SAPIENZA
UNIVERSITÀ DI ROMA

Realizzazione di una base di dati con MongoDB e confronto con il modello relazionale

Facoltà di ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Ingegneria informatica e automatica

Candidato

Alessio Devoto
Matricola 1701081

Relatore

Prof. Maurizio Lenzerini

Correlatore

Dott. Manuel Namici

Anno Accademico 2018/2019

Tesi non ancora discussa

Realizzazione di una base di dati con MongoDB e confronto con il modello relazionale

Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: devoto.1701081@studenti.uniroma1.it

Indice

Capitolo 1

Introduzione

L'archiviazione, l'elaborazione e la manutenzione di dati sono necessità primarie di aziende, organizzazioni e governi. La progettazione di basi di dati rigorose, il cui contenuto sia facilmente fruibile e accessibile è dunque, al giorno d'oggi, essenziale. Per *base di dati* intendiamo una collezione di dati, salvata in memoria secondaria, gestita da un apposito sistema software, detto *DataBase Management System* o DBMS. Il DBMS è dunque il "cuore" della base di dati, ed è responsabile della sua gestione.

Il primo sistema di gestione di basi di dati fu progettato agli inizi degli anni 60 da Charles Backman, alla General Electric. Da allora, molti progressi sono stati fatti in questo campo e molti altri sistemi sono stati progettati: con l'aumentare della quantità di dati disponibili, si sono rese necessarie tecnologie sempre più sofisticate dedicate alla loro gestione. Nel 1970 Edgar Codd propose il modello relazionale, che riscosse un grande successo e divenne rapidamente lo standard su cui furono progettati i DBMS moderni. Per molti anni il modello relazionale di Codd è rimasto il punto di riferimento ed è tuttora quello più utilizzato. Tuttavia, a partire dall'inizio del nuovo millennio, le quantità di dati da gestire sono diventate così grandi da rivelare alcune vulnerabilità e svantaggi nel modello relazionale e dando così adito a nuove filosofie di progettazione dei DBMS. I database basati su tali nuovi modelli, che differiscono sostanzialmente da quello relazionale, sono detti database non relazionali o NoSQL.

Questo progetto, condotto in collaborazione con il collega Giacomo Acitelli, si pone l'obbiettivo di costruire due basi di dati, una relazionale ed una non relazionale, a partire da una stessa fonte, e di confrontarne poi l'efficienza nella esecuzione di un insieme di interrogazioni. La fonte da cui si attinge è il database DBLP, di cui si parlerà diffusamente in seguito. Questo offre una collezione di informazioni riguardo a pubblicazioni di vario genere (libri, articoli, papers etc..) di interesse nel mondo della Computer Science. Per trattare il modello relazionale la scelta è ricaduta sul DBMS PostgreSQL, mentre per il modello non relazionale, su cui questa tesi si concentrerà maggiormente, si è scelto di utilizzare il sistema di gestione MongoDB.

Nel seguito si presenteranno le principali differenze tra i due modelli, tenendo conto dei rispettivi vantaggi e svantaggi sotto l'aspetto progettuale e implementativo. Si procederà poi con l'illustrazione della fase di progettazione, evidenziando le scelte compiute e motivandole. Si passerà poi alla realizzazione vera e propria

delle due basi di dati, concentrando l' attenzione sul paradigma non relazionale e sul DBMS MongoDB. Infine delle queries saranno effettuate su entrambi i database e se ne confronterà l'efficienza prestazionale.

Capitolo 2

Database relazionali e non relazionali

2.1 SQL e NoSQL

Il modello relazionale proposto da Edgar Codd nel 1970, che valse al suo inventore il premio Turing nel 1981, è rimasto per molti anni il punto di riferimento indiscusso per chiunque volesse progettare un Database management system. I DBMS basati su tale modello sono detti, appunto, *relazionali*, e spesso si fa riferimento a loro con l'acronimo RDBMS (Relational DataBase Management System). A partire dal 1974, nei laboratori della IBM, Donald Chamberlin elaborò un linguaggio che aveva le capacità di effettuare in modo efficiente manipolazioni di dati contenuti in un database relazionale. Questo linguaggio fu battezzato inizialmente con il nome "SEQUEL", ma fu poi rinominato SQL(STRUCTURED QUERY LANGUAGE), e divenne presto lo strumento standard per effettuare queries su questo tipo di database.

Dalla fine degli anni 90, sono sorte diverse problematiche che interessano i database relazionali. Tra queste, la difficoltà nella gestione di grandi quantità di dati, l'impossibilità di progettare database relazionali distribuiti - caratteristica, al giorno d'oggi, sempre più richiesta- e la conseguente non-scalabilità, hanno comportato la nascita di un movimento NoSQL(Not *Only* SQL), e di diversi DBMS progettati seguendo le linee guida di questa filosofia. L'autore della prima base relazionale NoSQL, Carlo Strozzi, dichiarò che "come movimento, NoSQL diparte in modo radicale dal modello relazionale, e quindi andrebbe chiamato in modo più appropriato NoREL, o qualcosa di simile".

2.2 Differenze nei due approcci

2.2.1 Modello relazionale: struttura e proprietà

Il modello relazionale è basato sul concetto di *relazione matematica*: siano $D_1, D_2, \dots D_n$ n insiemi non necessariamente distinti, una *relazione matematica* su $D_1, D_2 \dots D_n$ è un sottoinsieme del prodotto cartesiano $D_1 \times D_2 \cdots \times D_n$. Gli insiemi $D_1, D_2 \dots D_n$ sono detti **domini** della relazione. Una relazione su n domini ha grado n. Una

relazione è quindi un insieme di ennuple. Tra le varie ennuple che compongono la relazione, non esiste un ordinamento. Tuttavia ogni ennupla al suo interno è ordinata: il primo elemento viene dal primo dominio, il secondo elemento dal secondo dominio, e via discorrendo. Nel modello relazionale una relazione matematica è rappresentata con una **tabella**, in cui gli elementi sono detti **attributi**.

Le proprietà fondamentali che deve avere un DBMS per essere relazionale sono state elencate dallo stesso Codd in un articolo comparso sulla rivista ComputerWorld nel 1985. Queste sono:

- Regola 0: il sistema deve potersi definire come relazionale, base di dati e sistema di gestione. Affinché un sistema possa definirsi sistema relazionale per la gestione di basi di dati (RDBMS), tale sistema deve usare le proprie funzionalità relazionali (e solo quelle) per gestire la base di dati.

- Regola 1: l'informazione deve essere rappresentata sotto forma di tabelle. Le informazioni nel database devono essere rappresentate in maniera univoca, e precisamente attraverso valori in colonne che costituiscano, nel loro insieme, righe di tabelle.

- Regola 2: La regola dell' accesso garantito o delle chiavi primarie. Tutti i dati devono essere accessibili senza ambiguità. Ogni singolo valore scalare nel database dev'essere logicamente indirizzabile specificando il nome della tabella che lo contiene, il nome della colonna in cui si trova e il valore della chiave primaria della riga in cui si trova.

- Regola 3: trattamento sistematico del valore NULL. Il DBMS deve consentire all'utente di lasciare un campo vuoto, o con valore NULL. In particolare, deve gestire la rappresentazione di informazioni mancanti e quello di informazioni inadatte in maniera predeterminata, distinta da ogni valore consentito (per esempio, "diverso da zero o qualunque altro numero" per valori numerici), e indipendente dal tipo di dato. Queste rappresentazioni devono inoltre essere gestite dal DBMS sempre nello stesso modo.

- Regola 4: dizionario del modello relazionale. La descrizione della struttura del database e degli oggetti che lo compongono deve avvenire ad un livello logico, tramite un dizionario di metadati, e questo dizionario deve essere accessibile agli utenti del Database con le stesse modalità e lo stesso linguaggio di interrogazione utilizzato per accedere ai dati.

- Regola 5: accessibilità dei dati. Tutti i contenuti del database devono essere accessibili attraverso almeno un linguaggio relazionale (come ad esempio l'SQL) che abbia le seguenti caratteristiche:

  - abbia una sintassi lineare (ovvero le cui istruzioni possono essere semanticamente interpretate con una semplice lettura da sinistra verso destra)
  - possa essere utilizzato sia in forma interattiva che dall'interno di applicazioni

  - supporti operazioni di definizione e di manipolazione dei dati, le regole di sicurezza e i vincoli di integrità del database.

- Regola 6: aggiornamento delle viste di dati. In un database relazionale si possono creare viste che forniscono l'accesso a specifici subset di informazioni. Laddove il contenuto in termini di dati di queste viste è concettualmente modificabile, lo deve anche essere nella pratica.
- Regola 7: manipolazione dei dati ad alto livello. Da un database possiamo reperire informazioni multiple costituite da set di dati provenienti da più righe e/o più tabelle. Gli stessi set di dati, piuttosto che le singole informazioni, devono anche poter essere inseriti, aggiornati e cancellati.
- Regola 8: indipendenza dalla rappresentazione fisica. La struttura logica di un database deve essere indipendente dalle strutture di memorizzazione fisica: modifiche al piano fisico (come i dati vengono memorizzati, su quali unità, con quale organizzazione, ecc.) non devono richiedere un cambiamento alle modalità di accesso al database.
- Regola 9: indipendenza dalla rappresentazione logica. Le modifiche al livello logico (tabelle, colonne, righe, chiavi primarie, ...) non devono richiedere cambiamenti non giustificati alle applicazioni che utilizzano il database.
- Regola 10: i vincoli logici sui dati devono essere memorizzati nel Database. I vincoli di integrità propri delle entità e delle relazioni, le regole di sicurezza e le restrizioni di accesso devono essere definiti nel dizionario del database e sono quindi separati dalle applicazioni che lo utilizzano. Deve quindi essere possibile modificare tali vincoli senza inutilmente interessare le applicazioni esistenti.
- Regola 11: indipendenza di localizzazione. La distribuzione di porzioni del database su una o più allocazione fisiche o geografiche deve essere invisibile agli utenti del sistema. Le applicazioni esistenti devono continuare ad operare con successo quando i dati esistenti vengono ridistribuiti in modo diverso.
- Regola 12: regola di non sovversione: Gli strumenti di accesso ai dati non devono poter annullare le restrizioni del database, per esempio aggirando i vincoli di integrità, le relazioni o le regole di sicurezza.

Queste proprietà, conosciute anche come *"13 Regole di Codd"*, essendo molto restrittive, non sono rispettate da tutti i RDBMS moderni, che vengono ciononostante considerati *relazionali* a tutti gli effetti.

Proprietà ACID

Ulteriore tratto distintivo dei database relazionali è quello di garantire **l'integrità referenziale** della base di dati, ossia la consistenza delle relazioni in essa contenute. A tal fine, il database relazionale rispetta le proprietà cosiddette ACID:

- **Atomicità** Gli effetti della transazione devono essere del tutto visibili alla fine della stessa, oppure nessun effetto deve essere mostrato.

- **Consistenza** Alla fine della transazione i vincoli di integrità devono essere rispettati.
- **Isolamento** Transazioni concorrenti devono essere indipendenti, ossia il loro effetto deve essere lo stesso che si avrebbe nel caso in cui fossero eseguite separatamente.
- **Durabilità** L'effetto delle transazioni completate deve essere mantenuto nel database, anche in presenza di guasti.

2.2.2 Modello non relazionale

Un database non relazionale è un database che non usa lo schema tabulare di righe e colonne presente nella maggior parte dei sistemi di database tradizionali. I database non relazionali usano invece un modello di archiviazione ottimizzato per i requisiti specifici del tipo di dati da archiviare. I dati, ad esempio, possono essere archiviati come semplici coppie chiave/valore, come documenti JSON o sotto forma di un grafo composto da bordi e vertici. Il comune denominatore di tutti questi archivi dati è che non un usano un modello relazionale. Tendono invece a essere più specifici in merito al tipo di dati supportato e alle modalità di esecuzione delle query sui dati. Gli archivi dati di serie temporali, ad esempio, sono ottimizzati per query su sequenze di dati basate sul tempo, mentre gli archivi dati a grafo sono ottimizzati per l'esplorazione di relazioni ponderate tra entità. Nessuno dei due formati offrirebbe una generalizzazione corretta per attività di gestione di dati transazioni. Il termine NoSQL fa riferimento agli archivi dati che non usano SQL per le query ma usano altri costrutti e linguaggi di programmazione per eseguire query sui dati. In pratica, "NoSQL" significa "database non relazionale", anche se molti di questi database supportano query compatibili con SQL. La strategia per l'esecuzione di query sottostanti è in genere molto diversa dal modo in cui un tradizionale sistema RDBMS eseguirebbe la stessa query SQL. Nella sezione successiva si riporta un elenco dei principali tipi di database non relazionali.

2.3 Tipi di database non relazionali

- **Database a documento** Un archivio dati a documento gestisce un set di campi stringa denominati e di valori di dati oggetto in un'entità definita documento. Questi archivi memorizzano in genere i dati sotto forma di documenti JSON. Ogni valore di campo può essere un elemento scalare, ad esempio un numero o un elemento composto, come un elenco o una raccolta di tipo padre-figlio. I dati nei campi di un documento possono essere codificati in diversi modi, tra cui XML, YAML, JSON, BSON o anche archiviati come testo normale. Un archivio di documenti non richiede che tutti i documenti abbiano la stessa struttura. Questo approccio in formato libero offre una notevole flessibilità. Le applicazioni, ad esempio, possono archiviare dati diversi nei documenti in base ai cambiamenti dei requisiti aziendali.
- **Database a grafo** Un archivio dati a grafo archivia due tipi di informazioni: nodi e bordi. I nodi rappresentano le entità e i bordi specificano le relazioni

tra queste entità. I nodi e bordi possono avere proprietà che forniscono informazioni su tale nodo o bordo, analogamente alle colonne in una tabella. I bordi possono avere anche una direzione che indica la natura della relazione.

- **Database a colonne** Un archivio dati a colonne consente di organizzare i dati in righe e colonne. Nella sua forma più semplice, un archivio dati a colonne può risultare molto simile a un database relazionale, almeno a livello concettuale. È possibile considerare un archivio dati a colonne come contenente dati tabulari con righe e colonne, ma le colonne sono suddivise in gruppi, noti come famiglie di colonne. Ogni famiglia di colonne contiene un set di colonne logicamente correlate tra loro e, in genere, recuperate o modificate come un'unità. Altri dati di cui si accede separatamente possono essere archiviati in famiglie di colonne separate. All'interno di una famiglia di colonne, è possibile aggiungere nuove colonne in modo dinamico e le righe possono essere di tipo sparse, vale a dire una riga non deve necessariamente avere un valore per ogni colonna.
- **Database chiave:valore** Un archivio chiave/valore è essenzialmente una tabella hash di grandi dimensioni. Si associa ciascun valore dei dati con una chiave univoca e l'archivio chiave/valore usa questa chiave per archiviare i dati usando una funzione di hash appropriata. La funzione di hash è selezionata per fornire una distribuzione uniforme delle chiavi con hash nell'archiviazione dei dati.

Capitolo 3

Progettazione della base di dati

Nel seguito è presentata l'analisi delle fonti e la progettazione dei due database. Le informazioni raccolte durante ogni fase fungono da "input" per la fase successiva. È opportuno osservare che mentre nell'ambito della progettazione dei database relazionali esistono modelli e tecniche di comprovata efficienza, lo stesso non vale per i database non relazionali. Nel seguito, si farà uso del modello ENTITÀ-RELAZIONE (ER) per tutta la progettazione del database relazionale. Per il database NoSQL, si farà uso del modello ER solo per la progettazione concettuale.

3.1 Analisi dei documenti forniti da DBLP

DBLP è il prodotto di un progetto intrapreso all'università di Treviri nel 1993, che si prefigge di raccogliere materiale bibliografico riguardante pubblicazioni di diverso genere nel mondo della computer science. Si contano ad oggi circa 4 Milioni di pubblicazioni. La fonte primaria da cui si attingono i dati per i nostri database è il documento "DBLP.xml", scaricabile dal sito ufficiale di DBLP. A questo si accompagna un documento DTD (Document type definition) che definisce i vari record contenuti nel file xml e le loro dipendenze. Dettagli utili sui dati contenuti nei documenti sopra citati si trovano in [5].

3.1.1 Analisi di DBLP.xml

Il file "DBLP" utilizza 8 tipi di record, dedicati a rappresentare i possibili tipi di pubblicazioni. Ogni record può contenere attributi o entità:

Records

- **article** Articolo di giornale o rivista.
- **inproceedings** Articolo negli atti di una conferenza.
- **proceedings** Atti di una conferenza.
- **book** Libro o manuale.
- **incollection** Sezione di un libro avente titolo proprio.

- **phdthesis** Tesi di dottorato.
- **masterthesis** Tesi di laurea.
- **www** Sito internet.

Attributi

- **key** Chiave unica all'interno della bibliografia DBLP.
- **mdate** Data dell'ultima modifica.
- **reviewid** Chiave dell'ultima revisione. (Disponibile solo per i record *Article*)
- **rating** Classificazione dell'articolo. (Disponibile solo per i record *Article*)

Entità

- **author** Nome dell'autore.
- **editor** Nome dell'editore
- **title** Titolo.
- **booktitle** Titolo del libro di cui fanno parte elementi di tipo *Incollection* e *Inproceeding*.
- **pages** Numero di pagine, separato da virgole o da doppi trattini (–).
- **month** Il mese di pubblicazione (o, se inedito, il mese di creazione).
- **year** L'anno di pubblicazione (o, se inedito, l'anno di creazione).
- **address** Indirizzo dell'editore. Di solito solo la città, ma può anche essere l'indirizzo completo per gli editori meno noti.
- **journal** La rivista nel quale l'articolo è stato pubblicato.
- **volume** Il volume del giornale oppure del libro (se multivolume).
- **number** Atti di una conferenza.
- **number** Il numero del giornale, rivista o resoconto tecnico, se applicabile.
- **url** Per Inproceedings, Article e Incollection il campo url contiene il link alla tabella in cui il volume è contenuto.
- **ee** Collegamento verso un testo.
- **cdrom** Usata solo per ACM SIGMOD Anthology.
- **cite** Contiene le chiavi di altre pubblicazioni presenti nella bibliografia.
- **publisher** Il nome della casa editrice.

- **note** Offre la possibilità di inserire una nota.
- **crossref** Chiave dell'elemento gerarchicamente superiore.
- **isbn** Isbn di un libro, contenuto nell'elemento Book.
- **series** La collana di libri all'interno della quale è stato pubblicato.
- **school** L'istituto presso il quale è stata scritta la tesi.
- **chapter** Il numero del capitolo.

Si riporta di seguito, a titolo esemplificativo, un record *article* contenuto nel file:

```
<article mdate="2019-05-31" key="journals/access/ChenXJRL19">
<author orcid="0000-0001-5974-8935">Chen Chen</author>
<author>Junqi Xu</author>
<author>Wen Ji</author>
<author>Lijun Rong</author>
<author>Guobin Lin</author>
<title>Sliding Mode Robust Adaptive Control of Maglev Vehicle's
Nonlinear Suspension System Based on Flexible
Track: Design and Experiment.</title>
<pages>41874-41884</pages>
<year>2019</year>
<volume>7</volume>
<journal>IEEE Access</journal>
<ee type="oa">https://doi.org/10.1109/ACCESS.2019.2906245</ee>
<url>db/journals/access/access7.html#ChenXJRL19</url>
</article>
```

3.1.2 Analisi di DBLP.dtd

Il documento fornisce indicazioni sulla struttura interna della bibliografia.

Elenco dei tipi di record (entità) principali:

```
<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
                phdthesis|mastersthesis|www)*>
```

Elenco delle altre entità:

```
<!ENTITY  \% field "author|editor|title|booktitle|pages|year|address
|journal|volume|number|month|url|ee|cdrom|cite|publisher|note|crossref
|isbn|series|school|chapter">
```

Segue la descrizione di tutte le entità, con entità e attributi ad esse associati. Riportiamo, a titolo di esempio, il record di tipo *Article*, e omettiamo per brevità i rimanenti, che possono essere comunque consultati in appendice A.

```
<!ELEMENT article  (\%field;)*>
<!ATTLIST article>
```

```
key CDATA #REQUIRED
reviewid CDATA #IMPLIED
rating CDATA #IMPLIED
mdate CDATA #IMPLIED
```

3.2 Progettazione concettuale

In questa fase si considerano i requisiti e le informazioni raccolte nella fase precedente e si procede alla realizzazione dello schema concettuale della base di dati, secondo il modello E-R. Lo schema concettuale, che rappresenta "l'output" di questa prima fase, esprime la struttura *intensionale* del database. Tale schema deve quindi esplicitare, sebbene ad un alto livello di astrazione logica, quali sono **i tipi di oggetto** che dovrà trattare il database, **le relazioni** che li legano, e **i vincoli** che devono essere rispettati per garantire l'integrità dei dati.

I costrutti fondamentali che è necessario prendere in considerazione in questa fase sono:

- Entità
- Relazioni
- Attributi (di entità e di relazioni)
- Ruoli (che un entità svolge in una relazione)
- IS-A e generalizzazioni
- Vincoli

3.2.1 Entità

Una entità è una classe di oggetti che sono di interesse per l'applicazione, che hanno esistenza autonoma, e che hanno proprietà comuni. Ogni entità può possedere o meno un insieme di attributi. Su ogni entità sono inoltre definiti uno o più identificatori. Un identificatore di una entità è un insieme di proprietà (attributi o relazioni) che permettono di identificare univocamente le istanze di tale entità. In altre parole non esistono due istanze dell'entità che hanno lo stesso valore per tutte le proprietà che formano l'identificatore. Nell' elenco seguente sono sottolineati i vincoli di identificazione principali di ogni entità.

- **Author**
  - name : Elemento author del record XML.
- **Editor**
  - name: Elemento editor del record XML.
- **Publisher**
  - name: Elemento publisher del record XML.

  - href: Attributo href del record XML.
  - address: Elemento address del record XML.

- **Article**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - reviewid : Attributo reviewid del record XML.
  - rating : Attributo rating del record XML.
  - title : Elemento title del record XML.
  - url : Elemento url del record XML.
  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - crossref : Elemento crossref del record XML.
  - ee : Elemento ee del record XML.
  - note : Elemento note del record XML.

- **Journal**
  - name : Elemento journal del record XML.
  - volume : Elemento volume del record XML.
  - number : Elemento number del record XML.

- **Proceedings**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - title : Elemento title del record XML.
  - url : Elemento url del record XML.
  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - isbn : Elemento isbn del record XML.
  - ee : Elemento ee del record XML.
  - note : Elemento note del record XML.

- **Inproceedings**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - title : Elemento title del record XML.
  - url : Elemento url del record XML.
  - crossref : Elemento crossref del record XML.

  - booktitle : Elemento booktitle del record XML.
  - pages : Elemento pages del record XML.
  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - cdrom : Elemento cdrom del record XML.
  - ee : Elemento ee del record XML.
  - note : Elemento note del record XML.

- **Book**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - title : Elemento title del record XML.
  - isbn : Elemento isbn del record XML.
  - cdrom : Elemento cdrom del record XML.
  - url : Elemento url del record XML.
  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - note : Elemento note del record XML.

- **Incollection**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - title : Elemento title del record XML.
  - crossref : Elemento crossref del record XML.
  - booktitle : Elemento booktitle del record XML.
  - chapter : Elemento chapter del record XML.
  - pages : Elemento pages del record XML.
  - url : Elemento url del record XML.
  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - ee : Elemento ee del record XML.
  - note : Elemento note del record XML.

- **Thesis**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - title : Elemento title del record XML.
  - school : Elemento school del record XML.

  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - note : Elemento note del record XML.

- **MasterThesis** Questo elemento è stato generalizzato in Thesis.
- **PhdThesis** Questo elemento è stato generalizzato in Thesis.
- **WWW**
  - id : Attributo key del record XML.
  - mdate : Attributo mdate del record XML.
  - title : Elemento title del record XML.
  - url : Elemento url del record XML.
  - month : Elemento month del record XML.
  - year : Elemento year del record XML.
  - note : Elemento note del record XML.
- **Series**
  - name : Elemento series del record XML.
- **Cites**
  - label : Attributo label del record XML.
  - text : Elemento text del record XML.

3.2.2 Relazioni

Una relazione è un' associazione tra due o più entità. Nello schema E-R ad esempio, constatiamo che un articolo è contenuto in una rivista mediante la relazione *In*.

Una relazione, inoltre, può anche avere attributi descrittivi. Essi vengono utilizzati per registrare informazioni sulla relazione piuttosto che sulle entità partecipanti.Ad esempio, la relazione *In* ha l'attributo *Pages*, quest'ultimo indica le pagine della rivista in cui è contenuto quel determinato articolo.

Sulla base delle informazioni raccolte nella fase di analisi sono state individuate le seguenti relazioni:

In - Journal (0,n) ↔ (1,1) Articles : asserisce che ogni articolo è contenuto in una specifica rivista in determinate pagine. Ogni rivista, in determinate pagine di una specifica rivista contiene quindi un solo articolo.

- *pages* : Elemento *pages* del record XML.

In_a - Proceedings (0,1) ↔ (0,1) Series : asserisce che una conferenza può essere contenuta in uno specifico volume di una determinata serie. Ciascun volume di una serie contiene una sola conferenza.

- *volume* : Elemento *volume* del record XML.

Cited_1 - Cites (1,1) ↔ (0,n) Books : asserisce che un libro può avere un numero indeterminato di citazioni, mentre ogni citazione è riferita ad uno specifico libro.

Cited_2 - Cites (1,1) ↔ (0,n) Proceedings : asserisce che una conferenza può avere un numero indeterminato di citazioni, mentre ogni citazione è riferita ad una specifica conferenza.

Published_1 - Proceedings (0,1) ↔ (0,n) Publishers : asserisce che gli atti di una conferenza possono essere pubblicati da una sola casa editrice. D'altro canto, una casa editrice può pubblicare un numero indeterminato di atti di conferenze.

Published_2 - Books (0,1) ↔ (0,n) Publishers : asserisce che un libro può essere pubblicato da una sola casa editrice,ma, una casa editrice può pubblicare un numero indeterminato di libri.

Edited_1 - Proceedings (0,n) ↔ (0,n) Editors : Asserisce che una conferenza può essere edita da un numero indeterminato di editori, e che un'editore può pubblicare un numero indeterminato di conferenze.

Edited_2 - Incollections (0,n) ↔ (0,n) Editors : Asserisce che la parte di un libro può essere edita da un numero indeterminato di editori, e che un editore può pubblicare un numero indeterminato di parti di libri.

Edited_3 - Books (0,n) ↔ (0,n) Editors : Asserisce che un libro può essere edito da un numero indeterminato di editori, e che un editore può pubblicare un numero indeterminato di libri.

By_1 - Articles (1,n) ↔ (0,n) Authors : Asserisce che ogni articolo deve avere almeno un autore. Ogni autore può partecipare ad un numero indeterminato di articoli.

By_2 - WWW (1,n) ↔ (0,n) Authors : Asserisce che ogni sito web appartiene almeno ad un autore. Ogni autore può possedere un numero indeterminato di siti web.

By_3 - Books (1,n) ↔ (0,n) Authors : Asserisce che ogni libro deve avere almeno un autore. Ogni autore può partecipare ad un numero indeterminato di libri.

By_4 - Incollections (1,n) ↔ (0,n) Authors : Asserisce che un'entità Incollection deve avere almeno un autore. Ogni autore può partecipare ad un numero indeterminato di entità Incollection.

By_5 - Thesis (1,n) ↔ (0,n) Authors : Asserisce che ogni tesi deve avere almeno un autore. Ogni autore può partecipare ad un numero indeterminato di tesi.

By_6 - Inproceedings (1,n) ↔ (0,n) Authors : Asserisce che ogni articolo discusso in una conferenza deve avere almeno un autore. Ogni autore può partecipare ad un numero indeterminato di articoli discussi in una conferenza.

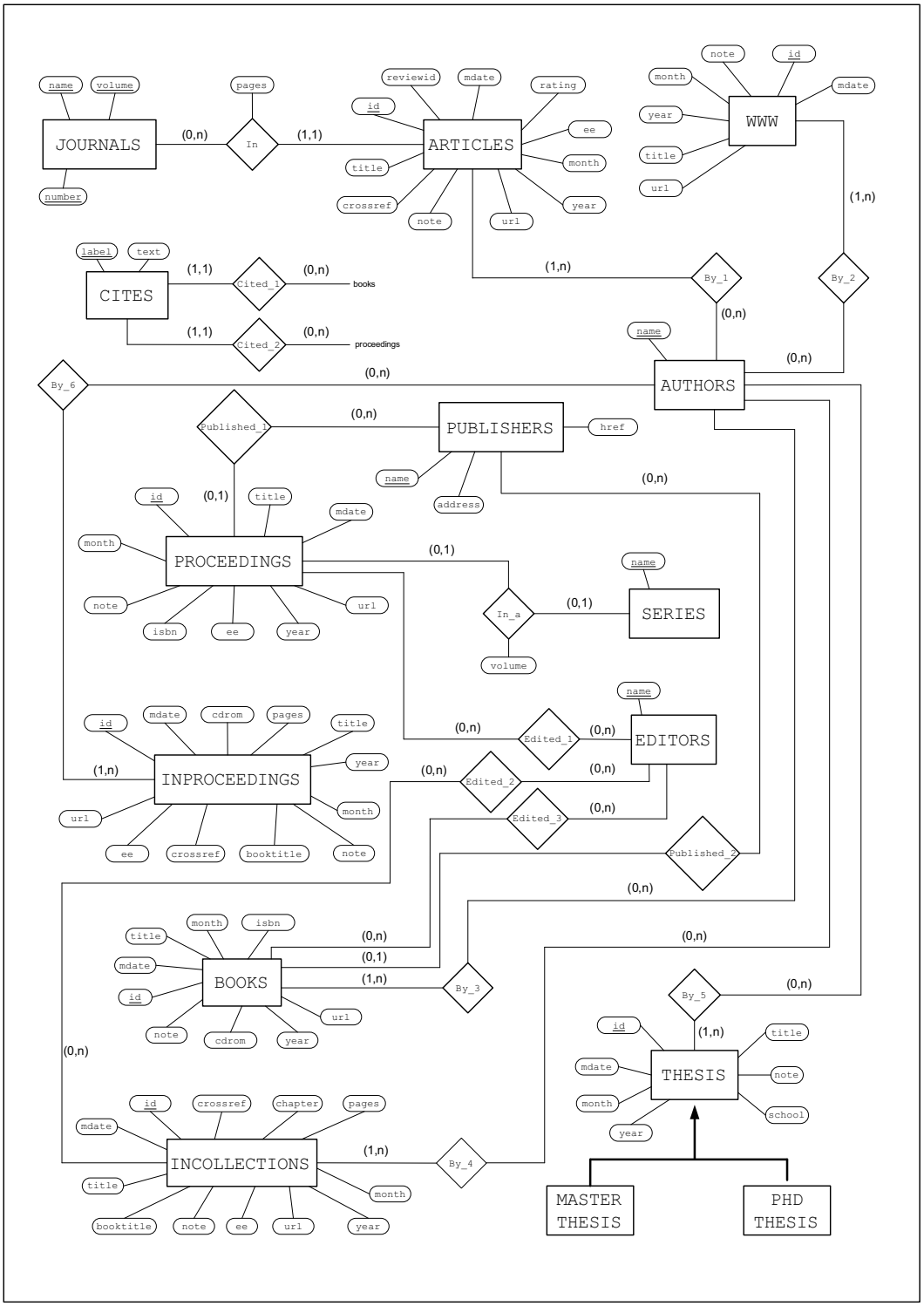

Figura 3.1. Schema concettuale

3.3 Progettazione logica della base di dati

In questa fase viene scelto il DBMS per implementare il progetto, e convertire la progettazione concettuale in uno schema del modello del DBMS scelto. Il risultato è uno **schema logico** nel modello di dati relazionale.

Ogni insieme di entità viene tradotta in una relazione (o tabella) nel seguente modo: ogni attributo di un insieme di entità diventa un attributo della tabella. Un insieme di relazioni, così come le entità, viene tradotto in una relazione del modello relazionale. Gli attributi della relazione includono:

- Gli attributi della chiave primaria di ciascun insieme di entità partecipante, come campi di chiave esterne.
- Gli attributi descrittivi dell'insieme di relazioni.

3.3.1 Le relazioni

Di seguito verranno elencate le relazioni della base di dati relazionale.

Authors(<u>id</u>, <u>name</u>) Poichè l'entità *Authors* è composta da un solo attributo, è stato scelto di integrare le relazioni *By_1, By_2, By_3, By_4, By_5, By_6* direttamente nella relazione *Authors*. Il vincolo di integrità imposto dalle cardinalità è rispettato per il fatto che sia **id** che **name** sono chiave per la relazione. In questo modo possono esistere 2 tuple con lo stesso autore, ma che differiscono per il campo id (un autore è responsabile di più pubblicazioni). Inoltre, una pubblicazione può avere più autori (stesso valore **id** ma diverso valore **name**). Non è invece possibile avere tuple il cui campo **id** è diverso da *null* ma il campo **name** è *null* (significherebbe che una pubblicazione non ha autori, violando quindi la cardinalità).

Editors(<u>id</u>, <u>name</u>) Anche per la relazione Editors è stato scelto di integrare le relazioni *Edited_1*, *Edited_2* ed *Edited_3* direttamente nella relazione stessa. Le considerazioni sul rispetto dei vincoli di integrità sono identiche al caso della relazione Authors, e non verranno discusse.

Articles(<u>id</u>, mdate, reviewid, rating, title, url, month, year, crossref, ee, note) Tabella per la memorizzazione di articoli.

Proceedings(<u>id</u>, mdate, title, url, month, year, isbn, ee, note) Tabella per la memorizzazione di elementi Proceeding.

Inproceedings(<u>id</u>, mdate, title, url, crossref, booktitle, pages, month, year, cdrom, ee, note)

Incollections(<u>id</u>, mdate, title, crossref, booktitle, chapter, pages, url, month, year, ee, note) Tabella per la memorizzazione di elementi Incollection.

Master_Thesis(<u>id</u>, mdate, title, school, month, year, note) Tabella per la memorizzazione di tesi di laurea.

Phd_Thesis(id, mdate, title, school, month, year, note) Tabella per la memorizzazione di tesi di dottorato.

Books(id, mdate, title, isbn, cdrom, url, month, year, note) Tabella per la memorizzazione di libri.

Publishers(name, address, href) Tabella per la memorizzazione delle case editrici.

WWW(id, mdate, title, url, month, year, note) Tabella per la memorizzazione di elementi www.

Cites(id, label, text) È stato scelto di integrare le relazioni *Cited_1* e *Cited_2* dello schema E-R direttamente nella tabella Cites. È stato aggiunto un'ulteriore campo, che corrisponde all'attributo **id** delle relazioni *Books* e *Proceedings*. Le cardinalità sono rispettate dalla condizione che sia **id** che **label** sono chiave per la relazione.

Published_1(id, name) Tabella per la memorizzazione della relazione *Published_1*. Sia **id** che **name** sono chiave per la relazione. Il campo **id** referenzia Proceedings mentre il campo **name** referenzia publisher. In questo modo viene garantito il vincolo di integrità imposto dalle cardinalità: gli elementi di una conferenza possono essere pubblicati da una sola casa editrice.

Published_2(id, name) Tabella per la memorizzazione della relazione *Published_2*. Stesse considerazioni della tabella Published_1.

In_a(id, name, volume) Poichè l'entità *Series* ha un solo attributo, è stato deciso di integrare l'entità in questione con la relazione *In_a*. Le chiavi della relazione sono *id* (contiene la chiave di un'entry *Proceedings*) e *name* (nome della serie). Le cardinalità sono rispettate poichè gli atti di una conferenza sono contenuti in un volume di una sola serie (non possono esistere due entry con stessi valori di id e name).

In_1(id, name, volume, number, pages) Anche in questo caso è stato scelto di integrare l'entità *Journal* con la relazione *In_1*. Tale scelta è motivata dal fatto che la relazione tra Journals e Articles è 1 ad 1, ciò significa che nella relazione In_1 non esisteranno 2 tuple con stessi valori. Di conseguenza, il vantaggio di modellare una relazione dello schema E-R come tabella veniva a cadere. Modellare **id**, **name**, **volume** e **number** come chiavi, implica che, in determinate pagine di un volume di una rivista è contenuto un solo articolo (rispettando quindi le cardinalità).

3.4 Osservazioni

È opportuno osservare che mentre la fase di progettazione concettuale è comune ad entrambi i paradigmi relazionale e non relazionale , quella di progettazione logica è legata al tipo di DBMS scelto. Nella parte precedente è stata riportata la

progettazione logica relativa ad un database relazionale. Nel caso di database no-SQL la realizzazione risulta meno complessa, in quanto si rende necessario solo il caricamento dei documenti nel DBMS, senza la costrizione dei vincoli.

Si è scelto, per la base di dati gestita da MongoDB, di *creare una collezione per ogni tipo di record principale, ossia per ogni tipo di pubblicazione*. Questa scelta favorisce l'aggregazione dei dati, e sfrutta appieno la flessibilià di un database non relazionale orientato ai documenti come MongoDB, consentendo di inserire i documenti senza effettuare controlli di integrità su vincoli e relazioni.

Capitolo 4

Implementazione delle basi di dati

L'inserimento dei dati nei due database è stato effettuato con metodologie molto diverse. Se da un lato si è dovuto usare grande rigore all' atto della creazione delle tabelle in PostgreSQL, le quali devono essere coerenti con la progettazione concettuale e logica, dall'altro si è profittato della grande flessibilità offerta dai database non relazionali orientati ai documenti, come MongoDB. Nel secondo caso infatti è stato necessario compilare dei files con i dati esportati da dblp.xml e importarli nel database, suddividendoli in collezioni.

4.1 Popolamento della base di dati non relazionale

Come detto in precedenza, l'importazione dei dati in MongoDB risulta molto più rapida e meno complessa rispetto a quella in PostgreSQL: MongoDB è in grado di importare direttamente documenti in formato JSON o CSV. Questi documenti vengono rappresentati all'interno del sistema in formato BSON (binary JSON), che facilita la gestione dei dati da parte del DBMS. Si è scelto, coerentemente con la progettazione logica, di creare una collezione per ogni tipo di record. È stato quindi necessario compilare, a partire dal file dblp.xml, 8 documenti in formato JSON, ognuno dei quali contenesse l'elenco di tutti gli elementi del rispettivo record. Ad esempio, nel file "articles.json" sono contenuti tutti i record article presenti in DBLP, trasformati secondo le regole di sintassi di JSON.

Creazione dei files in formato JSON

Per la compilazione dei documenti a partire dal file dblp.xml, si è deciso di scrivere una applicazione (chiamata *MongoConverter*) in linguaggio Java e si è dovuto scegliere tra due diversi meccanismi di *parsing*: DOM (Document object model) e SAX(Simple API for XML). Il primo meccanismo permette di convertire un file XML leggendolo e creando in memoria un albero, i cui nodi contengono ciascuno un elemento del file fornito in input. Il secondo meccanismo consiste invece nella scansione lineare del file XML, e nella sua elaborazione attraverso funzioni di *callback*. La scelta è ricaduta inevitabilmente sulla API di SAX, in quanto il primo metodo

è, a livello computazionale, molto oneroso e incapace di gestire file di dimensioni nell'ordine dei GB, come nel nostro caso. La creazione di un albero a partire dal file fornito sarebbe stata infatti un'operazione eccessivamente dispendiosa in termini di risorse di spazio e di tempo.

L' applicazione *MongoConverter* consiste di due classi: MongoConverter.java e HandlerDBLP.java. La prima si occupa della gestione dei file e del loro formato ed encoding, la seconda invece effettua la vera e propria lettura del documento: scandisce il file fornito in input, legge il contenuto elemento dopo elemento e crea una stringa in formato JSON, stampandola in output sul file opportuno. L'intero codice dell'applicazione MongoConverter è consultabile nell' appendice B.

Di seguito si riporta, a titolo di esempio, un record **article** del file *dblp.xml* e la sua traduzione in un oggetto JSON nel file *article.json*.

dblp.xml

```
<article mdate="2017-05-28" key="journals/acta/Saxena96">
<author>Sanjeev Saxena</author>
<title>Parallel Integer Sorting and Simulation Amongst CRCW Models.</title>
<pages>607-619</pages>
<year>1996</year>
<volume>33</volume>
<journal>Acta Inf.</journal>
<number>7</number>
<url>db/journals/acta/acta33.html#Saxena96</url>
<ee>https://doi.org/10.1007/BF03036466</ee>
</article>
```

article.json

```
{
  "volume": "33",
  "ee": "https:\/\/doi.org\/10.1007\/BF03036466",
  "number": "7",
  "pages": "607-619",
  "journal": "Acta Inf.",
  "mdate": "2017-05-28",
  "year": "1996",
  "author": "Sanjeev Saxena",
  "title": "Parallel Integer Sorting and Simulation Amongst CRCW Models.",
  "key": "journals\/acta\/Saxena96",
  "url": "db\/journals\/acta\/acta33.html#Saxena96"
}
```

Importazione dei documenti in MongoDB

Una volta creato il database "database0" ed i documenti per le collezioni, questi sono stati importati in MongoDB attraverso istruzioni fornite da linea di comando. Per ogni tipo di collezione, è stato necessario fornire il comando:

```
mongoimport --db  database0 --collection collezione --file collezione.json
```

in cui si è sostituito di volta in volta `collezione` con un tipo di record, ad esempio "Article". Il risultato dell'inserimento dei dati è presentato nella tabella seguente, in cui sono riportati, accanto ad ogni collezione creata, il numero di documenti, la dimensione della collezione e la dimensione media di ogni documento della collezione stessa.

Collezione	Documenti	Dimensione media documento	Dimensione totale collezione
articles	2.078.688	450 B	935.6 MB
books	17.597	367.2 B	6.5 MB
incollections	58.110	420.5 B	24.4 MB
inproceedings	2.432.533	444.8 B	1.1 GB
mastersthesis	12	329.8 B	4.0 KB
phdthesis	73.062	415.9 B	30.4 MB
proceedings	10.079	507.8 B	5.1 MB
www	2.329.917	127.1 B	296.0 MB

Capitolo 5

Confronto sulle prestazioni

Il database è stato interrogato direttamente da terminale, utilizzando i comandi secondo la sintassi di MongoDB. Per misurare le prestazioni è stato sufficiente aggiungere l'operatore *explain()* alla query, con argomento la stringa "executionStats".

5.1 Queries

Di seguito si presentano le query effettuate su entrambi i database, con i relativi codici ed i tempi di esecuzione.

5.1.1 Interrogazione 1

Restituisce il titolo e l'autore di ogni articolo. In SQL è stato necessario utilizzare l'operatore *join*, in MongoDB è stata sufficiente la funzione *find()*.

Query per PostgreSQL:

```
SELECT articles.title, authors.name
FROM articles JOIN authors on articles.id = authors.id
```

Query per MongoDB:

```
db.articles.find({}, {"title":1, "author":1})
```

Tempi di esecuzione	
PostgreSQL	MongoDB
2425 ms	1620 ms

5.1.2 Interrogazione 2

Restituisce il numero di libri pubblicati per ogni anno. In SQL è stato necessario utilizzare l'operatore di *groupby* e di *count*; in MongoDB la funzione *aggregate()*.

Query per PostgreSQL:

```
SELECT books.year, count(books.id)
FROM books
GROUP BY books.year
```

Query per MongoDB:

```
db.books.aggregate([{$group : {_id : "$year", tot : {$sum : 1}}}])
```

Tempi di esecuzione	
PostgreSQL	MongoDB
156 ms	825 ms

5.1.3 Interrogazione 3

Restituisce il titolo delle tesi di dottorato della "Harvard University". Query per PostgreSQL:

```
SELECT phdthesis.title
FROM phdthesis
WHERE phdthesis.school = 'Harvard University'
```

Query per MongoDB:

```
db.phdthesis.find(
 {"school":"Harvard University"}, {"title":1})
```

Tempi di esecuzione	
PostgreSQL	MongoDB
70 ms	40 ms

5.2 Conclusioni

Nella prima query MongoDB risulta avere, in media, prestazioni nettamente superiori a quelle di PostgreSQL. Questa maggiore efficienza del database non relazionale è dovuta all'aggregazione dei dati fatta in fase di progettazione: mentre nel primo caso è stato necessario effettuare una join tra due relazioni, operazione molto onerosa dal punto di vista computazionale, nel secondo è stato sufficiente utilizzare la funzione *find()*. Infatti in MongoDB ogni documento di tipo "article" contiene un array degli autori dell'articolo stesso. In questa circostanza è importante fare un'osservazione sull'operatore di join, presente in SQL e non in MongoDB. Sebbene l'operazione di join sia molto costosa in termini di risorse di tempo, e riduca quindi in alcuni casi le prestazioni di SQL rispetto a MongoDB, esistono tuttavia innumerevoli casi in cui effettuare una join è inevitabile, anche per un database non relazionale. In questi casi, l'unica opzione possibile è quella di eseguire il join direttamente dall' *application layer*, impiegando infine le stesse risorse che si sarebbero impiegate con SQL.

Nella seconda query, al contrario, PostgresSQL ha prestazioni mediamente superiori. La differenza in questo caso è dovuta all'invocazione della funzione *aggregate()* in MongoDB, con la quale si richiede al DBMS di contare, attraverso un meccanismo a pipeline, tutte le istanze di una data collezione, dopo averle raggruppate secondo il loro anno di pubblicazione. Tale funzione realizza innumerevoli operazioni ed è quindi responsabile della differenza di prestazioni in questa query.

Nella terza query si è scelto di effettuare una semplice interrogazione al database, che richiede di selezionare un sottoinsieme dell'elenco delle tesi di dottorato - quelle registrate alla Harvard University - e proiettarne il titolo in output. In questo caso l'efficienza dei DBMS si è dimostrata essere all'incirca uguale.

Appendice A

Elenco delle entità nel documento dblp.dtd

```
<!ELEMENT article       (%field;)*>
<!ATTLIST article
                        key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        reviewid CDATA #IMPLIED
                        rating CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT inproceedings (%field;)*>
<!ATTLIST inproceedings key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT proceedings   (%field;)*>
<!ATTLIST proceedings   key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT book          (%field;)*>
<!ATTLIST book          key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>
```

```
<!ELEMENT incollection  (%field;)*>
<!ATTLIST incollection  key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT phdthesis     (%field;)*>
<!ATTLIST phdthesis     key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT mastersthesis (%field;)*>
<!ATTLIST mastersthesis key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT www           (%field;)*>
<!ATTLIST www           key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT data          (%field;)*>
<!ATTLIST data          key CDATA #REQUIRED
                        mdate CDATA #IMPLIED
                        publtype CDATA #IMPLIED
                        cdate CDATA #IMPLIED
>

<!ELEMENT person ((author*, (note|url|cite)*)|crossref) >
<!ATTLIST person key CDATA #REQUIRED
          mdate CDATA #IMPLIED
          cdate CDATA #IMPLIED
 >

<!ELEMENT author    (#PCDATA)>
<!ATTLIST author
                    aux CDATA #IMPLIED
                    bibtex CDATA #IMPLIED
                    orcid CDATA #IMPLIED
                    label CDATA #IMPLIED
```

```
                    type CDATA #IMPLIED
>

<!ELEMENT editor    (#PCDATA)>
<!ATTLIST editor
                    aux CDATA #IMPLIED
                    orcid CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT address   (#PCDATA)>
<!ATTLIST address
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ENTITY % titlecontents "#PCDATA|sub|sup|i|tt|ref">
<!ELEMENT title     (%titlecontents;)*>
<!ATTLIST title
                    bibtex CDATA #IMPLIED
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT booktitle (#PCDATA)>
<!ATTLIST booktitle
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT pages     (#PCDATA)>
<!ATTLIST pages
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT year      (#PCDATA)>
<!ATTLIST year
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>
```

```
<!ELEMENT journal   (#PCDATA)>
<!ATTLIST journal
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT volume    (#PCDATA)>
<!ATTLIST volume
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT number    (#PCDATA)>
<!ATTLIST number
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT month     (#PCDATA)>
<!ATTLIST month
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT url       (#PCDATA)>
<!ATTLIST url
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT ee        (#PCDATA)>
<!ATTLIST ee
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT cite      (#PCDATA)>
<!ATTLIST cite
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
```

```
                    type CDATA #IMPLIED
                    ref CDATA #IMPLIED
>

<!ELEMENT school    (#PCDATA)>
<!ATTLIST school
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT publisher (#PCDATA)>
<!ATTLIST publisher
                    href CDATA #IMPLIED
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT note      (#PCDATA)>
<!ATTLIST note
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT cdrom     (#PCDATA)>

<!ELEMENT crossref  (#PCDATA)>
<!ELEMENT isbn      (#PCDATA)>
<!ATTLIST isbn
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT chapter   (#PCDATA)>
<!ELEMENT series    (#PCDATA)>
<!ATTLIST series
                    href CDATA #IMPLIED
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!ELEMENT publnr (#PCDATA) >
<!ATTLIST publnr
```

```
                    aux CDATA #IMPLIED
                    label CDATA #IMPLIED
                    type CDATA #IMPLIED
>

<!--  sub elements of the title element -->
<!ELEMENT ref (#PCDATA)>
<!ATTLIST ref href CDATA #REQUIRED>
<!ELEMENT sup (%titlecontents;)*>
<!ELEMENT sub (%titlecontents;)*>
<!ELEMENT i   (%titlecontents;)*>
<!ELEMENT tt  (%titlecontents;)*>
```

Appendice B

Codice Java dell' applicazione MongoConverter

Di seguito sono riportate le due classi Java dell'applicazione *MongoConverter*.

MongoConverter.java

```
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import javax.xml.parsers.*;
import java.io.*;
import static com.sun.org.apache.xerces.internal.impl.
   Constants.JDK_ENTITY_EXPANSION_LIMIT;


public class MongoConverter {
    public static int MAIN_NODES = 8;

    private static String xml_file_path = "/Users/
       alessiodevoto/tesi/dblp.xml";
    private static String json_file_path = "/Users/
       alessiodevoto/tesi/json_files/";

    private static String[] main_nodes = {"article", "
       proceedings", "inproceedings", "book", "
       incollection", "phdthesis", "mastersthesis", "www"
       };
    private static File[] output_files = new File[8];
    private static PrintWriter[] printwriters = new
       PrintWriter[8];

    public static void main(String[] args) throws
       ParserConfigurationException, SAXException {

```

```
        File xml_file = new File(xml_file_path);
23
        InputStream inputStream= null;
25      try {
            inputStream = new FileInputStream(xml_file);
27      } catch (FileNotFoundException e) {
            e.printStackTrace();
29      }

31      Reader reader = null;
        try {
33          reader = new InputStreamReader(inputStream,"
               ISO-8859-1");
        } catch (UnsupportedEncodingException e) {
35          e.printStackTrace();
        }
37      InputSource is = new InputSource(reader);
        is.setEncoding("ISO-8859-1");
39      System.out.println("[MongoConverter]Input file
           open");

41      for(int i=0; i<MAIN_NODES;i++){
            output_files[i] = new File(json_file_path+
               main_nodes[i]+".json");
43          try {
                printwriters[i] = new PrintWriter(new
                   OutputStreamWriter(new
                   FileOutputStream(output_files[i])));
45          } catch (FileNotFoundException e) {
                e.printStackTrace();
47          }
        }
49      System.out.println("[MongoConverter]Output files
           open");

51      SAXParserFactory factory = SAXParserFactory.
           newInstance();
        SAXParser parser = factory.newSAXParser();
53      parser.setProperty(JDK_ENTITY_EXPANSION_LIMIT, 0)
           ;
        System.out.println("[MongoConverter]Sax parser
           ready");
55
        HandlerDBLP handler = new HandlerDBLP(
           printwriters);
57      System.out.println("[MongoConverter]File handler
           ready");
```

```
59          System.out.println("[MongoConverter]Starting
               parsing process...");
            try {
61              parser.parse(is, handler);
            } catch (IOException e) {
63              e.printStackTrace();
            }
65          System.out.println("[MongoConverter]Parsing
               process terminated");

67          for(int i=0; i<MAIN_NODES;i++){
                printwriters[i].close();
69          }
            System.out.println("[MongoConverter]Printwriters
               closed");
71      }
    }
```

HandlerBDLP.java

```
    import netscape.javascript.JSObject;
2   import org.json.JSONObject;
    import org.xml.sax.Attributes;
4   import org.xml.sax.SAXException;
    import org.xml.sax.helpers.DefaultHandler;
6
    import java.io.PrintWriter;
8


10  public class HandlerDBLP extends DefaultHandler {

12      private PrintWriter[] writers;

14      private String current_element = "
           no_element_found_yet";
        private JSONObject current_object = new JSONObject();
16      private StringBuilder current_value = new
           StringBuilder();


18
        public HandlerDBLP(PrintWriter[] writers) {
20          super();
            this.writers = writers;
22      }


24
        public void startDocument() {
```

```
26          System.out.println("[HandlerDBLP]Starting to read
                document");
        }
28
        public void endDocument() {
30          System.out.println("[HandlerDBLP]Parsing
               terminated");
        }
32
        public void startElement(String uri, String localName
            , String qName, Attributes attributes) throws
           SAXException{
34          if(isMainNode(qName)) {
                current_object = new JSONObject();
36
                for(int i=0; i<attributes.getLength();i++){
38                  current_object.accumulate(attributes.
                       getLocalName(i), attributes.getValue(i
                       ));
                }
40          }
            else {
42              current_element = qName;
            }
44      }


46
        public void endElement(String uri, String localName,
           String qName) throws SAXException {
48          if(isMainNode(qName)){
                if(qName.equalsIgnoreCase("article")) {
                   writers[0].println(current_object);}
50              else if( qName.equalsIgnoreCase("proceedings"
                   )) {writers[1].println(current_object);}
                else if (qName.equalsIgnoreCase("
                   inproceedings")){writers[2].println(
                   current_object);}
52              else if (qName.equalsIgnoreCase("book")) {
                   writers[3].println(current_object);}
                else if (qName.equalsIgnoreCase("incollection
                   ")) {writers[4].println(current_object);}
54              else if (qName.equalsIgnoreCase("phdthesis"))
                    {writers[5].println(current_object);}
                else if (qName.equalsIgnoreCase("
                   mastersthesis")) {writers[6].println(
                   current_object);}
56              else if (qName.equalsIgnoreCase("www")) {
```

```
                writers[7].println(current_object);}
            }
        else {
            current_object.accumulate(current_element,
                current_value);
            current_value = new StringBuilder();
        }
    }

    public void characters(char[] ch,int start,int length
        ) {
        String value = null;
        value = new String(ch, start, length);
        current_value.append(value);
    }

    public void ignorableWhitespace(char[] ch,
                                    int start,
                                    int length)throws
                                        SAXException{}

    private boolean isMainNode(String qName){
        if(qName.equalsIgnoreCase("article")
                || qName.equalsIgnoreCase("proceedings")
                || qName.equalsIgnoreCase("inproceedings"
                    )
                || qName.equalsIgnoreCase("book")
                || qName.equalsIgnoreCase("incollection")
                || qName.equalsIgnoreCase("phdthesis")
                || qName.equalsIgnoreCase("mastersthesis"
                    )
                || qName.equalsIgnoreCase("www")
                || qName.equalsIgnoreCase("dblp")){
                    return true;}
        else return false;
    }
}
```

Bibliografia

[1] Poletti, Giorgio *Basi di dati: principi e strutture*

[2] T.Ray, Erik (2003) *Learning XML*

[3] M. Lenzerini(2018) *Dispense del corso di Basi di dati di Ingegneria Informatica* https://www.dis.uniroma1.it/ lenzerin/home/?q=node/44

[4] Codd, Edgar(1985) *Is your DBMS really relational* ComputerWorld

[5] Ley, Michael(2009) *Some lessons learned* PVLDB

[6] Rocco, Germano(2007) *Tesina per il corso di seminari di ingegneria del software*

[7] MongoDB Manual *Data model design* https://docs.mongodb.com/manual/core/data-model-design/

[8] MongoDB Manual *JSON and BSON* https://www.mongodb.com/json-and-bson?lang=it-it

[9] Microsoft(2018) *Dati non relazionali e NoSQL* https://docs.microsoft.com/it-it/azure/architecture/data-guide/big-data/non-relational-data

[10] Postgresql documentation https://www.postgresql.org/docs/

[11] Oracle Java Documentation *Parsing an XML File Using SAX* https://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html