminazione è indicata dal carattere *"."*.
Alcuni enzimi possono essere cancellati dalla EC list, altri sono rinumerati. I record contenenti enzimi obsoleti sono distinti dalla presenza di una delle seguenti linee di testo:

- per gli enzimi eliminati dalla lista:

```
DE    Deleted entry.
```

Figura 47 – Esempio di campo DE di enzimi eliminati dalla EC list

- per gli enzimi rinumerati:

```
DE    Transferred entry: x.x.x.x.
```

Figura 48 – Esempio di campo DE di enzimi rinumerati

Dove *"x.x.x.x"* è il nuovo EC number, da validare tramite l'espressione regolare *([1-6])([.]([1-9]|[1-9]([0-9]+)?)){0,3}*.

- *AN*

```
AN Aldehyde reductase.
```

Figura 49 – Esempio di campo AN file enzyme.dat

Campo di tipo opzionale, fornisce il nome utilizzato in letteratura per descrivere l'enzima: *"Aldehyde reductase"*. Il valore può estendersi su più righe di testo e la terminazione è indicata dal carattere *"."*.

- *CA*

```
CA An alcohol + NAD(+) = an aldehyde or ketone + NADH.
```

Figura 50 – Esempio di campo CA file enzyme.dat

Il campo *CA* può non essere presente in un record, descrive le reazioni catalizzate dall'enzima secondo le indicazioni fornita dalla NC-IUB. Il valore estratto dalla riga mostrata nell'esempio è: *"An alcohol + NAD(+) = an aldehyde or ketone + NADH"*.
La maggior parte delle reazioni sono fornite secondo il formato *"Substrate_11 + Substrate_12 [+ Substrate_1N...] = Substrate_21 + Substrate_22 [+ Substrate_2N]"*, ma possono contenere anche del testo libero. In entrambi i casi, il campo è terminato dal carattere *"."*.

- *CF*

```
CF Cobalt; NAD.
```

Figura 51 – Esempio di campo CF file enzyme.dat

Campo di tipo opzionale riporta la lista dei cofattori richiesti dall'enzima nel formato "*Cofactor_1; Cofactor_2 or Cofactor_3[; Cofactor_N...]"* o come testo libero terminato dal carattere *"."*.
Si è scelto di salvare nel database l'intera stringa contenente l'elenco dei cofattori e non i singoli cofattori poiché la presenza del testo libero non lo consentiva.
Il valore estratto dal caso in esempio è: *"Cobalt; NAD"*.

- *CC*

```
CC -!- Acts on primary or secondary alcohols or hemiacetals.
```

Figura 52 – Esempio di campo CC file enzyme.dat

Il campo *CC* è opzionale, contiene testo libero con informazioni utili riguardanti l'enzima.
La sequenza di caratteri *"-!-"* identifica l'inizio di una nuova stringa di commento.
Da questo campo si estraggono altre tipologie d'informazioni:

- Similarità

```
CC -!- May be identical with EC 1.1.1.19, EC 1.1.1.33 and EC
1.1.1.55.
CC -!- May be the same as EC 1.2.3.1.
```

Figura 53 – Esempio di campo CC contenente informazioni di similarità

In figura 53 sono mostrati due esempi di stringhe di commento contenenti informazioni di similarità tra enzimi di tipo *"ALIAS"*. Le parole chiave utilizzate per identificarle sono:

- *identical with EC*
- *the same as EC*

seguite da una lista di EC number. Ogni EC number appartenente alla lista viene validato attraverso la stessa espressione regolare utilizzata per validare il valore estratto dal campo *ID*.

- Azioni

```
CC -!- Inactivates bradykinin and anaphylatoxins in blood
plasma.
CC -!- Activates trypsinogen.
CC -!- It also catalyzes the hydrolysis of diphosphate to
form
two equivalents of phosphate.
CC -!- Also hydrolyzes other 4-substituted henylacetonitriles.
CC -!- Also reduces D-galacturonate.
CC -!- Acetylene is reduced to ethylene
CC -!- Also oxidizes L-histidinal.
```

Figura 54 – Esempio di campo CC contenente informazioni riguardanti azioni

La figura 54 mostra alcuni esempi di campi di tipo *CC* contenenti azioni svolte dagli enzimi. Le azioni che individuate nel file *enzclass.dat* sono:

- *inactivates;*
- *activates;*
- *catalyzes;*
- *hydrolyzes;*
- *reduces;*
- *is reduced;*
- *oxidizes.*

- History

```
CC -!- Formerly EC 1.1.1.33 and EC 1.1.1.55.
```

Figura 55 – Esempio di campo CC contenente informazioni di history

In figura 55 è mostrato un esempio di stringa di commento contenente informazioni di history. La parola chiave utilizzata per identificarla è *"Formerly"*, seguita dalla lista degli EC number con il quale era identificato l'enzima in precedenza.

- *PR*

```
PR PROSITE; PDOC00370
```

Figura 56 – Esempio di campo PR file enzyme.dat

Campo di tipo opzionale, riporta i riferimenti verso i documenti PROSITE che menzionano l'enzima descritto nel record. Il formato con cui viene presentata questa informazione è *"PR PROSITE; PSITE_Doc_ACNb",* dove *PSITE_Doc_ACNb* rappresenta un entry access number verso un documento di PROSITE, che fornisce informazioni relative a famiglie di proteine.
I valori estratti devono essere validati attraverso le espressioni regolari:

- *[P][D][O][C][0-9][0-9][0-9][0-9][0-9]*
- *[P][S][0-9][0-9][0-9][0-9][0-9]*

- *DR*

```
DR Q04409, EMI2_YEAST ;  Q9GTW9, GLK1_TRIVA ;  Q9GTW8, GLK2_TRIVA ;
```

Figura 57 – Esempio di campo DR file enzyme.dat

Il campo *DR* può non essere presente, riporta i riferimenti verso le entry di UniProtKB/Swiss-Prot[21] relazionate con l'enzima descritto nel record. Il formato con cui i riferimenti vengono rappresentati è *"DR AC_Nb, Entry_name; AC_Nb, Entry_name; AC_Nb, Entry_name;"*, dove *AC_Nb* è il numero di accesso primario all'entry di UniProtKB/Swiss-Prot e *Entry_name* è il nome dell'entry.
Il valore estratto di nostro interesse è il numero di accesso primario *AC_Nb* il quale deve essere validato attraverso le espressioni regolari:

- *[A-NR-Z][0-9][A-Z][A-Z0-9][A-Z0-9][0-9]*
- *[OPQ][0-9][A-Z0-9][A-Z0-9][A-Z0-9][0-9]*

Dal parsing di questo file si popolano le seguenti tabelle:

- *expasy_enzyme;*
- *expasy_enzyme_alternative_name;*
- *expasy_enzyme_action;*
- *expasy_enzyme_comment;*
- *expasy_enzyme_relationship;*
- *enzyme_history_imported;*
- *enzyme_similarity_imported;*
- *enzyme2prot_fam_dom_imported;*
- *protein2enzyme_imported.*

9.2 Banca dati OMIM

La banca dati OMIM è raggiungibile al sito http://www.ncbi.nlm.nih.gov/sites/entrez?db=omim.
OMIM fornisce tre tipi di file di testo:

- i file *genemap*, *morbidad* e *pubemed_cited* di tipo tabellare senza header;
- il file *omim.txt* di tipo flat senza header;
- il file *genemap.key* di tipo flat con header di lunghezza fissa.

Per processare i file sono stati utilizzati sia i Parser già disponibili all'interno del framework che i Parser per flat file che ho realizzato, descritti nel capitolo precedente.
È stato necessario importare inizialmente i file *omim.txt* e *genemap.key* in quanto contengono rispettivamente le informazioni necessarie ad identificare la natura delle entries descritte in OMIM e i valori e la descrizione dei codici utilizzati nei file *genemap* e *morbidmap*.

9.2.1 File omim.txt

Il file *omim.txt* contiene l'intero testo libero del database OMIM. La struttura indicata in figura 58 riporta un tipico record con tutti i possibili campi che lo compongono.

```
*RECORD*
*FIELD* NO
275350
*FIELD* TI
+275350 TRANSCOBALAMIN II DEFICIENCY
;;TC II DEFICIENCY;;
TCN2 DEFICIENCY
...

*FIELD* TX
DESCRIPTION

Transcobalamin II (TC II), a plasma globulin, is the primary
transport
protein for vitamin B12 (Hakami et al., 1971). Genetic absence
leads to
...

*FIELD* AV
.0001
TRANSCOBALAMIN II DEFICIENCY
TCN2, 4-BP DEL

In the compound heterozygote identified by Li et al. (1994), the
...

*FIELD* SA
...

*FIELD* RF
1. Acklin, M.; Frater-Schroder, M.; Haller, O.; Lundin, L. G.;
Prochazka,
M.; Skow, L. C.: Localization of transcobalamin II (Tcn-2) on
chromosome
...

*FIELD* CS
INHERITANCE:
   Autosomal recessive
...

*FIELD* CN
Cassandra L. Kniffin - updated: 12/14/2007
Ada Hamosh - reorganized: 11/10/2004
...
```

```
*FIELD* CD
Kelly A. Przylepa - revised: 03/06/2002

*FIELD* ED
joanna: 03/06/2002
...

*FIELD* CN
Marla J. F. O'Neill - updated: 7/11/2007
Carol A. Bocchini - updated: 4/14/2006
...

*FIELD* CD
Victor A. McKusick: 6/4/1986

*FIELD* ED
alopez: 04/07/2009
```

Figura 58 – Esempio di struttura del file omim.txt

Il Loader che importa il file è *OmimLoader.java*, presente all'interno del package *gfinder2.importer.omi,* ed estende il Parser *FlatFileParser*.
I record sono identificati dalla parola chiave **RECORD** mentre i campi sono individuato dalla parola chiave **FIELD* XX*, dove con *XX* si intende il codice identificativo di ogni campo; i valori possibili sono indicati nella tabella 5. Ogni elemento *FIELD* può essere costituito a sua volta da ulteriori sottocampi, questa suddivisione non avviene con un codice comune a tutti i campi ma ognuno deve essere gestito autonomamente. Il numero di sottocampi varia per tipo di FIELD e non tutti i FIELD sono sempre presenti all'interno di un record.
La fine del file è indicata dalla parola chiave **THE END**.

Identificativo campo	Nomi campo	Presenza del campo nel record
FIELD NO	Mim number	Testo sempre presente
FIELD TI	Titolo	Testo sempre presente
FIELD TX	Text	Testo sempre presente
FIELD AV	Allelic Variants	Presenza opzionale ma solo per record di geni Assente nei record di fenotipi
FIELD SA	See also	Testo opzionale
FIELD RF	References	Testo opzionale

FIELD CN	Contributors	Testo opzionale, possibilità di presenza doppia
FIELD CD	Creation Date	Testo sempre presente, possibilità di presenza doppia
FIELD ED	Edit History	Testo opzionale, possibilità di presenza doppia
FIELD CS	Clinical Synopsis	Testo opzionale

Tabella 5 – Tipi di campo presenti nel file omim.txt

Di seguito si fornisce una descrizione dettagliata delle tipologie d'informazioni proposte da ogni campo, corredata da esempi concreti.

- **FIELD* NO*

```
*FIELD* NO
275630
```

Figura 59 – Esempio di campo *FIELD* NO file omim.txt

Dal campo **FIELD* NO* si estrae il seguente valore:

- *Mim number: 275630*

  Rappresenta il Mim number, l'identificativo univoco assegnato da OMIM all'elemento che si sta importando, può rappresentare un gene oppure un fenotipo (nel seguito userò indistintamente i termini fenotipo e genetic disorder per indicare la stessa entità biomolecolare). Il Mim number deve essere validato attraverso l'espressione regolare *[0-9]{6}*.

- **FIELD* TI*

```
#275630 CHANARIN-DORFMAN SYNDROME; CDS
;;NEUTRAL LIPID STORAGE DISEASE WITH ICHTHYOSIS; NLSDI;;
TRIGLYCERIDE STORAGE DISEASE WITH IMPAIRED LONG-CHAIN FATTY ACID
OXIDATION;;
ICHTHYOTIC NEUTRAL LIPID STORAGE DISEASE;;
```

Figura 60 – Esempio di campo *FIELD* TI file omim.txt

Il campo **FIELD* TI* fornisce i seguenti sottocampi, che esprimono significati diversi in base alla riga cui appartengono:

Prima riga:

- *Tipo record: #*

  Il carattere "#" posto davanti al codice numerico identifica il tipo record, tale simbolo può assumere i seguenti valori:

  - * : gene con sequenza nota;
  - + : gene con sequenza e fenotipo noti;
  - # : fenotipo con basi molecolari note;

- % : fenotipo mendeliano con base molecolare sconosciuta;
- se non è indicato nessun simbolo ma semplicemente il codice numerico si tratta di fenotipi con sospetta base mendeliana;
- ^ : record storicizzato, tale record è stato sostituito da un altro Mim number oppure rimosso dal database.

▪ *Mim number: 275630*

Identificativo univoco già estratto dal campo precedente **FIELD NO**, questo valore viene trascurato.

Il testo che segue il Mim number può essere suddiviso in sottocampi dal carattere *";"* se presente, i campi che ne derivano sono il titolo e il simbolo della feature:

▪ *Titolo: CHANARIN-DORFMAN SYNDROME*

Nel caso l'elemento sia un record storicizzato, al posto del titolo è presente l'operazione eseguita sul record:

- *^275600 REMOVED FROM DATABASE*: indica chiaramente che l'identificativo 275600 è stato eliminato dal database;
- *^275650 MOVED TO 214950*: indica che l'entità rappresenata dall'identificativo 275650 è stata sostituita dalla feature avente ID 214950.

▪ *Simbolo: CDS*

Rappresenta il simbolo del gene o del fenotipo.

Righe successive:

▪ *Titolo alternativo: NEUTRAL LIPID STORAGE DISEASE WITH ICHTHYOSIS*

Il testo racchiuso tra la sequenza di caratteri *";;"* e *";;"* oppure tra *";;"* e *";"*.

▪ *Simbolo alternativo: NLSDI*

Il testo racchiuso tra la sequenza di caratteri *";"* e *";;"*.

- **FIELD* TX*

```
*FIELD* TX

DESCRIPTION

Combined methylmalonic aciduria (MMA) and homocystinuria is a
genetically heterogeneous disorder of cobalamin (cbl; vitamin
B12)
metabolism. The defect causes decreased levels of the coenzymes
adenosylcobalamin (AdoCbl) and methylcobalamin (MeCbl), which
results in
...

CLINICAL FEATURES

- CblD Combined Homocystinuria and Methylmalonic Aciduria

Goodman et al. (1970) reported 2 brothers, from a consanguineous
family,
```

```
with combined homocystinuria and methylmalonic aciduria. The
older sib
presented with developmental delay and neurologic abnormalities
at age
...
```

Figura 61 – Esempio di campo *FIELD* TX file omim.txt

Campo costituito da tre sottocampi:

- *Titolo: DESCRIPTION*

  È identificato dalla riga scritta con caratteri maiuscoli.

- *Sottotitolo: CblD Combined Homocystinuria and Methylmalonic Aciduria*

  È identificato dal carattere *"-"* ad inizio riga e dalla prima lettera del testo maiuscola.

- *Body: Goodman et al. (1970) reported 2 brothers, from a consanguineous family….*

  È costituito dalle righe di testo che non rientrano in nessuna delle due classificazioni precedenti.

Tutti e tre i sottocampi sono di tipo opzionale. All'interno dello stesso campo *FIELD TX* è possibile trovare più blocchi di tipo *Body* con il proprio titolo, come nell'esempio sopra riportato.

- **FIELD* AV*

```
*FIELD* AV
.0001
THYROXINE-BINDING GLOBULIN DEFICIENCY, COMPLETE
COMPLETE DEFICIENCY 5
TBG, LEU227PRO

In a French-Canadian male with complete TBG deficiency, Mori et
al.
(1990) found a CTA-to-CCA change at codon 227, leading to
substitution of proline for leucine. In addition, a TTG (leu)-
to-TTT (phe) change at codon 283 was found. The leu-to-phe
substitution is a TBG polymorphism present in about 16% of
French-Canadian males (see 314200.0003). It has no effect on the
serum concentration or properties of the common type of TBG
(TBG-C). Janssen et al. (1992) referred to this variant as CD5
for 'complete deficiency 5.'

.0002
THYROXINE-BINDING GLOBULIN, VARIANT A TBG-A;;TBG-ABORIGINE TBG,
ALA191THR

Takeda et al. (1989) showed that the sequence of the gene for
TBG-A
present in Australian aborigines differs at 2 positions from
that of the normal TBG allele: ACA (threonine) for GCA (alanine)
at codon 191 and TTT (phenylalanine) for TTG (leucine) at codon
283. They concluded that…
```

Figura 62 – Esempio di campo *FIELD* AV file omim.txt

Il campo **FIELD* AV* è di tipo opzionale ed è associato unicamente a record di geni; fornisce informazioni sulle varianti alleliche, un'alterazione della normale sequenza di un gene, il cui significato è spesso poco chiaro fino a un ulteriore approfondimento del genotipo corrispondente. I criteri di verifica utilizzati da OMIM si basano su: la prima mutazione scoperta, la frequenza della mutazione, il fenotipo distintivo, l'importanza storica, il meccanismo della mutazione, il meccanismo patogenetico e l'eredità distintiva (ad esempio, dominante con alcune mutazioni, recessivo con altre mutazioni nello stesso gene). La maggior parte delle varianti alleliche rappresenta mutazioni causate da malattie.

Una variante allelica è identificata dal Mim number dell'elemento principale cui si riferisce seguito da un punto decimale e da un numero univoco di quattro cifre che determina la variante. Ad esempio, la beta-globina locus (HBB) è definita dal codice 141900 e la sua variante emoglobina falciforme è codificata dal numero 141900.0243; le varianti alleliche presso il fattore IX (emofilia B) locus sono numerate 306900.0001-306900.0101. L'esempio precedentemente esposto mostra come un gene possa essere oggetto di più mutazioni e dunque abbia più varianti alleliche associate.

Il campo è così costituito:

- *ID: 0001*

  È l'identificativo della variante allelica. Contrariamente alla definizione del codice identificativo della variante, descritta in precedenza, nel file è indicata solo la parte variante, composta dal numero di quattro cifre;

- *Titolo: THYROXINE-BINDING GLOBULIN DEFICIENCY, COMPLETE*

  Rappresentato dalla riga successiva al codice identificativo, è espresso in caratteri maiuscoli.

Dopo il titolo sono presenti altre righe indicate in caratteri maiuscoli, l'ultima di queste è costituita da due campi, simbolo e descrizione separati dal carattere *","*, mentre le altre indicano i sottotitoli;

- *Simbolo: TBG*

  Rappresenta il simbolo della variante allelica.

- *Titolo: LEU227PRO*

  Indica un codice aggiuntivo che rappresenta la variante allelica rilevata.

- *Sottotitolo: COMPLETE DEFICIENCY 5*

  Indica un'ulteriore descrizione della variante rilevata.

- *Testo: In a French-Canadian male with complete TBG deficiency, Mori et al….*

  La parte rimanente di testo non riportata in caratteri maiuscoli rappresenta la descrizione testuale dettagliata della variante.

- **FIELD* SA*

```
*FIELD* SA
Balmer and Zografos (1980); Beauchamp  (1978); Curran and Robb
(1976);
Delleman and Winkelman (1973); Funderburk et al. (1977);
Narahara et al. (1984); Oliver et al. (1987); Rutledge et al.
(1986);
```

Figura 63 – Esempio di campo *FIELD* SA file omim.txt

Il campo **FIELD* SA* riporta indicazioni di pubblicazioni; le stesse informazioni sono indicate anche nel campo **FIELD* RF*, data la ridondanza dei dati, il campo è escluso dal processo d'importazione.

- **FIELD* RF*

```
*FIELD* RF
1. Arroyave, C. P.; Scott, I. U.; Gedde, S. J.; Parrish, R. K.;
Feuer, W. J.: Use of glaucoma drainage devices in the management
of glaucoma associated with aniridia. Am. J. Ophthal. 135: 155-
159, 2003.

2. Atchaneeyasakul, L.; Trinavarat, A.; Dulayajinda, D.;
Kumpornsin, K.; Thongnoppakhun, W.; Yenchitsomanus, P.;
Limwongse, C.: Novel and de-novo truncating PAX6 mutations and
ocular phenotypes in Thai aniridia patients. Ophthalmic Genet.
27: 21-27, 2006.
```

Figura 64 – Esempio di campo *FIELD* RF file omim.txt

Campo opzionale, riporta i riferimenti tra i record di OMIM e le citazioni degli stessi in pubblicazioni o articoli biomedici; sono indicati la citazione bibliografica completa del riferimento, autore o autori, il titolo dell'articolo, il nome della rivista, il volume, i numeri di pagina, e l'anno.

- *Numero: 1*

  Indica il numero progressivo del riferimento, utilizzato per identificare la citazione;

- *Autori: Arroyave, C. P.; Scott, I. U.; Gedde, S. J.; Parrish, R. K.; Feuer,W. J.*

  Il testo che precede il carattere *":"*, riporta i nominativi degli autori della pubblicazione separati tra loro dal carattere *";"* .

- *Titolo: Use of glaucoma drainage devices in the management of glaucoma associated with aniridia*

  Indica il titolo della pubblicazione.

- *Rivista: Am. J. Ophthal*

  Indica il nome della rivista nella quale è stato pubblicato l'articolo in questione; il testo riportato nell'esempio fa riferimento alla rivista American Journal of Ophthalmology.

- *Numero volume: 135*

Indica il numero della rivista sulla quale è apparso l'articolo.

- *Pagine: 155-159*

Indica l'intervallo di pagine in cui è riportato l'articolo in questione.

- *Anno: 2006*

Indica l'anno di pubblicazione.

- **FIELD* CN*

```
*FIELD* CN
Marla J. F. O'Neill - updated: 10/16/2008
Cassandra L. Kniffin - reorganized: 8/27/2002
Stylianos E. Antonarakis - updated: 5/18/2001
Dawn Watkins-Chow - updated: 3/27/2001
Victor A. McKusick - updated: 1/16/2001
Ada Hamosh - updated: 7/28/2000
```

Figura 65 – Esempio di campo *FIELD* CN file omim.txt

Campo di tipo opzionale, riporta i riferimenti di chi ha partecipato all'integrazione del database di OMIM:

- *Autore: Marla J. F. O'Neill*

Indica il nome di chi ha eseguito l'operazione sul database.

- *Azione: updated*

Campo di tipo opzionale, indica la tipologia di operazione eseguita, i valori che questo campo può assumere sono i seguenti: *updated, reorganized, revised, edited, reviewed.*

- *Data: 10/16/2008*

Indica la data nella quale è stata eseguita l'azione; il dato è fornito nel formato MM/GG/AAAA.

- **FIELD* CD*

```
*FIELD* CD
Victor A. McKusick: 6/4/1986
```

Figura 66 – Esempio di campo *FIELD* CD file omim.txt

Il campo **FIELD* CD* è sempre presente, riporta i riferimenti della creazione dell'elemento in OMIM; è costituito sempre da una sola riga ed è suddiviso in altri due dal carattere *":"*:

- *Autore: Victor A. McKusick*

Indica il nominativo di chi ha inserito l'elemento nel database di OMIM.

- *Data: 6/4/1986*

Indica la data in cui è stato creato l'elemento, il dato è fornito nel formato MM/GG/AAAA.

- **FIELD* ED*

```
*FIELD* ED
wwang: 10/16/2008
carol: 10/7/2008
carol: 10/6/2008
carol: 9/18/2008
carol: 4/15/2008
carol: 8/13/2007
ckniffin: 7/27/2007
```

Figura 67 – Esempio di campo *FIELD* ED file omim.txt

Il campo **FIELD* ED* è opzionale, riporta i riferimenti di chi ha compilato o corretto il contenuto dell'elemento di OMIM.
Il campo è suddiviso in altri due sottocampi dal carattere *":"*:

- *Autore : wwang*

  Indica il nominativo dell'autore della modifica.

- *Data: 6/4/1986*

  Indica la data in cui è stato compilato l'elemento, il dato è fornito nel formato MM/GG/AAAA.

- **FIELD* CS*

```
*FIELD* CS

HEAD AND NECK:
   [Eyes];
   Aniridia;
   Decreased vision;
   Cataract;
   Glaucoma;
   Peter's anomaly (congenital anomaly of the anterior segment);
   Corneal clouding
```

Figura 68 – Esempio di campo *FIELD* CS file omim.txt

Campo opzionale, riporta i dati della sinossi clinica, la localizzazione dei fenotipi.
I dati estratti concorrono alla costruzione di un vocabolario controllato delle definizioni utilizzate per identificare le parti del corpo umano nelle quali si manifestano i sintomi legati alle patologie evidenziate in OMIM.
Il campo **FIELD* CS* è suddiviso in altri tre sottocampi:

- *Titolo: INHERITANCE*

  Indica il nome dell'apparato (cardiovascolare, motorio) o del sistema (immunitario, nervoso) che evidenzia uno o più sintomi; è il primo elemento delle liste di clinical synopsys presenti nel campo **FIELD*CS* ed è terminato dal carattere *":"*.

- *Sottotitolo: Eyes*

Rappresenta la parte specifica dell'apparato indicato nel titolo che evidenzia determinati sintomi, è sempre indicato tra parentesi quadre "*[ ]*" e può essere terminato dai caratteri *";"* o *"."*.

- *Sintomo o segno: Aniridia*

Indica il sintomo che è stato riscontrato per la patologia associata; è contenuto nel testo non appartenente ai sottocampi precedenti e può essere terminato dai caratteri *";"* o *"."*.

Esiste la possibilità che all'interno del record di tipo **FIELD* CS* siano presenti uno o due campi di tipo **FIELD* CD*, **FIELD* ED* e **FIELD* CN*, come proposto nell'esempio all'inizio del paragrafo, vedi figura 58.

Per capire il significato di questo doppia definizione di campi identici è importante la sequenza con cui sono indicati nel record; in presenza di due campi dello stesso tipo, il primo campo si riferisce ai dati indicati nel campo **FIELD* CS* mentre il secondo campo è attribuito al record di OMIM.

In pratica se ci sono due campi **FIELD* CD*, il primo riporta l'autore e la data di creazione dei dati contenuti in **FIELD* CS*, mentre il secondo **FIELD* CD* indica l'autore e la data di creazione dell'entry OMIM.

Con le stesse regole si possono interpretare i due campi **FIELD* ED*, e **FIELD* CN*.

Per comodità nel seguito del testo i campi **FIELD* CD*, **FIELD* ED* e **FIELD* CN* che fanno riferimento ad un campo **FIELD* CS* saranno chiamati rispettivamente **FIELD* CDcs*, **FIELD* EDcs* e **FIELD* CNcs*.

Maggiori dettagli sulla gestione dei dati estratti dal campo CS sono discussi nel paragrafo 9.2.6.

Dal parsing di questo file si popolano differenti tabelle secondo il tipo di record letto:

- se il record letto descrive un gene, le tabelle popolate sono:
  - *omim_gene;*
  - *omim_gene_alternative;*
  - *omim_gene_text;*
  - *omim_gene_edit_history;*
  - *omim_gene_contributors;*
  - *omim_gene_reference_publication;*
  - *omim_gene_allelic_variant;*
  - *omim_gene_clinical_synopsys_edit_history;*
  - *omim_gene_clinical_synopsys_contributors;*
  - *omim_gene_synopsys_relationship;*
- se il record letto descrive un fenotipo le tabelle popolate sono:
  - *omim_disorder;*
  - *omim_disorder_alternative;*

- *omim_disorder_text;*
- *omim_disorder_edit_history;*
- *omim_disorder_contributor;*
- *omim_disorder_reference_publication;*
- *omim_disorder_clinical_synopsys_edit_history;*
- *omim_disorder_clinical_synopsys_contributors;*
- *omim_disorder_synopsys_relationship.*

Indipendentemente dalla tipologia del record di appartenenza, in presenza dal campo **FIELD* CS* si popolano le tabelle:

- *omim_clinical_synopsys;*
- *omim_clinical_synopsys_relationship;*
- *omim_clinical_synopsys_soubgroup.*

In seguito al popolamento della tabella *omim_clinical_synopsys* è necessario relazionare il suo contenuto con i rispettivi geni o fenotipi. A tale scopo si popolano le tabelle di associazione esposte di seguito.

Se il record letto descrive un gene:

- *gene2clinical_synopsys_imported;*
- *phenotype_4_gene2clinical_synopsys_imported.*

Viceversa, se il record letto descrive un fenotipo vengono popolate le seguenti tabelle:

- *genetic_disorder2clinical_synopsys_imported;*
- *phenotype_4_genetic_disoder2clinical_synopsys_imported.*

9.2.2 File genemap.key

Il file *genemap.key* contiene una descrizione del formato del file *genemap* e dei codici utilizzati per popolare i campi *status* e *method* del file.

Il Loader realizzato per l'importazione è *GenemapKeyLoader.java,* presente all'interno del package *gfinder2.importer.omim,* ed estende il Parser *FlatFileWithFixLengthHeaderParser.*

È considerata header la parte iniziale del file, nella quale vi è l'elenco dei campi contenuti nel file *genemap*. I record sono separati dall'header da una linea di testo composta dal carattere *"-"* e, come si può notare in figura 69, elencano le descrizioni dei codici utilizzati per popolare i campi *status* e *method* del file *genemap*.

```
OMIM Gene Map Key
Each entry is a list of fields, separated by the '|' character.
The fields are in order:
1 -   Numbering system, in the format  Chromosome.Map_Entry_Number
...
------------------------------------------------------------------------
---
Status codes:
C = confirmed - observed in at least two laboratories or in several
families.
P = provisional - based on evidence from one laboratory or one family.
```

```
...
------------------------------------------------------------------------
---
Method codes:
A   =  in situ DNA-RNA or DNA-DNA annealing (`hybridization'); e.g. ...
AAS = deductions from the amino acid sequence of proteins; e.g. ...
...
```

Figura 69 – Esempio di struttura del file genemap.key

La prima riga di ciascun record viene scartata mentre le successive sono processate in modo da estrarre una coppia codice-descrizione riconosciute attraverso il carattere separatore *"="*, come mostrato nell'esempio seguente:

- *Campo1: Code = C;*
- *Campo2: Description = confirmed - observed in at least two laboratories or in several families.*

Utilizzando le procedure automatiche per il popolamento delle tabelle di flag presenti nel framework, sono compilate le tabelle:

- *status_flags;*
- *method_flags.*

La verifica dei dati importati ha mostrato che non tutti i method codes presenti in *genemap* sono stati elencati nel file *genemap.key*. Ciò è dovuto al fatto che mentre il file *genemap* viene aggiornato costantemente, l'ultima versione del file *genemap.key* risale a parecchi anni fa. Si è scelto quindi di aggiungere i codici mancanti nella relativa tabella di flag tramite codice, creando una nuova enumerazione *MethodCodeFlag.java* che implementa l'interfaccia *IEncodedFlag*, creata per gestire l'aggiunta di valori di campi codificati all'interno del codice del framework GPDW.

9.2.3 File genemap

Il file *genemap* contiene alcuni valori del database OMIM che non sono presenti in *omim.txt*. Il file è di tipo tabellare con unico separatore di campo.

Il Loader che importa il file è *GenemapLoader.java,* presente all'interno del package *gfinder2.importer.omim,* ed estende il Parser *TabularFileParser*.

I campi sono separati dal carattere pipe *"|"*. In figura 70 è proposto l'esempio di un tipico record del file.

```
1.54|4|14|05|1p36.3-p36.2|PLOD,    PLOD1|P|Procollagen-lysine,    2-
oxoglutarate    5-dioxygenase    (lysine    hydroxylase)||153454|REa,
A|||Ehlers-Danlos  syndrome,  type  VI,  225400  (3);  Nevo  syndrome,
601451|(3)  |  |4(Plod)|Hautala (1992)
```

Figura 70 – Esempio di struttura del file genemap

I campi identificati sono:

- *Campo1: Numbering system =1.54*

  Sistema di numerazione delle entries presenti in questo file, ed è costituito da due campi suddivisi dal carattere *"."*; i due sottocampi sono:

  - *Chromosome =1*
  - *Map_Entry_Number = 54*
- *Campo2: Month entered = 4*
  Mese in cui è stato catalogato il record nel file *genemap*;
- *Campo3: Day entered = 14*
  Giorno in cui è stato catalogato il record nel file *genemap*;
- *Campo4: Year entered = 05*
  Anno in cui è stato catalogato il record nel file *genemap*;
- *Campo5: Location = 1p36.3-p36.2*
  Definisce in quale cromosoma e in che posizione del cromosoma è presente il gene importato;
- *Campo6: Symbol = PLOD, PLOD1*
  Indica il simbolo o i simboli associati al gene che si sta importando;
- *Campo7: Status = P*
  Indica la certezza con la quale è stata stabilita l'assegnazione dei loci dei cromosomi o il collegamento tra due loci;
  I valori che il campo può assumere sono:
  - *C : "confirmed - observed in at least two laboratories or in several families.";*
  - *P: "provisional - based on evidence from one laboratory or one family.";*
  - *I: "inconsistent - results of different laboratories disagree.";*
  - *L: "limbo - evidence not as strong as that provisional, but included for heuristic reasons.".*

  Questi valori sono stati codificati in precedenza e inseriti nella tabella *status_flags* dal Loader del file *genemap.key*. Sono utilizzati per validare i valori estratti dal campo *Status* del file *genemap*; nel caso il confronto dia esisto negativo il valore non viene inserito nella tabella e viene mostrato un messaggio di warning nel log del programma;
- *Campo8: Title= Procollagen-lysine, 2-oxoglutarate 5-dioxygenase (lysine hydroxylase)*
  Titolo del gene o del fenotipo;
- *Campo9:* campo vuoto*;*
- *Campo10: Mim number = 153454*
  Codice identificativo assegnato da OMIM;
- *Campo11: Method = Rea, A*
  Esprime i metodi per la mappatura dei geni. Come nel caso del campo *Status* questi valori devono essere validati con quelli già presenti nella tabella *method_flags* popolata dal Loader *GenemapKeyLoader;*
- *Campo12: Comments*
  Indica i commenti relativi al gene o al fenotipo;

- *Campo13:* campo vuoto
- *Campo14: Disorders = Ehlers-Danlos syndrome, type VI, 225400 (3); Nevo syndrome, 601451*
- *Campo15: Disorders, cont. = (3)*
- *Campo16: Disorders, cont* = campo vuoto

  I campi 14,15,16 esprimono la malattia o le malattie associate al record importato. Nel caso sia indicata più di una malattia, ciascuna è suddivisa dal carattere *";"* individuando dei sottocampi.

  Se nel primo campo non vi è sufficiente spazio per indicare il nome della malattia, il testo mancante è inserito nel campo successivo.

  Come mostra l'esempio, il *campo14* non è abbastanza grande da contenere il testo della seconda malattia indicata, *Nevo syndrome, 601451 (3)*; una parte di testo viene inserita nel *campo14* fino a completamento dello spazio disponibile ed il testo rimanente, *(3)*, è inserito nel *campo15*.

  I campi possono essere considerati come un unico grande campo suddiviso in tre.

  Dai sottocampi si estrapolano definizioni e valori specifici con riferimento alla malattia espressa nel testo, identificati da particolari formattazioni o sequenze di caratteri come esposto di seguito:

  - *Desease subtitle = Ehlers-Danlos syndrome, type VI*

    Indica il nome della malattia o del disturbo;
  - *Mim number associato al Desease = 225400*

    Indica il codice identificativo assegnato da OMIM alla malattia, i dati relativi a tale codice sono stati precedentemente importati nella tabella *omim_disorder* dal Parser *OmimLoader.java*. Il codice è individuabile alla fine del testo del sottocampo di ogni disturbo ed è costituito dal testo che segue l'ultimo carattere *","* e precede la sequenza *"(n)",* per n si intende un numero compreso tra 4 e 1.
  - *Mutation position= (3)*

    Questo campo è costituito dal numero riportato tra parentesi tonde alla fine del testo di ogni sottocampo, viene codificato nel seguente modo:

    (1) : *"Mutation positioned by mapping the wildtype gene";*

    (2) : *"Mutation by mapping the disease phenotype itself";*

    (3) : *"Mutation was positioned by mapping the wildtype gene and by mapping the disease phenotype itself.";*

    (4) : *"Unknow".*
  - *Desease type*: non è presente nell'esempio.

    Indica il tipo del disturbo:

    - se il testo del campo Desease è racchiuso tra parentesi quadre *"[ ]"* viene codificato come: *"Mutations that lead to universal susceptibility to a specific infection, to frequent resistance to a specific infection."*;

  - se il testo del campo Desease è racchiuso tra parentesi graffe *"{ }"* viene codificato come: *"Certain "nondiseases", mainly genetic variations that lead to apparently abnormal laboratory test values.";*
- *Limbo*: non è presente nell'esempio.
  Il campo assume valore *True* se prima del nome del disturbo è presente il carattere *"?"*, tale codice è equivalente allo stato Limbo indicato per il *campo7*.

- *Campo17: Mouse correlate = 4(Plod)*
  Riferimento ai cromosomi del topo.
- *Campo18: Reference = Hautala (1992)*
  Esprime il nome e l'anno della pubblicazione, o dell'articolo in cui è stato citato il record che si sta importando. Il campo è composto da due sottocampi:
  - *Autore riferimento = Hautala*
  - *Anno riferimento = 1992*
    Il campo anno è proposto con le seguenti formattazioni:
    - *Abdalla (1992); Koch (1992)*: più autori suddivisi dal carattere *";"*;
    - *Small (1991, 1992)*: più anni riferiti allo stesso autore separati dal carattere *","*;
    - *Ryan (1992a, 1992b)* : più anni riferiti allo stesso autore separati dal carattere *","* con l'aggiunta di una lettera per differenziarli.

Se il codice estratto dal *campo10* identifica un gene, le tabelle popolate sono:

- *omim_gene;*
- *omim_disorder;*
- *omim_gene_alternative;*
- *omim_gene_reference_publicat ion;*
- *omim_gene_method.*

Viceversa, se il codice estratto dal *campo10* descrive un fenotipo, le tabelle popolate sono:

- *omim_disorder;*
- *omim_disorder_alternative;*
- *omim_disorder_reference_publication;*
- *omim_disorder_method.*

9.2.4 File morbidmap

Il file *morbidmap* contiene tipologie di valori simili a quelli descritti per il file precedente. Il file è di tipo tabellare con unico separatore di campo.
Il Loader che importa il file è *MorbidmapLoader.java*, presente all'interno del package *gfinder2.importer.omim,* ed estende il Parser *TabularFileParser*.
I campi sono separati dal carattere pipe *"|"*, in figura 71 è mostrato un esempio di un tipico record del file.

```
Corneal dystrophy, hereditary polymorphous posterior, 122000 (3)
|VSX1, RINX, PPCD, PPD, KTCN|605020|20p11.2
```

Figura 71 – Esempio di struttura del file morbidmap

I campi individuati sono:

- *Campo1: Corneal dystrophy, hereditary polymorphous posterior, 122000 (3)*
  Esprime il nome del disturbo, la gestione del campo è stata eseguita come per i campi *Disorders* del file *genemap* (vedi paragrafo precedente).
- *Campo2: VSX1, RINX, PPCD, PPD, KTCN*
  Esprime i simboli associati al gene o al genetic disorder indicato al *Campo3*, la gestione del campo è stata eseguita come per il *Campo6* del file *genemap* (vedi paragrafo precedente).
- *Campo3:605020*
  Codice identificativo assegnato da OMIM;
- *Campo4: 20p11.2*
  Esprime la localizzazione del gene, assume lo stesso significato del *Campo5* del file *genemap* ed è stato gestito con le stesse modalità(vedi paragrafo precedente).

Se il codice reperito dal *Campo3* identifica un gene, le tabelle popolate sono:

- *omim_gene;*
- *omim_disorder;*
- *omim_gene_alternative.*

Viceversa, se identifica un fenotipo, le tabelle popolate sono:

- *omim_disorder;*
- *omim_disorder_alternative.*

9.2.5 File pubmed_cited

Il file *pubmed_cited* propone l'associazione tra i codici identificativi dei record di OMIM e i codici identificativi di PubMed.
PubMed è un database bibliografico contenente informazioni sulla letteratura scientifica biomedica dal 1949 ad oggi, comprende più di 19 milioni di citazioni di articoli biomedici e riviste scientifiche.
Il file intende rappresentare il legame esistente tra le pubblicazioni di PubMed e gli elementi di OMIM citati in queste pubblicazioni.
Questa sorgente è di tipo tabellare con unico separatore di campo. Il Loader che importa il file è *PubmedLoader.java,* presente all'interno del package *gfinder2.importer.omim,* ed estende il Parser *TabularFileParser*. I campi sono separati dal carattere tabulazione.
Di seguito è proposto un esempio di alcuni record del file.

```
100050   1     6344635
100050   2     8322809
```

```
100070   1     7651004
100070   2     3047388
100070   3     588966
```

Figura 72 – Esempio di struttura del file pubmed_cited

Prendendo in esame la prima riga, i campi che si possono estrarre sono i seguenti:

- *Campo1: Mim number = 100050*
  Codice identificativo assegnato da OMIM.
- *Campo2*: *contatore progressivo*=1 (campo non importato)
- *Campo3: PubMedID = 6344635*
  Codice identificativo della sorgente dati PubMed.

Se il codice reperito dal *Campo1* identifica un gene, la tabella popolata è *gene2publication_reference_imported*; viceversa, se il codice reperito dal *Campo1* identifica un fenotipo, la tabella popolata è *genetic_disorder2publication_reference_imported.*

9.2.6 Normalizzazione e strutturazione delle localizzazioni dei fenotipi

L'analisi dei fenotipi è un'attività fondamentale per comprendere le interazioni genetiche alla base di malattie ereditarie[22]. Poiché le informazioni raccolte in OMIM provengono da testo libero, molto spesso sono utilizzati termini diversi per definire lo stesso concetto.
È stato necessario normalizzare le informazioni inerenti le localizzazioni dei fenotipi, memorizzate nella tabella *omim_clinical_synopsys*, in modo da ottenere una lista, organizzata gerarchicamente, di termini controllati . Ciò comporta l'identificazione dei valori sinonimi, e il mantenimento in tabella solo del valore più rappresentativo di ogni gruppo di sinonimi (es. *"abd"* e *"abdomen"* sono sinonimi; si mantiene *"abdomen"* eliminando *"abd"*). Per evitare problemi nella gestione dei valori degli altri campi presenti in tabella, ho realizzato tale azione direttamente all'atto del caricamento dei dati (nel Loader *OmimLoader.java*). Durante il popolamento della tabella *omim_clinical_synopsys* ho individuato se un dato valore di *name* è da inserire, essendo unico o sinonimo preferenziale di altri, o se è da trasformare in un altro valore prima di inserirlo in tabella (se non è stato già inserito). Per mantenere traccia di tutti i valori nel file, ho creato e popolato parallelamente in schema *log* una tabella uguale a *omim_clinical_synopsys, omim_clinical_synopsys_orig*.
La realizzazione del punto precedente ha richiesto la definizione di una struttura dati di appoggio (vedi figura 73) nel file *data_sources.xml,* in cui per ogni valore possibile del campo *name* della tabella *omim_clinical_synopsys* è indicato se è un valore preferenziale, quindi da mantenere, o se è un valore sinonimo, da sostituire con l'equivalente valore preferenziale, indicando quale questo sia. Dato che vi sono valori di nomi di clinical synopsys ambigui, cioè uguali ma riferiti a gruppi diversi, ho aggiunto la possibilità di definire il nome dell'item padre, in modo da differenziarli. Nel caso dal parsing del file si estragga un valore avente item padre diverso da quello definito in XML viene segnalata l'anomalia nel log.

Inoltre, dato che risultava utile raggruppare più valori preferenziali in uno stesso gruppo, ho creato una tabella aggiuntiva, *omim_clinical_synopsys_subgroup*, che descrive quali valori preferenziali sono raggruppati in quale/i gruppo/i. La definizione di tale tabella è stata aggiunta alla struttura XML descritta in precedenza.

```
-<omim_clinical_synopsys_synonyms source_handle=" " file_handle=" ">
   <group name=" " group_handle=" "/>
   <group name=" " group_handle=" "/>
  -<item name=" ">
     <synonym_name/>
     <group handle=" "/>
   </item>
  -<item name=" " has_father_name=" ">
     <is_preferred/>
     <group handle=" "/>
     <group handle=" "/>
   </item>
 </omim_clinical_synopsys_synonyms>
```

Figura 73 – Struttura XML utilizzata per definire le localizzazioni dei fenotipi estratte da OMIM

Ho legato tale struttura all'handle della sorgente e del file che ne fornisce i dati, in modo che sia processata solo quando avviene l'importazione di quel file.
Per facilitare i controlli, ho creato nello schema *log* una tabella *omim_clinical_synopsys_normalization* contenente i valori estratti da questa struttura XML.
Nel caso nel file *omim.txt* sia presente un valore name di clinical synopsys non presente nella struttura, viene mostrato nel log del programma un messaggio di errore, ma il valore viene comunque inserito in tabella *omim_clinical_synopsys*. L'amministratore in base all'errore evidenziato aggiungerà all'interno della struttura XML un <*item*> corrispondente, in modo che al successivo run non si manifesti più l'errore.
In tabella 6 è mostrata la struttura gerarchica definita per alcune delle localizzazioni dei fenotipi estratte dalla banca dati OMIM.

Phenotype location
Abdomen
Biliary tract
External features
Gastrointestinal
Liver
Pancreas
Spleen
Cardiovascular
Cardiac
Heart
Vascular
Genitourinary
Bladder
External genitalia
External genitalia, female
External genitalia, male

Internal genitalia
Internal genitalia, female
Internal genitalia, male
Kidneys
Ureters
Neurologic
Behavioral/psychiatric manifestations
Central nervous system
Peripheral nervous system

Tabella 6 – Esempio di struttura gerarchica definita per le localizzazioni dei fenotipi estratte da OMIM

Dall'analisi dei dati estratti dalla banca dati OMIM, è emersa la necessità di definire nuove relazioni tra i nomi di clinical synopsys poiché mancanti nei file di sorgente. Ad esempio tra "endocrine" (padre) e "endocrine feature" (figlio). Ho, quindi, introdotto la possibilità di avere un tag <*has_parent_name*> solo per "preferred items" da gestire nel codice al termine dell'importazione dei dati, inserendo in tabella *omim_clinical_synopsys_relationship* una tupla corrispondente. Prima dell'inserimento ho però controllato che tra le relazioni presenti in *omim_clinical_synopsys_relationship* non fossero già presenti le relazioni inverse (es. "endocrine feature" – "endocrine"); in tal caso l'anomalia viene segnalata nel log con un messaggio di warning e non si inserisce la nuova relazione, che altrimenti creerebbe un errore durante l'esecuzione della procedura di unfolding. Infine, viene segnalata la presenza di relazioni di tipo gerarchico di un item con se stesso; tali relazioni vengono eliminate dalla tabella *omim_clinical_synopsys_relationship* e memorizzate nella tabella *omim_clinical_synopsys_relationship_deleted_entries* creata nello schema *log*.

9.2.7 Eliminazione/merge delle tuple duplicate

I dati importati da OMIM sono particolarmente complessi e la gestione delle tuple con valore dei campi *source_id* e *source_name* duplicato non può essere semplicemente demandato alla procedura descritta nel capitolo 6. Essa, infatti, fallisce nel tentativo di eseguire il merge per molte delle tuple importate. Queste tuple vengono eliminate dalle tabelle dello schema *public* e inserite nelle tabelle **_err* dello schema *log*. È stato quindi necessario analizzare il motivo per il quale tali tuple sono eliminate e studiare una procedura ad hoc per recuperare il maggior numero possibile d'informazioni. Dall'analisi è emerso che i campi aventi valori contrastanti per lo stesso ID sono *title, symbol* e *is_obsolete* delle tabelle *omim_gene* e *omim_disorder*. In particolare, i valori dei campi *title* e *symbol* riferiti a uno stesso identificativo risultano contrastanti in quanto memorizzati in modo diverso nei vari file di sorgente forniti da OMIM. Per il campo *is_obsolete* il problema deriva dal fatto che esso è definito *NOT NULL*, quindi i record forniti da una sorgente diversa da *omim.txt,* per i quali non è rappresentata tale informazione, hanno tale campo settato di default a *false;* ciò rendeva quindi impossibile eseguire il merge tra i record obsoleti forniti dal file *omim.txt* e i record forniti dagli altri file.
Si è scelto quindi di eliminare temporaneamente tali colonne dalle tuple presenti nella tabella **_err*, cioè le tuple rimosse dalla tabella originale in seguito a una prima applicazione

della procedura DuplicatesChecker, rieseguire la procedura per l'eliminazione/merge duplicati su queste entries e reinserire i valori di *title, symbol* e *is_obsolete* prendendo solo i valori forniti dal file *omim.txt,* considerato il più affidabile tra quelli forniti da OMIM. Infine, occorre recuperare i valori dei campi *symbol* e *title* delle tuple non reintegrate, inserendoli nelle tabelle *omim_gene_alternative* e *omim_disorder_alternative.*

Poiché la procedura DuplicatesChecker è invocata un numero ripetuto di volte su una stessa tabella, si è scelto di utilizzare un suffisso diverso per ogni chiamata, in modo da non sovrascrivere le tabelle generate nello schema *log* dalla chiamata precedente.

Si elencano di seguito i vari passi seguiti e le query utilizzate per ottenere i dati importati desiderati:

1. Applicazione della procedura per il merge delle tuple duplicate alla tabella *omim_gene* (con suffisso "_1").

2. Inserimento in tabella *omim_gene_alternative* dei valori dei campi *title* e *symbol* selezionati dalla tabella *omim_gene_err_1* forniti da sorgente diversa da *omim.txt* (in modo che possano esserne aggiornati i valori dei campi FK).

```
INSERT INTO omim_gene_alternative (SELECT DISTINCT omim_gene_oid, gene_title, symbol,
reference_file FROM log.omim_gene_err_1 WHERE reference_file NOT IN (SELECT
reference_file_id FROM metadata.source_reference_file WHERE imported_file_xml_id =
'omim_omimData_File'))
```

Figura 74 – Query utilizzata per l'inserimento dei titoli e dei simboli alternativi nella tabella omim_gene_alternative

3. Aggiornamento dei valori dei campi FK nelle tabelle secondarie di *omim_gene.*

4. Creazione della tabella *omim_gene_err_1_notitle* e inserimento delle entries della tabella *omim_gene_err_1* in tabella *omim_gene_err_1_notitle.*

```
CREATE TABLE log.omim_gene_err_1_notitle AS (SELECT * FROM log.omim_gene_err_1))
```

Figura 75 – Query utilizzata per la creazione della tabella omim_gene_err_1_notitle

5. Eliminazione delle colonne *gene_title*, *symbol* e *is_obsolete* dalla tabella *omim_gene_err_1_notitle.*

```
ALTER TABLE log.omim_gene_err_1_notitle DROP COLUMN gene_title
ALTER TABLE log.omim_gene_err_1_notitle DROP COLUMN symbol
ALTER TABLE log.omim_gene_err_1_notitle DROP COLUMN is_obsolete
```

Figura 76 – Query utilizzate per l'eliminazione dei campi gene_title, symbol e is_obsolete dalla tabella omim_gene_err_1_notitle

6. Applicazione procedura *DuplicatesChecker* su *omim_gene_err_1_notitle* con suffisso "_2". La tabella *omim_gene_err_1_notitle_err_1* creata nello schema log conterrà tutte le entries per le quali non è stato possibile eseguire il merge, nonostante

l'eliminazione dei campi *title, symbol* e *is_obsolete*, cioè tutte le entries eliminate in modo definitivo dalla tabella *omim_gene*.

7. Aggiornamento dei valori dei campi FK nelle tabelle secondarie di *omim_gene,* utilizzando come campo di riferimento del vincolo chiave esterna il campo *omim_gene_oid* della tabella *omim_gene_err_1_notitle* (la tabella *omim_gene_err_1_notitle* non è fisicamente legata a tabelle secondarie, ma le entries sulle quali è eseguito il merge sono le stesse che saranno reinserite in *omim_gene,* quindi i valori dei campi chiave esterna nelle tabelle secondarie di *omim_gene* devono essere aggiornati anche in questo caso).

8. Recupero dei campi *title*, *symbol* e *is_obsolete* tramite il "join" tra le tabelle *omim_gene_err_1_notitle* e *omim_gene_err_1*, considerando solo le entries aventi come file di sorgente (campo *reference_file*) *omim.txt* e creazione della tabella *omim_gene_err_1_title*.

```
DROP TABLE IF EXISTS log.omim_gene_err_1_title CASCADE; CREATE TABLE
log.omim_gene_err_1_title AS (SELECT DISTINCT s1.omim_gene_oid, s1.source_id,
s1.source_name, s1.reference_file,  s1.feature_type, s1.taxonomy_id, s2.gene_title, s2.symbol,
s1.creation_date, s1.creation_author, s2.is_obsolete, s1.status, s1.comments,
s1.mouse_correlate, s1.year, s1.month, s1.day, s1.sequence_gene_type,
s1.cytogenetic_location, s1.chromosome, s1.map_entry_number,
s1.clinical_synopsys_creation_date, s1.clinical_synopsys_creation_author FROM
log.omim_gene_err_1_notitle AS s1 JOIN log.omim_gene_err_1 AS s2 ON
s1.source_id=s2.source_id AND s1.source_name=s2.source_name WHERE s2.reference_file IN
(SELECT reference_file_id FROM metadata.source_reference_file WHERE imported_file_xml_id
= 'omim_omimData_File'))
```

Figura 77 – Query utilizzata per la creazione della tabella omim_gene_err_1_title

9. Inserimento in *omim_gene* dei record di *omim_gene_err_1_title*.

```
INSERT INTO omim_gene (SELECT * FROM log.omim_gene_err_1_title)
```

Figura 78 – Query utilizzata per l'inserimento delle entries della tabella omim_gene_err_1_title nella tabella omim_gene

10. Applicazione della procedura per l'eliminazione/emerge duplicati su *omim_gene* con suffisso "_3".

11. Aggiornamento dei valori dei campi FK nelle tabelle secondarie di *omim_gene*.

12. Rimozione dalle tabelle secondarie di *omim_gene* delle tuple con valori dei campi FK non presenti in tabella principale *omim_gene* (e loro inserimento in tabelle **_deleted_sec*).

13. Creazione della tabella *omim_gene_deleted_entries*, nella quale sono inserite le tuple il cui valore del campo *source_id* non è presente nella tabella *omim_gene*, cioè geni

descritti in OMIM che non risultano importati all'interno del data warehouse. Se la procedura ha avuto esito positivo tale tabella non deve contenere entries.

```
CREATE TABLE log.omim_gene_deleted_entries AS (SELECT a.*, b.imported_file_name FROM
log.omim_gene_err_1 AS a JOIN metadata.source_reference_file AS b on a.reference_file =
b.reference_file_id WHERE a.source_id NOT IN (SELECT source_id FROM  omim_gene))
```

Figura 79 – Query utilizzata per la creazione della tabella omim_gene_deleted_entries

14. Applicazione dei punti precedenti (1-13) sulla tabella *omim_disorder*.
15. Applicazione della procedura per l'eliminazione/emerge duplicati sulla tabella *omim_clinicals_synopsys* .
16. Aggiornamento dei valori dei campi FK nelle tabelle secondarie di *omim_clinical_synopsys*.
17. Applicazione della procedura *DuplicatesChecker* su tutte le tabelle secondarie di OMIM (tabelle di sorgente secondarie di *omim_gene* e *omim_disorder*) e sulle tabelle **_relationship* importate.
18. Abilitazione PK, FK e U su tutte le tabelle sorgenti di OMIM (in un'altra classe *OmimEnableConstraints.java*).

10 Validazione e testing

10.1 Errori e inconsistenze riscontrate nei dati importati

In questo sottocapitolo sono raggruppate tutte le anomalie riscontrate durante l'importazione dei file delle banche dati descritte in precedenza e le soluzioni tecniche adottate per gestirle .

10.1.1 Banca dati ExPASy ENZYME

- **File enzyme.dat**

Non si riscontrano anomalie durante l'importazione dei file di sorgente della banca dati ExPASy ENZYME. Tuttavia si è scelto di segnalare nel log del programma alcune situazioni particolari in modo che l'amministratore possa verificare che non vi siano errori nei dati importati:

- importazione di EC number per i quali non esiste una classe di enzimi di appartenenza;
- matching di sottostringhe del campo *CC* con i pattern di similarità elencati nel nono capitolo, alle quali non seguono uno o più EC number.

10.1.2 Banca dati OMIM

- **File omim.txt**

Di seguito è esposta una panoramica sulle problematiche riscontrate durante l'importazione del file e le strategie implementate per ovviare a simili problematiche.

Importazione dati **FIELD* ED* e **FIELD* EDcs*:
Come mostra l'esempio di figura 80, le righe 34201 e 34203 coincidono, ciò generava errori di chiave duplicata nella tabella corrispondente, *omim_gene_edit_history* o *omim_disorder_edit_history,* secondo la tipologia di dato importato. Per questo motivo è stato aggiunto il campo *ord,* appartenente al vincolo di unicità, che rappresenta un indice autoincrementante gestito in fase d'importazione.

```
34171 *FIELD* ED
34172 wwang: 01/27/2010
34173 ckniffin: 1/13/2010
34174 ckniffin: 11/30/2009
34175 wwang: 11/23/2009
34176 wwang: 11/20/2009
34177 ckniffin: 10/27/2009
34178 alopez: 10/11/2007
34179 terry: 10/3/2007
34180 alopez: 2/20/2007
34181 terry: 2/19/2007
34182 carol: 4/4/2006
34183 ckniffin: 1/6/2006
34184 wwang: 8/10/2005
34185 terry: 8/8/2005
34186 ckniffin: 6/16/2005
34187 carol: 4/21/2003
34188 tkritzer: 12/27/2002
34189 terry: 12/23/2002
34190 tkritzer: 11/19/2002
34191 tkritzer: 7/29/2002
34192 tkritzer: 7/26/2002
34193 terry: 7/22/2002
34194 terry: 7/17/2002
34195 terry: 7/10/2002
34196 carol: 12/14/2001
34197 mcapotos: 12/4/2001
34198 alopez: 10/26/1999
34199 terry: 9/24/1999
34200 dkim: 6/26/1998
34201 mark: 3/13/1996
34202 terry: 3/13/1996
34203 mark: 3/13/1996
```

Figura 80 – Esempio di record uguali in omim.txt - *FIELD* ED

Importazione dati **FIELD* CN* e **FIELD* CNs*:

In figura 81 è evidenziata la presenza di righe identiche all'interno del campo **FIELD* CN*: le righe 1008524 e 1008526 presentano gli stessi valori, generando, in fase di abilitazione degli indici, errori di chiave duplicata nella tabella corrispondente *omim_gene_contributors* o *omim_disorder_contributors*. Anche in questo caso è stato aggiunto il campo *ord*, che come indicato in precedenza, viene gestito in fase d'importazione.

```
1008516 *FIELD* CN
1008517 Matthew B. Gross - updated: 1/30/2009
1008518 Patricia A. Hartz - updated: 1/30/2009
1008519 Marla J. F. O'Neill - updated: 11/19/2008
1008520 Marla J. F. O'Neill - updated: 3/6/2008
1008521 Marla J. F. O'Neill - updated: 2/7/2005
1008522 Marla J. F. O'Neill - updated: 10/22/2004
1008523 Ada Hamosh - updated: 7/7/2003
1008524 Victor A. McKusick - updated: 5/21/2003
1008525 Denise L. M. Goh - updated: 5/21/2003
1008526 Victor A. McKusick - updated: 5/21/2003
1008527 Denise L. M. Goh - updated: 4/18/2003
1008528 Victor A. McKusick - updated: 5/13/2002
1008529 Paul Brennan - updated: 3/11/2002
1008530 Ada Hamosh - updated: 9/13/2000
1008531 Victor A. McKusick - updated: 5/28/1998
1008532 Victor A. McKusick - updated: 8/1/1997
1008533 Mark H. Paalman - edited: 1/23/1997
1008534 Alan F. Scott - updated: 9/16/1996
1008535
```

Figura 81 – Esempio di record uguali in omim.txt - *FIELD* CN

Inoltre è stata riscontrata la presenza di alcune entry del campo **FIELD* CN* in cui non erano presenti né il nome né l'azione eseguita; dato che l'informazione rimanente risultava di poco valore si è scelto di non importare tali entries e di segnalare l'anomalia nel log del programma.

Importazione dati **FIELD* CS*
Sono stati riscontrati nel file alcuni record aventi le righe di testo del campo **FIELD* CS* non formattate correttamente, rendendone impossibile il parsing. Anche in questo caso viene segnalata l'anomalia nel log dell'applicazione.

- **File genemap**

Campo18- Reference
Secondo quanto riportato nel file *genemap.key*, ogni record contenuto nel file *genemap* è costituito da 18 campi. In realtà, quando in un record non è indicato alcun valore per il campo *reference*, che dovrebbe rappresentare il diciottesimo campo, esso non è presente, nemmeno con valore *NULL*, e il record è costituito da 17 campi.
Il Parser utilizzato è strutturato in modo da eseguire un controllo sul numero di campi da estrarre e quelli effettivamente individuati nel file, la mancata corrispondenza di tali valori genera delle eccezioni, che vengono gestite generando un messaggio di errore nel log dell'applicazione.
Dato che questa situazione è prevista, si è scelto di ridefinire il metodo che si occupa di eseguire questo controllo in modo che non siano visualizzati nel log tali messaggi di warning.

[Campo14- Campo15- Campo16] - Deseases
Da questi tre campi si recupera un elenco di elementi che identificano il nome dei disturbi; oltre al testo è indicato anche il codice identificativo Mim number assegnato da OMIM. Sebbene nella maggior parte dei casi quest'ultimo sia presente, sono state evidenziate situazioni di record che non riportato nessun identificativo Mim number ma solo il nome definito per il disurbo. In tal caso si è deciso di utilizzare la seguente strategia: si assume che il codice associato alla malattia sia quello indicato nel *Campo10 Mim number* e s'importano i dati associandoli a questo codice. Se sono presenti più disturbi senza codice identificativo assegnato, si processa e s'importa solo il primo assegnandogli il *Mim number* del *Campo10*, mentre i successivi vengono trascurati, segnalando l'anomalia nel log dell'applicazione.
Inoltre sono state rilevate alcune entry aventi il campo *Deseases* mal formattato, in quanto vi era la presenza di parentesi graffe aperte che non venivano chiuse. In questo caso il *campo desease_type* della tabella *omim_disorder* viene posto a valore *NULL* e si segnala l'errore nel log dell'applicazione.

10.2 Quantificazione dei dati importati e delle tempistiche d'importazione

Di seguito sono presentate le tabelle contenenti le tempistiche d'importazione e la quantificazione dei dati importati(*) dalle banche dati oggetto della presente Tesi. I test sono stati eseguiti su una macchina con processore Intel Core Duo E8400 da 3Ghz e disco Hitachi a 5600RPM e 8Mb di cache.

Banca Dati	Nome del Parser	Tabelle popolate	Numero di entries*	Tempo di esecuzione
ExPASy ENZYME	EnzymeClassLoader	2	594	1 sec
	EnzymeDataLoader	9	223.979	51 sec
OMIM	OmimLoader	23	818.190	3.887 sec
	GenemapKeyLoader	2	34	< 1 sec
	GenemapLaoder	9	51.779	118 sec
	MorbidmapLoader	5	5.597	77 sec
	PubmedLoader	2	149.048	168 sec

Tabella 7 – Totale entries importate e tempistiche

Banca dati	Nome tabella database	Numero di entries*
ExPASy ENZYME	enzyme2protein_fam_dom_imported	1.342
	enzyme_history_imported	891
	enzyme_similarity_imported	51
	expasy_enzyme	4.630
	expasy_enzyme_action	531
	expasy_enzyme_alternative_name	8.311
	expasy_enzyme_comment	6.846
	expasy_enzyme_relationship	4.620
	protein2enzyme_imported	197.351
OMIM	gene2clinical_synopsys_imported	752
	gene2genetic_disorder_imported	23.531
	gene2publication_reference_imported	92.889
	gene_history_imported	545
	genetic_disorder2clinical_synopsys_imported	40.584

	genetic_disorder2publication_reference_imported	55.337
	genetic_disorder_history_imported	493
	omim_clinical_synopsys	98
	omim_clinical_synopsys_relationship	63
	omim_clinical_synopsys_subgroup	117
	omim_disorder	7.155
	omim_disorder_alternative	10.976
	omim_disorder_clinical_synopsys_contributors	2.019
	omim_disorder_clinical_synopsys_edit_history	6.388
	omim_disorder_contributors	19.828
	omim_disorder_edit_history	87.243
	omim_disorder_method	1.314
	omim_disorder_reference_publication	65.344
	omim_disorder_relationship	11.847
	omim_disorder_text	21.607
	omim_gene	14.149
	omim_gene_allelic_variant	19.127
	omim_gene_alternative	20.380
	omim_gene_clinical_synopsys_contributors	43
	omim_gene_clinical_synopsys_edit_history	82
	omim_gene_contributors	40.414
	omim_gene_edit_history	134.598
	omim_gene_method	18.638
	omim_gene_reference_publication	99.457
	omim_gene_relationship	67.504
	omim_gene_text	53.306
	phenotype_4_gene2clinical_synopsys_imported	1.389
	phenotype_4_genetic_disorder2clinical_synopsys_imported	69.787

Tabella 8 – Totale entries suddivise per tabella

* Dopo l'applicazione della procedura per l'eliminazione/merge delle tuple duplicate.

11 Conclusioni

Il risultato di questa Tesi è l'implementazione di una architettura software e di procedure automatizzate per l'importazione e l'integrazione di dati genomici e proteomici forniti da banche dati con caratteristiche differenti tra di loro. Esso si integra quindi nel progetto GPDW e concorre nella realizzazione finale di un data warehouse integrato e consistente, che sarà successivamente messo a disposizione dei ricercatori con l'aggiunta di dati di funzionalità proteica e patologie geniche.
Il lavoro è stato suddiviso in tre stadi principali:

1. analisi dell'architettura esistente del framework con lo scopo di individuare eventuali errori progettuali e componenti ottimizzabili;
2. definizione e implementazione delle procedure di importazione e integrazione automatiche. In particolare, lo sviluppo software può essere distinto in due fasi:
   a) astrazione delle procedure automatiche d'importazione, che ha richiesto la ristrutturazione del file di configurazione *data_sources.xml* e la realizzazione dei componenti: Importer di sorgente generico, Loader dati generico e Importer dati specifici (di sorgente, external reference, relationship, history, similarity e associazione);
   b) implementazione della procedura per l'eliminazione/merge delle tuple duplicate applicabile in maniera automatica a una qualsiasi tabella presente nel data warehouse;
3. applicazione delle procedure implementate all'importazione nel data warehouse dei dati forniti dalle banche dati ExPASy ENZYME e OMIM considerate all'interno del progetto GPDW; questo punto si è sviluppato in diverse fasi di seguito elencate:
   a) analisi dei file dati di ogni sorgente, descrizione concettuale e logica dei dati e progettazione del modello relazionale per ogni banca dati;
   b) sviluppo di Parser, Loader e Importer di dati specifici, estendendo le classi generiche, per l'estrazione dei dati dalle sorgenti considerate e loro importazione automatizzata nel data warehouse; le procedure per l'importazione sono state progettate tenendo conto degli obbiettivi e dei requisiti definiti, considerando la necessità di sviluppare funzionalità che possano essere ereditate e riutilizzate;
   c) integrazione con l'architettura software preesistente e modifica della struttura dei file di configurazione XML per l'importazione dei nuovi dati;
   d) utilizzo del software realizzato per la creazione di nuove parti del data warehouse GPDW e importazione in esse dei dati considerati;
   e) testing e verifica, questa fase prevede la quantificazione dei dati importati, degli errori e delle inconsistenze rilevati dalle procedure automatiche di verifica

implementate, e la valutazione dei tempi impiegati per l'importazione e il controllo di tali dati.

L'utilizzo delle classi generiche astratte sviluppate riduce l'impegno richiesto per la definizione di codice specifico, in precedenza necessario per l'importazione di dati da nuove sorgenti dati.

La metodologia d'importazione automatica dei dati implementata ha permesso l'integrazione, in modo più efficiente ed efficace, di un numero maggiore di banche dati, e conseguentemente di entità biomediche e biomolecolari, nonché ha consentito un miglior controllo automatico dei dati importati. Questo consente l'aumento delle informazioni nel data warehouse e un consolidamento della qualità dei dati importati. Premesso che se la qualità dei dati integrata è da sempre legata alla qualità dei dati forniti dalle fonti dati originali, tale aspetto può essere rafforzato con il controllo automatico di correttezza di tutte le operazioni effettuate sui dati importati e la verifica dell'esattezza dei dati importati da ciascuna sorgente con riferimenti incrociati fra i dati delle diverse fonti. Il risultato dei controlli di qualità può consentire di rivelare incongruenze o informazioni mancanti nelle banche dati pubbliche, in modo da sostenere il miglioramento della qualità delle informazioni genomiche e proteomiche a disposizione di tutta la comunità scientifica.

12 Sviluppi futuri

Il Virtual Bioinformatic Lab è un progetto che mira a integrare e accorpare il maggior numero di banche dati biomolecolari presenti sul Web. All'interno di questa Tesi è stata concentrata l'attenzione sulla ristrutturazione dell'architettura del framework GPDW aumentando il grado di astrazione delle procedure automatiche d'importazione e sull'applicazione di tali modifiche a due nuove banche dati. Il progetto proseguirà in questa direzione con l'intento di aumentare il numero di banche dati integrate.

La metodologia d'importazione dei dati oggetto di questa Tesi, attualmente applicata alle sole sorgenti ExPASy ENZYME e OMIM, sarà estesa a tutte le banche dati considerate all'interno del progetto GPDW, in modo che l'intero processo di importazione e integrazione dei dati segua un unico standard processuale.

Un ulteriore sviluppo può essere individuato nella realizzazione di un unico file di configurazione XML in cui racchiudere quanto adesso viene definito all'interno dei file *data_sources.xml* e *feature_definition.xml,* facilitando le operazioni di configurazione del framework GPDW e il controllo incrociato delle informazioni in tali attuali due file di configurazione.

Inoltre, è prevista la realizzazione di un modulo dedito al download dei file forniti dalle diverse banche dati considerate all'interno del progetto, che consentirà di ottenere informazioni costantemente aggiornate.

Nel prossimo futuro sarà fornito al pubblico l'accesso al data warehouse. L'idea è di fornire diversi servizi Web che permetteranno la classica interrogazione bioinformatica, in modo da sostenere gli scienziati e i ricercatori nell'analisi di correlazione dei dati prodotti dai loro esperimenti.

13 Bibliografia

[1] Masseroli M. Biomolecular databanks. Bioinformatica e biologia computazionale per la medicina molecolare 2009. Available from: http://www.bioinformatics.polimi.it/masseroli/bbcmm/dispense/9_BiomolecularDataBanks.pdf

[2] Bairoch A., Apweiler R., Wu CH., Barker WC., Boeckmann B., Ferro S. The universal protein resource (uniprot). Nucleic Acid Research 2005; 33: 154-159. Available from: http://nar.oxfordjournals.org/cgi/reprint/33/suppl_1/D154

[3] De Bakey ME. The National Library of Medicine. Evolution of a premier information center. JAMA 1991; 266:1252-1258.

[4] Hamosh A., Scott AF., Amberger JS., Bocchini CA., McKusick VA. Online mendelian inheritance in man (Omim), a knowledgebase of human genes and genetic disorders. Nucleic Acids Research 2005; 33: 514–517. Available from: http://nar.oxfordjournals.org/cgi/content/full/33/suppl_1/D514

[5] Matthews L., Gopinath G., Gillespie M., Caudy M., Croft D., de Bono B., Garapati P., Hemish J., Hermjakob H., Jassal B., Kanapin A., Lewis S., Mahajan S., May B., Schmidt E., Vastrik I., Wu G., Birney E., Stein L., D'Eustachio P. Reactome knowledgebase of human biological pathways and processes. Nucleic Acids Research 2009; 37: 619-622. Avaible from: http://www.reactome.org

[6] Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium Nature Genet 2000; 25: 25-29.

[7] Adams MD. The genome sequence of drosophila melanogaster. Science 2000; 287: 2185-2195.

[8] Bairoch A., Apweiler R., Wu CH., Barker WC., Boeckmann B., Ferro S. The universal protein resource (uniprot). Nucleic Acid Research 2005; 33: 154-159. Available from: http://nar.oxfordjournals.org/cgi/reprint/33/suppl_1/D154

[9] Liu M., Liberzon A., Kong SW., Lai WR., Park PJ., Kohane IS. Network-based analysis of affected biological processes in type 2 diabetes models. PLoS Genetics 2007; 3.

[10] Atzeni P., Ceri S., Fraternali P., Paraboschi S., Torlone R. Basi di dati: architetture e linee di evoluzione. McGrawHill 2007.

[11] Atzeni P., Ceri S., Paraboschi S., Torlone R. Basi di dati modelli e linguaggi di interrogazione. McGrawHill 2006.

[12] Dove A. Proteomics: translating genomics into products?. Nature biotechnology 1999; 17: 223-6. Available from: http://cmbi.bjmu.edu.cn/cmbidata/proteome/reviews/03.pdf

[13]Masseroli M., Ceri S., Tettamanti L., Campi A. Model-driven modular integration of genomic and proteomic information. IEEE Transactions on Knowledge and Data Engineering 2009; 1-9.

[14]Masseroli M. Tassonomia file dati di annotazioni genomiche e proteomiche. Bioinformatica e biologia computazionale per la medicina molecolare 2009. Available from: http://www.bioinformatics.polimi.it/masseroli/bbcmm/dispense/esercitazioni/E4_TassonomiaFileDati.pdf

[15]Karp PD. What we do not know about sequence analysis and sequence databases. BMC Bioinfomatics 1998; 14: 753-754.

[16]International Human Genome Sequencing Consortium. Finishing the euchromatic sequence of the Homo genome. Nature 2004; 431: 931-945.

[17]Arnold K., Gosling J., Holmes D. The java programming language, fourth edition. Prentice Hall 2006.

[18]Binato A., Fuggetta A., Sfardini L. Ingegneria del software: creatività e metodo. Pearson Education 2006.

[19] PCRE - Perl Compatible Regular Expressions. 2011. Available from: http://www.pcre.org/pcre.txt

[20]Gasteiger E., Gattiker A., Hoogland C., Ivanyi I., Appel R. D. and Bairoch A. ExPASy: the proteomics server for in-depth protein knowledge and analysis. Nucleic Acids Research 2003. Avaible from: http://nar.oxfordjournals.org/content/31/13/3784.full.pdf

[21]Boeckmann B., Bairoch A., Apweiler R., Blatter MC., Estreicher A., Gasteiger E., The swiss-prot protein knowledgebase and its supplement. Nucleic Acid Research 2003; 31:365-370. Available from: http://nar.oxfordjournals.org/cgi/reprint/31/1/365?ijkey=4HTKis2UlymOA&keytype=ref&siteid=nar

[22]Masseroli M., Galati O., Pinciroli F. GFINDER: genetic disease and phenotype location statistical analysis and mining of dynamically annotated gene lists. 2005; 1-10

14 Appendice

14.1 Mapping tra campi in file dati e in base dati creata

S'inserisce in questo capitolo il mapping tra i campi estratti dai file forniti delle banche dati ExPASy ENZYME e OMIM, descritti nel capitolo 9, e i campi delle tabelle che vanno a popolare. I campi che in tabella non sono mappati non sono associati a nessun dato estratto e non vengono popolati con valori letti direttamente dal file.

14.1.1 Banca dati ExPASy ENZYME

- **File enzclass.txt**

Le tabelle 9 e 10 mostrano l'abbinamento tra i campi estratti dal file *enzclass.txt* e i campi del database.

Campo estratto	Campo file	Campo database
		expasy_enzyme_oid
EC number	1. -. -.-	source_id
		source_name
		reference_file
		feature_type
Name	Oxidoreductases.	name
		catalytic_activity
		cofactor
		term_position

Tabella 9 – Mapping per i campi della tabella expasy_enzyme

I campi della tabella *expasy_enzyme* assumono i seguenti valori:

1. expasy_enzyme_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: l'identificativo univoco EC number generato da ExPASy ENZYME;
3. source_name: il nome codificato della sorgente dati che fornisce l'identificativo, vale a dire *expasy_enzyme*;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzclass.txt*;
5. feature_type: campo codificato dai metadati, restituisce il tipo di feature che si sta importando, in questo caso enzimi;

6. name: il nome associato all'ID, fornito da ExPASy ENZYME, che si sta importando;
7. catalytic_activity: attività catalitica dell'enzima che si sta analizzando. Viene settato a *NULL* in quanto non fornito dal file sorgente;
8. cofactor: cofattori enzimatici associati all'enzima analizzato. Viene settato a *NULL* in quanto non fornito dal file sorgente;
9. term_position: campo codificato, restituisce la posizione dell'enzima all'interno della struttura gerarchica *("Root", "Leaf„ "Disconnect", "Other")*.

Campo estratto	Campo file	Campo database
		term_oid
		related_term_oid
		relationship_type
		inferred
		reference_file

Tabella 10 – Mapping per i campi della tabella expasy_enzyme_relationship

I campi della tabella *expasy_enzyme_relationship* assumono i seguenti valori:

1. term_oid: l'identificativo assegnato dalla procedura di importazione all'enzima figlio;
2. related_term_id: l'identificativo assegnato dalla procedura di importazione all'enzima padre;
3. relationship_type: tipo della relazione, nel caso in esame *"IS_A"*;
4. inferred: questo campo non viene popolato durante l'importazione di ExPASy ENZYME;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzclass.txt*.

- **File enzyme.dat**

Di seguito, dalla tabella 11 alla tabella 19, sono rappresentati gli abbinamenti tra i campi estratti dal file *enzyme.dat* e i campi delle tabelle create all'interno del data warehouse.
La tabella 11 mostra il mapping per i campi della tabella *expasy_enzyme* del database.

Campo estratto	Campo file	Campo database
		expasy_enzyme_oid
EC number	ID	source_id
		source_name
		reference_file
		feature_type

NC-IUB name	DE	name
Catalytic_activity	CA	catalytic_actvity
Cofactor	CF	cofactor
		term_position

Tabella 11 – Mapping per i campi della tabella expasy_enzyme

1. expasy_enzyme_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: l'identificativo univoco EC number fornito da ExPASy ENZYME;
3. source_name: il nome codificato della sorgente dati che fornisce l'dentificativo, vale a dire *expasy_enzyme*;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*;
5. feature_type: campo codificato dai metadati, restituisce il tipo di dato che si sta importando, in questo caso enzimi;
6. name: il nome, fornito dalla NC-IUB, associato all'ID che si sta importando;
7. catalytic_activity: attività catalitica dell'enzima che si sta analizzando;
8. cofactor: cofattori enzimatici associati all'enzima analizzato;
9. term_position: campo codificato, restituisce la posizione dell'enzima all'interno della struttura gerarchica *("Root", "Leaf„ "Disconnect", "Other")*.

La tabella seguente mostra il mapping per i campi della tabella *expasy_enzyme_alternative_name* del database.

Campo estratto	Campo file	Campo database
		expasy_enzyme_oid
Nome alternativo	AN	alternative_name
		reference_file

Tabella 12 – Mapping per i campi della tabella expasy_enzyme_alternative_name

1. expasy_enzyme_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *expasy_enzyme*;
2. alternative_name: nome alternativo a quello fornito dalla NC-IUB, utilizzato in letterature per descrivere l'enzima;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*.

La tabella seguente mostra il mapping per i campi della tabella *expasy_enzyme_action* del database.

Campo estratto	Campo file	Campo database
		expasy_enzyme_oid
Azione	CC	action
		reference_file

Tabella 13 – Mapping per i campi della tabella expasy_enzyme_action

1. expasy_enzyme_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *expasy_enzyme*;
2. action: campo codificato che rappresenta l'azione associata all'enzima importato;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*.

In tabella 14 è mostrato il mapping per i campi della tabella *expasy_enzyme_comment* del database.

Campo estratto	Campo file	Campo database
		expasy_enzyme_oid
Commento	CC	comment
		reference_file

Tabella 14 – Mapping per i campi della tabella expasy_enzyme_comment

1. expasy_enzyme_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *expasy_enzyme*;
2. comment: stringa di commento, fornisce informazioni utili sull'enzima che si sta analizzando;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*.

Campo estratto	Campo file	Campo database
		term_oid
		related_term_oid
		relationship_type
		inferred
		reference_file

Tabella 15 – Mapping per i campi della tabella expasy_enzyme_relationship

I campi della tabella *expasy_enzyme_relationship* assumono i seguenti valori:

1. term_oid: l'identificativo assegnato dalla procedura di importazione all'enzima figlio;
2. related_term_id: l'identificativo assegnato dalla procedura d'importazione all'enzima padre, fornito dal file *enzclass.txt*. Ad esempio all'enzima avente ID *"1.1.1.1"* sarà associato l'enzima *"1.1.1"*.
3. relationship_type: tipo della relazione, viene popolato con il valore codificato *"IS_A"*;
4. inferred: questo campo non viene popolato durante l'importazione di ExPASy ENZYME;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*.

La tabella 16 mostra il mapping per i campi della tabella *enzyme_history_imported* del database.

Campo estratto	Campo file	Campo database
EC number obsoleto	ID	enzyme_discontinued_id
		source_name
		reference
		reference_file
EC number corrente	DE	enzyme_id
		history_type

Tabella 16 – Mapping per i campi della tabella enzyme_history_imported

1. enzyme_discontinued_id: l'identificativo univoco EC number del record obsoleto fornito da ExPASy ENZYME;
2. source_name: il nome codificato della sorgente dati che fornisce gli ID sorgenti, vale a dire *expasy_enzyme*;
3. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso coincide con source_name;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*;
5. enzyme_id: campo opzionale contenente l'identificativo dell'enzima che sostituisce l'entry obsoleta;
6. history_type: campo codificato contenente la tipologia di history, in questo caso assume valore *"REPLACED_BY"*.

La tabella *enzyme_similarity_imported* è popolata a partire dai dati estratti dal campo *CC*. Di seguito ne viene mostrato il mapping dei campi.

Campo estratto	Campo file	Campo database
EC number	ID	enzyme1_id
		source1_name
EC number simile	CC	enzyme2_id
		source2_name
		reference
		reference_file
		similarity_type

Tabella 17 – Mapping per i campi della tabella enzyme_similarity_imported

1. enzyme1_id: l'identificativo univoco EC number dell'enzima che si sta analizzando;
2. source1_name: il nome codificato della sorgente dati che fornisce l'identificativo enzyme1_id, vale a dire *expasy_enzyme*;
3. enzyme2_id: l'identificativo univoco EC number dell'enzima simile all'enzima che si sta analizzando;
4. source2_name: il nome codificato della sorgente dati che fornisce l'identificativo enzyme2_id, vale a dire *expasy_enzyme*;
5. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso *expasy_enzyme*;
6. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*;
7. similarity_type: campo codificato contente la tipologia di similarità, in questo caso *"ALIAS"*.

La tabella 18 mostra il mapping per i campi della tabella *enzyme2prot_fam_dom_imported* del database.

Campo estratto	Campo file	Campo database
		annotation_oid
EC number	ID	enzyme_id
		origin_data_source
Prosite access number	PR	prot_fam_dom_id
		destination_data_source
		reference
		reference_file
		association_type

Tabella 18 – Mapping per i campi della tabella enzyme2prot_fam_dom_imported

I dati che si stanno importando indicano una relazione tra due sorgenti, per questo motivo sono presenti sia gli identificatori forniti dalla sorgente dati che il nome della sorgente dati stessa.

1. annotation_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. enzyme_id: l'identificativo univoco EC number dell'enzima appartenente alla relazione;
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo enzyme_id , in questo caso *expasy_enzyme*;
4. prot_fam_dom_id: l'identificativo univoco *PSITE_Doc_ACNb* (PROSITE access number) della famiglia di proteine appartenente alla relazione;
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo prot_fam_dom_id, in questo caso *prosite*;
6. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso è *expasy_enzyme*;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*;
8. association_type: campo codificato, descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"ANNOTATED_TO"*.

La tabella seguente mostra il mapping per i campi della tabella *protein2enzyme_imported* del database.

Campo estratto	Campo	Campo database
		annotation_oid
UniProtKB/Swiss-	DR	protein_id
		origin_data_source
EC number	ID	enzyme_id
		destination_data_source
		reference
		reference_file
		association_type

Tabella 19 – Mapping per i campi della tabella protein2enzyme_imported

1. annotation_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. protein_id: l'identificativo univoco UniProtKB/Swiss-Prot *AC_Nb* (access number) della proteina appartenente alla relazione;
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo enzyme_id , in questo caso *uniprot*;
4. enzyme_id: l'identificativo univoco EC number dell'enzima appartenente alla relazione;
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo enzyme_id, in questo caso *expasy_enzyme*;
6. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso *expasy_enzyme*;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *enzyme.dat*;
8. association_type: campo codificato che descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"RELATED_TO"*.

14.1.2 Banca dati OMIM

- **File omim.txt**

Di seguito dalla tabella 20 alla tabella 45 sono rappresentati gli abbinamento tra i campi estratti dal file *omim.txt* e i campi delle tabelle del database.
La tabella 20 mostra il mapping per i campi della tabella *omim_gene* del database.

Campo estratto	Campo file	Campo database
		omim_gene_oid
Mim number	*FIELD* NO	source_id
		source_name
		reference_file
		feature_type
		taxonomy_id
Titolo	*FIELD* TI	gene_title
Simbolo	*FIELD* TI	symbol
Data	*FIELD* CD	creation_date
Autore	*FIELD* CD	creation_author

	FIELD TI	is_obsolete
Tipo Record	*FIELD* TI	sequence_gene_type
Data	*FIELD* CDcs	clinical_synopsys_creation_date
Autore	*FIELD* CDcs	clinical_synopsys_creation_author
		term_position

Tabella 20 – Mapping per i campi della tabella omim_gene

1. omim_gene_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: l'identificativo univoco assegnato da OMIM validato tramite l'espressione regolare *[0-9]{6};*
3. source_name: il nome codificato della sorgente dati che fornisce l'identificativo, vale a dire *omim*;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*;
5. feature_type: campo codificato dai metadati che restituisce il tipo di dato che si sta importando, in questo caso geni;
6. taxonomy_id: non è recuperato direttamente dal file ma è impostato come costante e assume valore *"9606",* poichè i dati di OMIM si riferiscono al solo organismo umano;
7. gene_title: titolo del gene;
8. symbol: simbolo associato al gene;
9. creation_date: data in cui è stato inserito il gene nel database di OMIM;
10. creation_author: autore che ha provveduto all'inserimento del gene nella sorgente OMIM;
11. is_obsolete: il campo può assumere valori booleani, si imposta il valore *True* quando il record importato è un record storicizzato;
12. sequence_gene_type: campo codificato che viene elaborato è trasformato in flag.
13. clinical_synopsys_creation_date: data in cui è stata creata la sinossi clinica per il gene importato;
14. clinical_synopsys_creation_author: autore che ha creato la sinossi clinica per il gene importato;

15. term_position: campo codificato che esprime la posizione del gene all'interno dell'eventuale struttura gerarchica in cui è contenuto; non essendo i geni estratti da OMIM coinvolti in relazioni di tipo gerarchico assume il valore "*Disconnect*".

Di seguito saranno mostrate le tabelle collegate alla tabella principale *omim_gene*, indicata in precedenza.
La tabella 21 è compilata in presenza di almeno un sottotitolo alternativo o un simbolo alternativo.

Campo estratto	Campo file	Campo database
		omim_gene_oid
Titolo_alternativo	*FIELD* TI	gene_title
Simbolo_alternativo	*FIELD* TI	symbol
		reference_file

Tabella 21 – Mapping per i campi della tabella omim_gene_alternative

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. gene_title: titolo alternativo del gene;
3. symbol: simbolo alternativo associato al gene;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella seguente mostra il mapping per i campi della tabella *omim_gene_text* del database.

Campo estratto	Campo file	Campo database
		omim_gene_oid
		ord
Titolo	*FIELD* TX	title
Sottotitolo	*FIELD* TX	sub_title
Body	*FIELD* TX	body
		reference_file

Tabella 22 – Mapping per i campi della tabella omim_gene_text

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. ord: è un contatore progressivo gestito in fase di importazione per mantenere ordinata la sequenza dei testi importati;
3. title: campo codificato nel quale viene memorizzato il titolo del testo;

4. sub_title: campo codificato, rappresenta il sottotitolo del testo;
5. body: corpo del testo;
6. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 23 è popolata in presenza del campo **FIELD* ED*.

Campo estratto	Campo file	Campo database
		omim_gene_oid
		ord
Autore	*FIELD* ED	author
Data	*FIELD* ED	date
		reference_file

Tabella 23 – Mapping per i campi della tabella omim_gene_edit_history

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e data uguali;
3. author: nome dell'autore che ha eseguito la modifica;
4. date: data della modifica nel formato MM/GG/AA;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 24 è compilata in presenza del campo **FIELD* CN*.

Campo estratto	Campo file	Campo database
		omim_gene_oid
		ord
Autore	*FIELD* CN	author
Data	*FIELD* CN	date
Azione	*FIELD* CN	action
		reference_file

Tabella 24 – Mapping per i campi della tebella omim_gene_contributors

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore, data e azione uguali;

3. author: nome dell'autore che ha partecipato all'integrazione del database di OMIM;
4. date: data nella quale è stata eseguita l'integrazione del database di OMIM;
5. action: campo codificato, rappresenta la tipologia di azione di integrazione del database OMIM;
6. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 25 è popolata in presenza del campo **FIELD* RF*.

Campo estratto	Campo file	Campo database
		omim_gene_oid
Autore	*FIELD* RF	author
Anno	*FIELD* RF	Year
Numero	*FIELD* RF	ord
Titolo	*FIELD* RF	text
Rivista	*FIELD* RF	text
Numero volume	*FIELD* RF	text
Pagine	*FIELD* RF	text
		reference_file

Tabella 25 – Mapping per i campi della tabella omim_gene_reference_publication

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene;*
2. year: anno della pubblicazione;
3. author: autore della pubblicazione, in presenza di più di un autore si inserisce solo il primo;
4. text: si accorpano in questo campo gli autori esclusi dal campo precedente e i campi *Titolo, Rivista, Numero volume* e *Pagine;*
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 26 è compilata in presenza del campo **FIELD* EDcs*.

Campo estratto	Campo file	Campo database
		omim_gene_oid
		ord
Autore	*FIELD* EDcs	author
Data	*FIELD* EDcs	date
		reference_file

Tabella 26 – Mapping per i campi della tabella omim_gene_clinical_synopsys_edit_history

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e data uguali;
3. author: nome dell'autore che ha eseguito la modifica;
4. date: data della modifica nel formato MM/GG/AA;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 27 è compilata in presenza del campo **FIELD* CNcs*.

Campo estratto	Campo file	Campo database
		omim_gene_oid
		ord
Autore	*FIELD* CNcs	author
Data	*FIELD* CNcs	date
Azione	*FIELD* CNcs	action
		reference_file

Tabella 27 – Mapping per i campi della tabella omim_gene_clinical_synopsys_contributors

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore, data e azione uguali;
3. author: nome dell'autore che ha partecipato all'integrazione del database di OMIM;
4. date: data nella quale è stata eseguita l'integrazione del database di OMIM;
5. action: campo codificato, rappresenta la tipologia di azione di integrazione del database OMIM;
6. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 28 è compilata in presenza del campo **FIELD* AV*, presente solo per i record di geni.

Campo estratto	Campo file	Campo database
		omim_gene_oid
ID	*FIELD* AV	allelic_variant_id
		allelic_variant_sub_id
Title	*FIELD* AV	title
Sottotitolo	*FIELD* AV	subtitle
Simbolo	*FIELD* AV	gene_symbol
Descrizione	*FIELD* AV	short_description
Testo	*FIELD* AV	text
		reference_file

Tabella 28 – Mapping per i campi della tabella omim_gene_allelic_variant

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. allelic_variant_id: l'identificativo della variante allelica;
3. allelic_variant_sub_id: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di più sottotitoli;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 29 viene compilata quando nel campo **FIELD* TX* vi è un riferimento verso un altro un altro gene. Un esempio è indicato in figura 82.

```
*FIELD* TI
*102570 ACTIN, PLATELET
*FIELD* TX
See 102540
```

Figura 82 – Esempio di campo *FIELD* TX contenente un riferimento ad un altro record

Campo estratto	Campo file	Campo database
		term_oid
		related_term_oid
		relationship_type
		inferred
		reference_file

Tabella 29 – Mapping per i campi della tabella omim_gene_relationship

In questo caso si nota come tra il record 102570 e il 102540 sia presente una relazione.

1. term_oid: l'identificativo assegnato dalla procedura di importazione al gene che si sta analizzando (*102570*);
2. related_term_id: l'identificativo assegnato dalla procedura di importazione al gene in relazione con il gene che si sta importando (*102540*);
3. relationship_type: tipo della relazione, nel caso in esame *"IS_RELATED_TO"*;
4. inferred: questo campo non viene popolato durante l'importazione di OMIM;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

I pattern riconosciuti dal parser sono: *see 'MimNumber', (….;'MimNumber'), (….,'MimNumber')* e *('MimNumber')*.

La tabella 30 mostra il mapping per i campi della tabella *omim_disorder* del database.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
Mim number	*FIELD* NO	source_id
		source_name
		reference_file
		feature_type
Titolo	*FIELD* TI	disorder_title
Simbolo	*FIELD* TI	disorder_symbol
Data	*FIELD* CD	creation_date
Autore	*FIELD* CD	creation_author
	FIELD TI	is_obsolete
Tipo record	*FIELD* TI	molecular_basis_type
Data	*FIELD* CDcs	clinical_synopsys_creation_date
Autore	*FIELD* CDcs	clinical_synopsys_creation_author
		term_position

Tabella 30 – Mapping per i campi della tabella omim_disorder

1. omim_disorder_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: l'identificativo univoco assegnato da OMIM, validato tramite l'espressione regolare *[0-9]{6}*;
3. source_name: il nome codificato della sorgente dati che fornisce l'identificativo, vale a dire *omim*;

4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso la sorgente dati è sempre *omim.txt*;
5. feature_type: campo codificato dai metadati, restituisce il tipo di dato che si sta importando, in questo caso genetic disorders;
6. gene_title: titolo del fenotipo;
7. symbol: simbolo associato al fenotipo;
8. creation_date: data in cui è stato inserito il fenotipo nel database;
9. creation_author: autore che ha provveduto all'inserimento del fenotipo in OMIM;
10. is_obsolete: il campo può assumere valori booleani, si imposta il valore *True* quando il record importato è storicizzato;
11. molecular_basis_type: tipo di basi molecolari, viene elaborato è trasformato in flag;
12. clinical_synopsys_creation_date: data in cui è stato creata la sinossi clinica per il fenotipo importato;
13. clinical_synopsys_creation_author: nome dell'autore che ha creato la sinossi clinica per il fenotipo importato;
14. term_position: campo codificato, esprime la posizione del fenotipo all'interno dell'eventuale struttura gerarchica in cui è contenuto; non essendo i fenotipi estratti da OMIM coinvolti in relazioni di tipo gerarchico assume il valore "*Disconnect*".

Di seguito saranno mostrate le tabelle collegate tramite un vincolo di chiave esterna alla tabella principale *omim_disorder*.
La tabella seguente viene compilata in presenza di almeno un sottotitolo alternativo o un simbolo alternativo.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
Titolo_alternativo	*FIELD* TI	disorder_title
Simbolo_alternativo	*FIELD* TI	disorder_symbol
		reference_file

Tabella 31 – Mapping per i campi della tabella omim_disorder_alternative

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. disorder_title: titolo alternativo del genetic disorder;
3. symbol: simbolo alternativo associato al genetic disorder;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

Le tabella 32 viene compilata in presenza del campo **FIELD* TX*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
		ord
Titolo	*FIELD* TX	title
Sottotitolo	*FIELD* TX	sub-title
Body	*FIELD* TX	body
		reference_file

Tabella 32 – Mapping per i campi della tabella omim_disorder_text

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. ord: è un contatore progressivo gestito in fase di importazione per mantenere ordinata la sequenza dei testi importati;
3. title: campo codificato nel quale viene memorizzato il titolo del testo;
4. sub_title: campo codificato, rappresenta il sottotitolo del testo;
5. body: corpo del testo;
6. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

Le tabella 33 viene compilata in presenza del campo **FIELD* ED*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
		ord
Autore	*FIELD* ED	author
Data	*FIELD* ED	date
		reference_file

Tabella 33 – Mapping per i campi della tabella omim_disorder_edit_history

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella tabella *omim_disorder;*
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e data uguali;
3. author: nome dell'autore che ha eseguito la modifica;
4. date: data della modifica nel formato MM/GG/AA;

5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 34 è popolata in presenza del campo **FIELD* CN*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
		ord
Autore	*FIELD* CN	author
Data	*FIELD* CN	date
Azione	*FIELD* CN	action
		reference_file

Tabella 34 – Mapping per i campi della tabella omim_disorder_contributors

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore, data e azione uguali;
3. author: nome dell'autore che ha partecipato all'integrazione del database di OMIM;
4. date: data nella quale è stata eseguita l'integrazione del database di OMIM;
5. action: campo codificato, rappresenta la tipologia di azione di integrazione del database di OMIM;
6. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 35 è compilata in presenza del campo **FIELD* RF*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
Numero	*FIELD* RF	ord
Anno	*FIELD* RF	year
Autori	*FIELD* RF	author
Titolo	*FIELD* RF	text
Rivista	*FIELD* RF	text
Numero volume	*FIELD* RF	text
Pagine	*FIELD* RF	text
		reference_file

Tabella 35 – Mapping per i campi della tabella omim_disorder_reference_pubblication

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. year: anno della pubblicazione;
3. author: autore della pubblicazione. In presenza di più di un autore, si inserisce solo il primo;
4. text: si accorpano in questo campo gli autori esclusi dal campo precedente e i campi *Titolo, Rivista, Numero volume* e *Pagine;*
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 36 è compilata in presenza del campo **FIELD* EDcs*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
		ord
Autore	*FIELD* EDcs	author
Data	*FIELD* EDcs	date
		reference_file

Tabella 36 – Mapping per i campi della tabella omim_disorder_clinical_synopsys_edit_history

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e data uguali;
3. author: nome dell'autore che ha eseguito la modifica;
4. date: data della modifica nel formato MM/GG/AA;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 37 è compilata in presenza del campo **FIELD* CNcs*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
		ord
Autore	*FIELD* CNcs	author
Data	*FIELD* CNcs	date
Azione	*FIELD* CNcs	action
		reference_file

Tabella 37 – Mapping per i campi della tabella omim_disorder_clinical_synopsys_contributors

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e data uguali;
3. author: nome dell'autore che ha eseguito la modifica;
4. date: data della modifica nel formato MM/GG/AA;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 38 è compilata seguendo le modalità indicate per la tabella 29 illustrata per *omim_gene_relationship*.

Campo estratto	Campo file	Campo database
		term_oid
		related_term_oid
		relationship_type
		inferred
		reference_file

Tabella 38 – Mapping per i campi della tabella omim_disorder_relationship

I riferimenti verso altri Mim number estratti dal campo **FIELD* TX*, oltre i casi già esemplificati, possono essere di un gene verso un fenotipo o di un fenotipo verso un gene. In questi casi viene popolata la tabella di associazione tra feature *gene2genetic_disorder_imported* mostrata in tabella 39.

Campo estratto	Campo file	Campo database
		annotation_oid
Mim number	*FIELD* NO	gene_id
		origin_data_source
		genetic_disorder_id
		destination_data_source
		reference
		reference_file
		association_type
		taxonomy_id

Tabella 39 – Mapping per i campi della tabella gene2genetic_disorder_imported

1. annotation_oid: l'identificativo univoco assegnato dalla procedura d'importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. gene_id: rappresenta il codice univoco assegnato al gene da OMIM, prelevato dal campo**FIELD* NO,* già utilizzato per popolare la tabella principale *omim_gene*. Il codice recuperato per essere importato deve essere validato dall'espressione regolare: *[0-9]{6};*
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo gene_id, vale a dire *omim;*
4. genetic_disorder_id: rappresenta il codice univoco assegnato al gene da OMIM, prelevato dal **FIELD* NO* già utilizzato per popolare la tabella principale *omim_genetic_disorder*, il codice recuperato per essere importato deve essere validato dall'espressione regolare *[0-9]{6};*
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo genticic_disorder_id, in questo caso è *omim*;
6. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso la sorgente dati è sempre *omim*;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso la sorgente dati è sempre *omim.txt;*
8. association_type: campo codificato che descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"ANNOTATED_TO";*
9. taxonomy_id: è impostato con il valore *"9606",* perché i dati di OMIM sono riferiti solo all'organismo umano.

La tabella 40 mostra il mapping per i campi della tabella *omim_clinical_synopsys* del database.

Campo estratto	Campo file	Campo database
		omim_clinical_synopsys_oid
		source_id
		source_name
		reference_file
		feature_type
Titolo/Sottotitolo	*FIELD* CS	name
		term_position

Tabella 40 – Mapping per i campi della tabella omim_clinical_synopsys

1. omim_clinical_synopsys_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: questo campo di norma indica l'identificativo fornito dalla sorgente, OMIM non codifica questi valori, quindi vista l'obbligatorietà della compilazione del campo, si è optato per l'inserimento di un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
3. source_name: il nome codificato dai metadati della sorgente dati che fornisce la feature, vale a dire *omim;*
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt;*
5. feature_type: campo codificato dai metadati, restituisce il tipo di dato che si sta importando, in questo caso clinical synopsys;
6. name: in questo campo sono caricati i titoli e i sottotitoli estratti dal campo **FIELD* CS*, nel caso il valore da importare sia un sottotitolo è necessario compilare anche la tabella *omim_clinical_synopsys_relationship* indicata sotto, le due tabelle risultano legate dalla relazione padre/figlio;
7. term_position : campo codificato, restituisce la posizione dell'entry all'interno della struttura gerarchica *("Root", "Leaf,, "Disconnect", "Other")*.

La tabella 41 mette in relazione un'entry di tipo clinical_synopsys estratta dal titolo di un campo **FIELD* CS* con un'entry sempre di tipo clinical_synopsys estratta da un sottotitolo dello stesso campo **FIELD* CS*.

Campo estratto	Campo file	Campo database
		term_oid
		related_term_oid
		relationship_type
		inferred
		reference_file

Tabella 41 – Mapping per i campi della tabella omim_disorder_relationship

1. term_oid: assume il valore del campo omim_clinical_synopsys_oid del record padre inserito nella tabella *omim_clinical_synopsys*, il record padre è costituito dal titolo;
2. related_tem_oid: assume il valore del campo omim_clinical_synopsys_oid del record figlio inserito nella tabella *omim_clinical_synopsys*, il record figlio è costituito dal sottotitolo;

3. relationship_type: campo codificato, esprime la tipologia della relazione che lega i due campi term_oid e related_term_oid; per le relazioni tra clinical_synopsys assume il valore "*IS_A*";
4. inferred: questo campo non viene popolato durante l'importazione di OMIM;
5. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 42 viene popolata in base a quanto definito nella struttura XML descritta nel paragrafo 9.2.6.

Campo estratto	Campo file	Campo database
		omim_clinical_synopsys_oid
		sub_group_name

Tabella 42 – Mapping per i campi della tabella omim_clinical_synopsys_subgroup

1. omim_clinical_synopsys_oid_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_clinical_synopsys*;
2. sub_group_name: campo codificato, contiene il nome del gruppo cui appartiene l'entry rappresentata dall'oid memorizzato in tabella.

La tabella 43 mostra il mapping per i campi della tabella *gene2clinical_synopsys_imported* del database.

Campo estratto	Campo file	Campo database
		annotation_oid
Mim number	*FIELD* NO	gene_id
		origin_data_source
		clinical_synopsys_id
		destination_data_source
		reference
		reference_file
		association_type
		taxonomy_id

Tabella 43 – Mapping per i campi della tabella gene2clinical_synopsys_imported

1. annotation_oid: l'identificativo univoco assegnato dalla procedura d'importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;

2. gene_id: rappresenta il codice univoco assegnato al gene da OMIM, prelevato dal campo **FIELD* NO,* utilizzato per popolare la tabella principale *omim_gene*. Il codice recuperato per essere importato deve essere validato dall'espressione regolare: *[0-9]{6};*
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo gene_id, vale a dire *omim*;
4. clinical_synopsys_id: contiene il valore dal campo source_id della tabella *omim_clinical_synopsys,* generato in maniera automatica durante il popolamento di tale tabella. Il codice recuperato per essere importato deve essere validato dall'espressione regolare: *[0-9]+;*
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo clinical_synopsys_id, in questo caso è *omim*;
6. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso la sorgente dati è sempre *omim*;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*;
8. association_type: campo codificato, descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"ANNOTATED_TO";*
9. taxonomy_id: è impostato con il valore *"9606",* perché i dati di OMIM sono riferiti solo all'organismo umano.

I sintomi sono messi in relazione con la patologia alla quale fanno riferimento e al sistema anatomico interessato, popolando la tabella seguente.

Campo estratto	Campo file	Campo database
		annotation_oid
Sintomo o segno	*FIELD*CS	phenotype
		reference_file

Tabella 44 – Mapping per i campi della tabella pheotype_4_gene2clinical_synopsys_imported

1. annotation_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *gene2clinical_synopsys_imported*;
2. phenotype: campo codificato, indica il sintomo riscontrato per la patologia associata;

3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

La tabella 45 mostra il mapping per i campi della tabella *genetic_disorder2clinical_synopsys_imported* del database.

Campo estratto	Campo file	Campo database
		annotation_oid
Mim number	*FIELD* NO	genetic_disorder_id
		origin_data_source
		clinical_synopsys_id
		destination_data_source
		reference
		reference_file
		association_type

Tabella 45 – Mapping per i campi della tabella genetic_disorder2clinical_synopsys_imported

1. annotation_oid: l'identificativo univoco assegnato dalla procedura d'importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. genetic_disorder_id: rappresenta il codice univoco assegnato al genetic_disorder da OMIM prelevato dal campo **FIELD*NO*, utilizzato per popolare la tabella principale *omim_disorder*. Il codice recuperato per essere importato deve essere validato dall'espressione regolare: *[0-9]{6};*
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID sorgente memorizzato nel campo genetic_disorder_disorder_id, vale a dire *omim*;
4. clinical_synopsys_id: contiene il valore del campo source_id della tabella *omim_clinical_synopsys*, generato in maniera automatica durante il popolamento di tale tabella. Il codice recuperato per essere importato deve essere validato dall'espressione regolare: *[0-9]+;*
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID sorgente memorizzato nel campo in clinical_synopsys_id, in questo caso è *omim*;
6. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso la sorgente dati è sempre OMIM;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*;

8. association_type: campo codificato che descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"ASSOCIATED_WITH"*.

In modo analogo a quanto avviene per i geni, i sintomi sono messi in relazione con la patologia alla quale fanno riferimento e il sistema anatomico interessato, popolando la seguente tabella.

Campo estratto	Campo file	Campo database
		annotation_oid
Sintomo o segno	*FIELD*CS	phenotype
		reference_file

Tabella 46 – Mapping per i campi della tabella phenotype_4_genetic_disorder2clinical_synopsys_imported

1. annotation_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. phenotype: campo codificato, indica il sintomo riscontrato per la patologia associata;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *omim.txt*.

- File genemap.key

La tabella 47 mostra il mapping tra i campi estratti dal file *genemap.key* e i campi della tabella di flag creata nel database.

Campo estratto	Campo file	Campo database
		id
Campo1	Code	name
Campo2	Description	description

Tabella 47 – Mapping per i campi di una generica tabella di flag

1. id: identificativo univoco generato automaticamente dall'applicazione al momento dell'inserimento di un nuovo record, utilizzato per popolare il campo codificato nelle tabelle di sorgente;
2. name: rappresenta il valore del campo codificato;
3. description: descrizione testuale del valore codificato.

- **File genemap e morbidmap**

Di seguito sono rappresentate le tabelle di abbinamento tra i campi estratti dal file *genemap* e i campi delle tabelle del database. Poiché le tipologie d'informazioni contenuta nel file *morbidmap* sono le stesse del file *genemap*, non si riporta il paragrafo contenente il mapping per i campi estratti da tale file poiché è possibile consultare le tabelle qui presenti.
Se il record letto rappresenta un gene, si popola la tabella *omim_gene* come mostrato nella seguente tabella.

Campo estratto	Campo file	Campo database
		omim_gene_oid
Campo10	Mim number	source_id
		source_name
		reference_file
		feature_type
		taxonomy_id
Campo8	Title	gene_title
Campo6	Symbol	symbol
Campo7	Status	status
Campo12	Comments	comments
Campo17	Mouse correlate	mouse_correlate
Campo4	Year entered	year
Campo2	Month entered	month
Campo3	Day entered	day
Campo5	Location	cytogenic_location
Campo1	Numbering system	chromosome
Campo1	Numbering system	map_entry_number

Tabella 48 – Mapping per i campi della tabella omim_gene

1. omim_gene_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: l'identificativo univoco assegnato da OMIM estratto dal *campo10* del file;
3. source_name: il nome codificato della sorgente dati che fornisce l'identificativo, vale a dire *omim;*
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *genemap*;

5. feature_type: campo codificato dai metadati, restituisce il tipo di dato che si sta importando, in questo caso geni;
6. taxonomy_id: non è recuperato direttamente dal file ma è impostato come costante e assume valore *"9606",* poiché i dati di OMIM sono riferiti solo all'organismo umano;
7. gene_title: titolo associato al gene;
8. symbol: il *campo6* può contenere un elenco di simboli, nel campo symbol della tabella s'inserisce il primo simbolo mentre gli altri popolano la tabella *omim_gene_alternative*;
9. status: stato del gene;
10. comments: commenti riferiti all'entry;
11. mouse correlate: riferimento ai cromosomi del topo correlati con il gene estratto;
12. year: anno in cui è stato catalogato il gene in *genemap*;
13. month: mese in cui è stato catalogato il gene in *genemap*;
14. day: giorno in cui è stato catalogato il gene in *genemap*;
15. cytogenic_location: posizione del gene estratto da *genemap*;
16. chromosome: cromosoma associato al gene estratto dal *campo1*;
17. map_entry_number: sottocampo estratto dal *campo1,* numero utilizzato per indicare l'ordine del gene in ogni cromosoma;

La tabella 49 è compilata in presenza di almeno due simboli all'interno del *campo6* estratto dal file.

Campo estratto	Campo file	Campo database
		omim_gene_oid
Symbol	Campo6	gene Symbol
		reference_file

Tabella 49 – Mapping per i campi della tabella omim_gene_alternative

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. symbol: il *campo6* può contenere un elenco di simboli, nel campo symbol della tabella si inseriscono, singolarmente, i simboli successivi al primo;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *genemap*.

La tabella 50 è compilata in presenza del *campo18* all'interno della riga estratta dal file.

Campo estratto	Campo file	Campo database
		omim_gene_oid
	Numero	ord
Campo18	Autore	author
Campo18	Anno	year
		progressive_year
		reference_file

Tabella 50 – Mapping per i campi della tabella omim_gene_reference_publication

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e anno di pubblicazione uguali;
3. progressive_year: è un numero incrementale gestito dal programma di importazione, utilizzato per differenziare record in cui per un autore sono presenti più reference pubblications in uno stesso anno;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso la sorgente dati è sempre *genemap*.

La tabella 51 è compilata in presenza di valori all'interno del *campo11* del file.

Campo estratto	Campo file	Campo database
		omim_gene_oid
Campo11	Method	method
		reference_file

Tabella 51 – Mapping per i campi della tabella omim_gene_method

1. omim_gene_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_gene*;
2. method: method code associato al gene memorizzato;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso la sorgente dati è sempre *genemap*.

La tabella 52 mostra il mapping per i campi della tabella *omim_disorder* del database.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
Campo10	Mim number	source_id
		source_name

		reference_file
		feature_type
Campo8	Title	disorder_title
Campo 14-15-16	Desease subtitle	disorder_subtitle
Campo8	Title	disorder_title
Campo6	Symbol	disorder_symbol
Campo7	Status	status
Campo12	Comments	comments
Campo17	Mouse correlate	mouse_correlate
Campo4	Year entered	year
Campo2	Month entered	month
Campo3	Day entered	day
Campo5	Location	cytogenetic_location
Campo1	Numbering system	chromosome
Campo1	Numbering system	map_entry_number
Campo14-15-16	Mutation	mutation_position
Campo14-15-16	Deseases type	desease_type
Campo14-15-16	Limbo	limbo_status

Tabella 52 – Mapping per i campi della tabella omim_disorder

1. omim_disorder_oid: l'identificativo univoco assegnato dalla procedura di importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. source_id: l'identificativo univoco assegnato da OMIM;
3. source_name: il nome codificato della sorgente dati che fornisce l'identificativo, vale a dire *omim;*
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *genemap*;
5. feature_type: campo codificato dai metadati, restituisce il tipo di dato che si sta importando, in questo caso genetic disorders;
6. disorder_title: titolo associato al genetic disorder che si sta importando;
7. disorder_subtitle: sottotitolo associato al genetic disorder che si sta importando;
8. disorder_symbol: il *campo6* può contenere un elenco di simboli, nel campo disorder_symbol della tabella si inserisce il primo simbolo mentre gli altri popolano la tabella *omim_disorder_alternative*;
9. status: stato del fenotipo;
10. comments: commenti riguardanti l'entry memorizzata;

11. mouse_correlate: riferimento ai cromosomi del topo correlati con il fenotipo estratto;
12. year: anno in cui è stata catalogata l'entry in *genemap*;
13. month: mese in cui è stata catalogata l'entry in *genemap*;
14. day: giorno in cui è stato catalogata l'entry in *genemap*;
15. cytogenic_location: posizione del fenotipo estratto da *genemap*;
16. chromosome: cromosoma associato al fenotipo;
17. map_entry_number: sottocampo estratto dal *campo1,* numero utilizzato per indicare l'ordine del fenotipo in ogni cromosoma;
18. mutation_position: identifica la tipologia di mutazione, viene elaborato è trasformato in flag;
19. desease_type: tipologia del disturbo, viene elaborato è trasformato in flag;
20. limbo_status: assume solo valori booleani. Il campo viene impostato con valore *True* se è presente il carattere di *"?"* prima del nome del disturbo, in caso contrario il campo viene settato a *False*.

La tabella 53 è popolata in presenza di almeno due simboli nel *campo6*.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
Symbol	Campo6	disorder_symbol
		reference_file

Tabella 53 – Mapping per i campi della tabella omim_disorder_alternative

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. symbol: il *campo6* può contenere un elenco di simboli, nel campo disorder_symbol della tabella si inseriscono, singolarmente, i simboli successivi al primo;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *genemap*.

La tabella 54 è popolata in presenza del *campo18* all'interno della riga estratta dal file.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
		ord
Campo18	Autore	author
Campo18	Anno	year

		progressive_year
		reference_file

Tabella 54 – Mapping per i campi della tabella omim_disorder_reference_publication

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. ord: è un contatore progressivo impiegato in fase di importazione per gestire la presenza di record con autore e anno uguali;
3. progressive_year: è un numero incrementale automatico gestito dal programma di importazione, utilizzato per differenziare record nei quali per un autore sono presenti più pubblicazioni in uno stesso anno;
4. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *genemap*.
5. La tabella 55 è compilata in presenza di valori all'interno del *campo11* del file.

Campo estratto	Campo file	Campo database
		omim_disorder_oid
Campo11	Method	method
		reference_file

Tabella 55 – Mapping per i campi della tabella omim_disorder_method

1. omim_disorder_oid: contiene l'identificativo assegnato dalla procedura alla tupla inserita nella tabella *omim_disorder*;
2. method: method code associato al genetic disorder memorizzato;
3. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *genemap*.

- File pubmed_cited

Di seguito sono rappresentate le tabelle di abbinamento tra i campi estratti dal file *pubmed_cited* e i campi delle tabelle del database.
Se il codice reperito dal *Campo1* identifica un gene, la tabella popolata è *gene2publication_reference_imported* mostrata in tabella 56.

Campo estratto	Campo file	Campo database
		annotation_oid
Campo1	Mim number	gene_id

		origin_data_source
Campo3	PubMedID	publication_reference_id
		destination_data_source
		reference
		reference_file
		association_type
		taxonomy_id

Tabella 56 – Mapping per i campi della tabella gene2publication_imported

1. annotation_oid: l'identificativo univoco assegnato dalla procedura d'importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. gene_id: rappresenta il codice univoco assegnato al gene da OMIM, per essere importato deve essere validato dall'espressione regolare: *[0-9]{6};*
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo gene_id, vale a dire *omim*;
4. pubblication_reference_id: rappresenta il codice univoco assegnato da PubMed alla pubblicazione, per essere importato deve essere validato dall'espressione regolare: *[0-9]+;*
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo publication_reference_id, in questo caso è *pubmed*;
6. reference: l'identificativo della fonte che ha fornito questa voce, in questo caso la sorgente dati è sempre *omim*;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *pubmed_cited*;
8. association_type: campo codificato, descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"RELATED_TO"*.
9. taxonomy_id: è impostato con il valore *"9606"*, perché i dati di OMIM sono riferiti solo all'organismo umano

Viceversa, se il codice reperito dal *Campo1* identifica un fenotipo, la tabella popolata è *genetic_disorder2publication_reference_imported* mostrata in tabella 57.

Campo estratto	Campo file	Campo database
		annotation_oid
Campo1	Mim number	genetic_disorder_id
		origin_data_source

Campo3	PubMedID	publication_reference_id
		destination_data_source
		reference
		reference_file
		association_type

Tabella 57 – Mapping per i campi della tabella genetic_disorder2publication_imported

1. annotation_oid: l'identificativo univoco assegnato dalla procedura d'importazione, consiste in un contatore incrementato ad ogni inserimento di un nuovo record nella tabella;
2. genetic_disorder_id: rappresenta il codice univoco assegnato al fenotipo da OMIM, per essere importato deve essere validato dall'espressione regolare: *[0-9]{6};*
3. origin_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo genetic_disorder_id, vale a dire *omim*;
4. pubblication_reference_di: rappresenta il codice univoco assegnato alla pubblicazione da PubMed, per essere importato deve essere validato tramite l'espressione regolare: *[0-9]+;*
5. destination_data_source: l'identificativo della sorgente dati che fornisce l'ID memorizzato nel campo publication_reference_id, in questo caso è *pubmed*;
6. reference: identificativo delle fonte che ha fornito questa voce, in questo caso la sorgente dati è *omim*;
7. reference_file: campo codificato nel quale viene memorizzato l'identificativo del file che ha fornito questa voce, in questo caso il file sorgente è sempre *pubmed_cited*;
8. association_type: campo codificato, descrive la tipologia della relazione che intercorre tra i due identificativi memorizzati, in questo caso *"ASSOCIATED_WITH"*.