UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

PROGETTO E SVILUPPO DI UN SISTEMA GESTIONALE PER LA MIGRAZIONE DI DOCUMENTI

***Relatore**:*

Prof. Riccardo Martoglia

Correlatore:

Prof. Giacomo Cabri

Candidato:

Tatiana Pusnei

Anno Accademico 2015/2016

RINGRAZIAMENTI

Ringrazio profondamente l'Ing Giacomo Cabri per avermi sostenuto e supportato, "nonché sopportato", per l'intera durata del mio percorso universitario, trasmettendomi la forza e la capacità di superare le mie difficoltà.

Ringrazio l'Ing. Riccardo Martoglia per la continua disponibilità e assistenza.

Un ringraziamento speciale agli amici, specialmente a Monica, e alla mia "famiglia adottiva" che hanno condiviso ogni momento ed emozione del mio percorso.

PAROLE CHIAVE

Base di Dati

Java

SQL

File / Documenti

Directory

Progetto

JDBC Template

Sommario

Indici delle figure ---------- *6*

INTRODUZIONE ---------- 1

PARTE I. CASO DI STUDIO ---------- 4

CAPITOLO 1: BACKGROUND ---------- 5

1.1 Specifica dei requisiti ---------- *5*

1.1.1 Requisiti iniziali ---------- 6

1.1.2 Schema concettuale del progetto ---------- 7

1.2 Dettagli concettuali ---------- *8*

1.3 Tecnologie e strumenti di sviluppo ---------- *10*

1.3.1 Il framework Spring ---------- 10

1.3.2 Il Contenitore di Spring, IoC ---------- 12

1.3.3 La componente di Spring, JDBC Template ---------- 13

1.3.4 Apache Tika ---------- 14

CAPITOLO 2. VALUTAZIONI SU TIPI DI DATI E LIBRERIE JAVA ---------- 16

2.1 Java e strutture dati ---------- *17*

2.1.1 ArrayList ---------- 18

2.1.2 LinkedList ---------- 20

2.1.3 Vector ---------- 22

2.1.4 Pila e Coda ---------- 24

2.1.5 Sintesi su le strutture dati ---------- 25

2.2 Tipi di dati per SQL ---------- *27*

2.3 Le librerie di input/output in Java ---------- *30*

PARTE II. PROGETTO ED IMPLEMENTAZIONE ---------- 35

CAPITOLO 3. CREAZIONE ARCHIVIO DATI ---------- 36

3.1 Specifica Dei Requisiti ---------- *37*

3.2 Progettazione database ---------- *37*

3.2.1. Il modello concettuale ---------- 39

3.2.2 Il Modello Entità-Relazione (ER) ---------- 40

3.2.3 Modello logico ---------- 42

3.3 Raffinamenti al database ---------- *45*

3.3.1 Il modello concettuale ---------- 46

3.3.2 Traduzione modello concettuale, il progetto logico. ---------- 47

3.3.3 Schema E-R completa ---------- 50
3.4 Implementazione nel DBMS ---------- *52*
3.4.1 Connessione al server ---------- 52
3.4.2 Operazioni su Database ---------- 53
CAPITOLO 4. IMPLEMENTAZIONE IN JAVA ---------- 60
4.1 Modello di dati ---------- *61*
4.2 Sviluppo delle classi Java ---------- *63*
4.2.1 Classe di dominio ---------- 64
4.2.2 Classi DAO ---------- 65
4.2.3 Classi RowMapper(recuperare multipli oggetti) ---------- 67
4.2.4 Recuperare singoli oggetti ---------- 68
4.3 JDBC Connection ---------- *69*
4.3.1 L'invio dei comandi ---------- 72
4.4 File di configurazione ---------- *74*
4.5 Estrazione metadati ---------- *76*
CONCLUSIONI ---------- 79
APPENDICI ---------- 81
BIBLIOGRAFIA ---------- 85

Indici delle figure

Figura 1 Schema concettuale del progetto. 8

Figura 2 Architettura Spring. 11

Figura 3 Campi di tipo char..... 29

Figura 4 Schema E-R. 42

Figura 5 Gerarchia Metadati. 47

Figura 6 Schema E-R completa..... 51

Figura 7 Tabella File. 55

Figura 8 Java Application con DB server 63

Figura 9 DB Connection 70

Figura 10 JDBC uso dei driver..... 71

Figura 11 Tika structure. 76

Introduzione

Il motivo principale che mi ha portato a scegliere un tirocinio esterno è stato il desiderio di avviarmi al futuro lavoro di squadra e conoscere le peculiarità dell’ambito lavorativo informatico. Con lo scopo di impegnarmi nella formazione come tecnico in grado di programmare, gestire e coordinare incarichi professionali, miravo anche ad un’esperienza che mi arricchisse dal punto di vista intellettuale e scientifico.

Il mio percorso di formazione ha avuto inizio grazie al tirocinio presso CASTGroup, azienda che si occupa di gestione e controllo dell'impresa, digitalizzazione di ogni processo aziendale, realizzazione di portali e software. L'ambiente aziendale, costituito da numerosi dipendenti con mansioni diverse, si è dimostrato idoneo per svolgere attività costruttive. Inoltre, l’organizzazione mi ha consentito di collocarmi velocemente e correttamente nel contesto lavorativo, fatto dovuto alla sua obbiettività, flessibilità e fiducia nel proprio personale trovati all’interno dell’azienda.

In questo elaborato di tesi si descrivono l'analisi, la progettazione e gli strumenti che hanno portato allo sviluppo di un'applicazione per la migrazione

dei documenti. L'idea parte dalla necessità di creare un programma che possa gestire opportunamente documenti provenienti da vari dispositivi di storage ed accolti in un unico database. L’insieme di dati viene sottoposto all’operazione di scrematura, la procedura implica di conseguenza la migrazione verso una destinazione prescelta dal manipolatore dell’archivio. Nel prossimo futuro si prevede un adattamento del software per il nuovo gestionale aziendale Alfresco.

In questo periodo ho avuto modo di applicare le conoscenze apprese durante lo studio universitario e impadronirmi di nuovi strumenti informatici. Le materie di Basi di Dati e Programmazione ad Oggetti sono servite come fondamento per lo sviluppo del progetto. Compiere varie analisi sui dati e scegliere le strutture dati adeguate per il progetto, ha costituito un compito delicato per me, in quanto si tratta di responsabilità che determinano l’ottimo funzionamento del programma.

Ho usato parte del framework Spring per lo sviluppo di applicazioni su piattaforma Java, utilizzando i pattern dell'Inversion of Control (IoC) e del Dependency Injection (DI) per controllare gli Enterprise JavaBeans ed il pattern DAO tramite la libreria di JDBCTemplate.

La tesi è strutturata in due parti importanti.

La prima parte è formata da due capitoli: il primo capitolo riguarda la presentazione del progetto, i dettagli concettuali e gli strumenti necessari per la produzione dell’applicazione. Il secondo capitolo tratta le valutazioni sui tipi di dati e le librerie Java che hanno avuto un impatto significante per il progetto. Lo studio sulle strutture date è stato necessario per effettuare le scelte giuste, principalmente per le caratteristiche del progetto che presentavano elementi di criticità per la coerenza dei dati e per la precisione dei risultati.

La seconda parte si suddivide in due capitoli, il terzo ed il quarto, e comprende tutti gli aspetti pratici della tesi. Prevede la descrizione del funzionamento e l'implementazione dell’applicazione sulla base dei requisiti, cioè la creazione della basi di dati e la spiegazione del codice del programma.

Il capitolo 3 tratta la parte che va dalla creazione fino alla messa in produzione del database che viene dapprima definito (mediante schema E-R e schema

logico) e poi implementato. Il software si appoggia sul database e in seguito su questo si eseguono le operazioni relative.

L'ultimo capitolo, il quarto, fornisce informazioni sul codice scritto in Java, sul software che genera il collegamento al server ed il procedimento per l'inserimento dei dati nel database con le classi di Java. I paragrafi riportano un'analisi dettagliata delle classi in Java che hanno portato alla realizzazione del software; nell'ultima sezione si descrive la modalità di estrazione dei metadati.

PARTE I. CASO DI STUDIO

Capitolo 1: Background

In questa sezione saranno presentate le richieste poste per la realizzazione del progetto. La prima fase include la situazione iniziale, cioè le specifiche dei vari requisiti necessari. Successivamente si presentano le caratteristiche del gestionale aziendale prescelto e delle tecnologie da utilizzare per compiere il lavoro. Nella seconda parte si effettua la valutazione su alcuni tipi di dati gestiti dal software. Essa illustra le scelte fatte riguardo ai compiti che il software svolge, senza entrare nei dettagli implementativi. Quest'ultima è servita per descrivere in modo completo le funzionalità principali e le caratteristiche delle librerie che sono state importanti per l'utilizzo nel progetto.

1.1 Specifica dei requisiti

Il progetto nasce dalla necessità di effettuare la raccolta di dati è diventata troppo difficile da gestire e da controllare, si rende necessario allora creare un software che possa soddisfare queste funzionalità in maniera più efficiente.

Indubbiamente ogni settore informatico affronta una situazione in cui la quantità dei dati personali (del quadro aziendale) e quelli esterni (clienti) diventa eccessiva. Il creatore stesso (chi crea un documento) non presta molta attenzione a cosa succede dopo che ha creato, nominato e salvato un documento[1]. Giustamente non è suo compito, in quel contesto lavorativo viene creato un numero cospicuo di documenti in un certo periodo di tempo e tutti sono salvati sul repository aziendale. L'idea di base sta nel gestire opportunamente la situazione, pensando prima di tutto ad una massa di dati che sia composta da migliaia di file. Inevitabilmente la situazione diventerà incontrollabile se si perde di vista quest'aspetto.

Si pone quindi la necessità di creare un piccolo sistema di gestione di documenti a livello locale (non per usarlo come sistema di produzione per i clienti), con l'utilizzo delle risorse tecniche ed informative disponibili nell'azienda. Allo scopo di rinnovare la piattaforma aziendale, si opta per un nuovo gestionale che potrebbe ottimizzare gli aspetti riguardanti la modalità di elaborazione di documenti, la facilitazione dell'accesso, il guadagno di più spazio su disco, la riduzione degli errori umani legati ai processi di gestione dei file. Il nuovo meccanismo implica la creazione di un database come supporto all'archiviazione dei documenti.

1.1.1 Requisiti iniziali

L'azienda dispone di un sistema che organizzi i dati nelle varie locazioni, una grande varietà di documenti disposti in modo casule nei vari punti del server aziendale, si tratta di migliaia di file disponibili.

Viene richiesto di creare un sistema che organizzi i dati per la preparazione dello spostamento su un singolo dispositivo di storage, che si interfacci con un software nell'elaborazione e nell'output dei risultati. Una volta copiati nel nuovo repository, l'utente deve poter avere una selezione e visualizzazione ben divisa e ben organizzata dei documenti.

[1] Con documento si intende qualsiasi tipo di file con vari scopi, sia documenti office (.doc, .xlsx,..) sia file con contenuto di un'immagine, etc.

La situazione iniziale si presenta come segue:

I vari punti dove risiedono i documenti sono:

- Server Microsoft Visual Source Safe
- Server Subversion
- Dischi condivisi

I vari step di migrazione saranno:

- Individuazione file da migrare
- Scrematura dei duplicati.
- Individuazione di massima di quante nuove aree documentali sono da creare.

Per ogni area documentale ci sarà da:

- Decisione delle regole di utilizzo e di notifica
- Configurazione dello spazio condiviso dagli utenti
- Importazione e classificazione dei documenti.
- Creazione della documentazione e delle regole dello spazio.

1.1.2 Schema concettuale del progetto

Dai requisiti elencati, finora possiamo modellare uno schema logico del progetto.

Riassumendo, si prevedeva la creazione di un sistema di gestione per la migrazione di documenti. Da tutti i punti dove sono salvati i documenti, vengono inseriti in un DBMS e sottoposti a varie operazioni ed infine i risultati desiderati saranno trasferiti all'interno del nuovo ECM [2] (repository) aziendale, che nel nostro caso è Alfresco versione 5.1.

[2] ECM –Entreprise Content Management

Presentazione del procedimento:

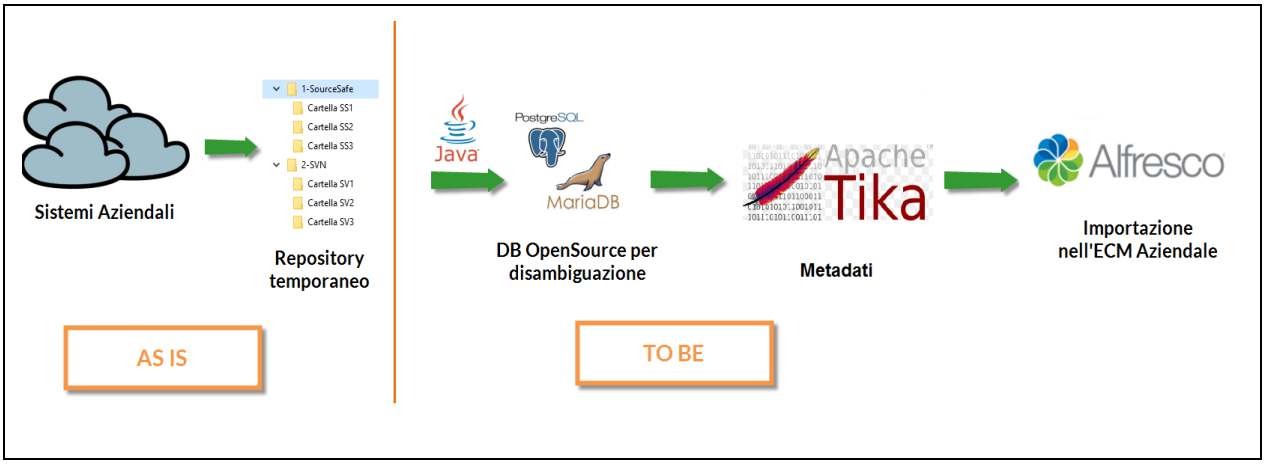


Figura 1 Schema concettuale del progetto.

1.2 Dettagli concettuali

Il software richiesto si propone come supporto alla gestione dell'attività di spostamento dei documenti. La specifica dei requisiti del sistema si riferisce alla scrematura dei dati/file caricati nel DB, che saranno poi copiati e manipolati del nuovo ECM dell'azienda.

Il beneficio principale che deriva dall'utilizzo del software è la molteplicità di operazioni che si possono effettuare senza ricorrere all'utilizzo di altri software, funzionalità che può sembrar banale, ma che offre la possibilità di: eliminazioni di duplicati, sottrazioni di dati divisi per categorie, estrazioni di metadati. In poche parole si tratta di velocizzare il procedimento di gestione di attività che in precedenza erano svolte utilizzando diversi programmi di base del computer.

1.2.1 AMBIENTI DI LAVORO

Per svolgere il compito si devono avere conoscenze di base dei linguaggi di programmazione Java e SQL. Gli ambienti di sviluppo che si decide di adottare sono Eclipse e MySQLWorkbench:

ECLIPSE

Per scrivere codice in Java si usa L'IDE Eclipse NEON, con l'installazione di JDK[3] versione 1.8. Eclipse è adeguato per progetti di qualsiasi dimensione, supportando grazie ad un'interfaccia facilmente espandibile tutte le tecniche di programmazione necessarie.

MYSQLWORKBENCH

Per creare il database con le tabelle abbiamo utilizzato il DBMS MySQLWorkbench, che si presta molto bene per la richiesta amministrativa del server.

MySQLWorkbench fornisce la modellazione dei dati, lo sviluppo SQL e gli strumenti di amministrazione completi per la configurazione del server, l'amministrazione degli utenti e il backup.

1.2.2 GLI STRUMENTI DI JAVA

Java come linguaggio prelevante sull'intero ciclo di progettazione e realizzazione del progetto sostiene a pieno le nostre richieste con il suo potente e vasto uso delle librerie e framework disponibili per l'implementazione.

Java mette a disposizione un numero arbitrario di librerie da utilizzare. Queste librerie, contenute in vari package, sono utilizzate da noi per determinati metodi o attributi comuni per semplificare e uniformare il programma.

Durante il ciclo della programmazione abbiamo usufruito delle seguente librerie:

- di input e output (java.io; java .nio);
- per attivare i livelli di log (java.apache.log4j);
- per creare applicazioni di rete (java.sql);

[3] JDK- Java Development Kit

Gli strumenti ed i framework Java sono librerie basate su applicazioni web che forniscono un'interfaccia di utilizzatore. Noi abbiamo fatto uso di alcuni di loro, funzionali alla realizzazione dell'obiettivo:

- estrazione di metadati (Apache Tika);
- realizzazione dell'applicazione e connessione DB (Spring);

1.3 Tecnologie e strumenti di sviluppo

Uno degli aspetti fondamentali quando si costruisce un'applicazione è la persistenza dei dati, per realizzare la quale, nel mondo Java, esiste una varietà di API[4] e framework. Un framework è definito da un insieme di classi astratte e dalle relazioni tra di loro. Istanziare un framework significa fornire un'implementazione delle classi astratte. Alla base di ogni framework c'è sempre una serie di librerie di codice utilizzabili in fase di linking 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. L'utilizzo di un framework impone dunque al programmatore una precisa metodologia di sviluppo del software [3, 4, 7, 11].

1.3.1 Il framework Spring

Spring è un framework open source che offre un supporto completo d'infrastrutture per sviluppare applicazioni Java robuste e complesse. Composto di cinque livelli: DAO, ORM(su AOP), Web, JEE e Core Container, la framework ha la capacità di presentare in maniera modulare tutti i suoi componenti. Prevede un ampio uso per vari tipi di progetti specializzati da anni nell'adattarsi a quasi tutti tipi di richieste e software con sicurezza e affidabilità.

[4] API –Application Programming Interface

Un altro vantaggio offerto da Spring è di poter escludere le parti del framework non necessarie all'applicazione che si sta sviluppando, includendo esclusivamente quelle utili.

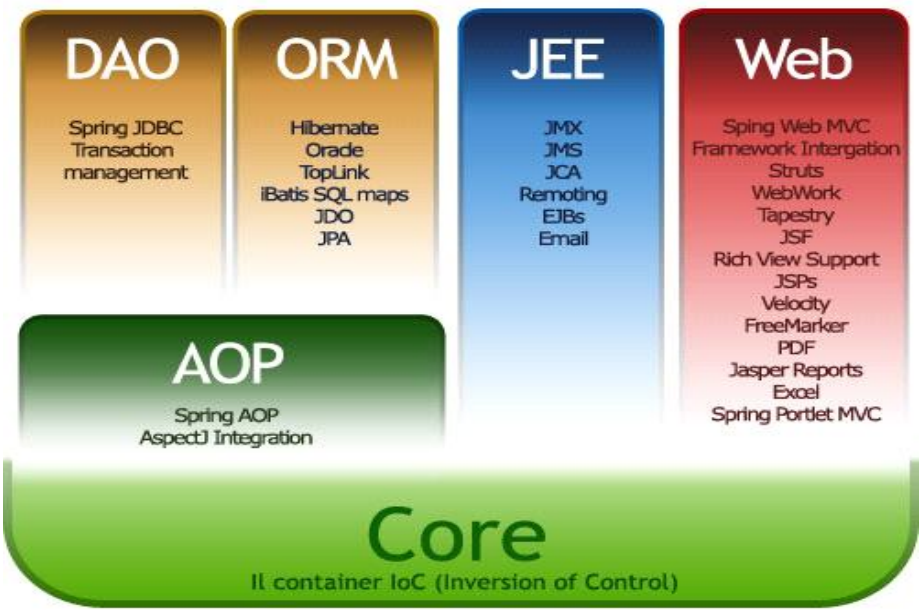


Figura 2 Architettura Spring.

Data Access/Integration Object (DAO)- Fornisce un livello di astrazione per l'accesso ai dati mediante tecnologie eterogenee tra loro come ad esempio JDBC, Hibernate o JDO[5]. Questo modulo tende a nascondere la complessità delle API di accesso ai dati, semplificando ed uniformando quelle che sono le problematiche legate alla gestione delle connessioni, delle transazioni e delle eccezioni.

Spring partecipa nel nostro contesto mettendo a disposizione tutte le sue tecnologie in un ambiente altamente flessibile, sfruttando appieno le potenzialità offerte dalla Dependency Injection e dall'Aspect Oriented. Tutte le applicazioni sono formate da oggetti che per raggiungere l'obiettivo prefissato devono collaborare tra di loro, cioè supporta la creazione di dipendenze tra gli oggetti di diverse classi.

[5] JDO- Java Data Objects

1.3.2 Il Contenitore di Spring, IoC

In Spring, gli oggetti non sono responsabili per trovare o creare gli altri oggetti di cui abbiamo bisogno per svolgere il proprio lavoro. Al contrario, Spring dispone di un contenitore che offre i riferimenti agli oggetti che collaborano tra di loro.

Dependency Injection - per riferirsi ad una specifica implementazione delle dipendenze, rivolta ad invertire il processo di risoluzione delle dipendenze, facendo in modo che queste vengono iniettate dall'esterno. L'idea di base della Dependency Injection è di avere un componente esterno (assembler) che si occupi dell'Injection. In particolare esistono tre forme di Injection:

1. **Costructor Injection**, dove la dipendenza viene iniettata tramite l'argomento del costruttore.

2. **Setter Injection**, dove la dipendenza viene iniettata attraverso il metodo "set"

3. **Interface Injection** che si basa sul mapping tra interface e relativa implementazione (non utilizzato da Spring)

L'inezione di dipendenza può essere realizzata in molteplici modi, tra cui il più semplice consiste nell'utilizzo di una factory.

Tecnicamente parlando, lo IoC è realizzato da due interfacce:

- **BeanFactory**, che definisce le funzionalità di base per la gestione dei bean

- **ApplicationContext**, che estende queste funzionalità basilari aggiungendone altre proprietà come l'integrazione con AOP.

L'interfaccia `BeanFactory` ha il compito di: creare i bean necessari per l'applicazione, inizializzare le loro dipendenze tramite l'utilizzo dell'Injection, gestirne l'intero ciclo di vita dei bean.

Gli oggetti che formano la spina dorsale della nostra applicazione e che sono gestiti dal contenitore Spring IoC sono chiamati ***bean***. Un bean è un oggetto che è creato da un'istanza, assemblato, e comunque gestito dal contenitore Spring IoC. La definizione di un bean rappresenta la creazione di un'istanza di un oggetto, raggiungibile nel contesto dell'applicazione tramite il suo riferimento all'interno del

contenitore. I bean, contenuti nel contenitore, sono gestiti con i metadati di configurazione che si fornisce dal contenitore Spring in forma di definizioni XML.

Esempio di dichiarazione di un bean nel file contenitore:

```
<bean id="helloWorld"

class="com.tutorialspoint.HelloWorld">

<property name="message" value="Hello World!"/>

</bean>
```

Spieghiamo il contenuto ed il significato dell'esempio:

Gli attributi :

- **id**: è utilizzato per assegnare un nome identificativo all'istanza del bean. Tale nome sarà utilizzato per accedere alle proprietà della classe all'interno del programma. ·

- **class:** con quest'attributo si specifica il nome della classe del bean da istanziare o, se il bean esiste già, richiamare un'istanza precedentemente creata. È l'attributo che di solito si utilizza.

- **property_name**: indica i casi in cui si vuole accedere ad un bean già istanziato e del tipo specificato.

- **ref**: rappresento l'id di un bean con cui la classe ha creato una dipendenza. Fa riferimento alle proprietà del bean passato.

1.3.3 La componente di Spring, JDBC Template

JDBC è l'API che permette al programma Java di accedere alle risorse di un database. È definita nel package java.sql (javax.sql per le funzionalità server), ed è interpretata come Java DataBase Connectivity. La componente JDBC gioca il ruolo più importante per la nostra applicazione, senza di essa non sarebbe nemmeno possibile realizzarla.

Nello sviluppo dell'applicazione la componente si occupa dell'intera gestione dei processi di accesso ai dati, ovvero:

- Richiesta della connessione al datasource
- Creazione del PreparedStatement e valorizzazione dei parametri
- Esecuzione del PreparedStatement
- Estrazione dei risultati dal ResultSet (in caso d'interrogazioni)
- Gestione delle eccezioni
- Chiusura della connessione

Spring offre diverse possibilità per la persistenza dei dati mediante JDBC, la principale della quale è l'utilizzo della classe `Jdbctemplate.` Questa classe implementa l'intero processo di accesso ai dati attraverso template methods, rendendo possibile la personalizzazione di tutte le fasi di tale processo mediante l'ovverride dei metodi specifici. Jdbctemplate viene utilizzata all'interno di un'implementazione DAO, viene configurata in un contenitore Spring, IoC, e dato a DAO come riferimento di bean. Nel nostro programma sono le classi con il nome che finisce in DAO ad implementare le funzionalità di JDBC.

Template methods sono i metodi che gestiscono le parti fisse di un processo JDBC- controllo delle transazioni, gestione delle risorse e gestione delle eccezioni. Contemporaneamente, gestisce anche le specifiche di accesso di dati che si riferiscono all'applicazione: creare istruzioni, associare parametri e coordinare i ResultSet.

Ogni metodo che richiede l'accesso ai dati implica la realizzazione di connessione al database. Le istruzioni specifiche all'interazione con DB permettono l'esecuzione di comandi SQL del tipo: CREATE, INSERT, UPDATE, DELETE e SELECT.

1.3.4 Apache Tika

Tika viene utilizzata nello step finale, dopo aver concluso con i primi due passaggi potremo proseguire con il terzo, l'estrazione dei metadati [13, 14].

Il toolkit Apache Tika è una libreria open source che svolge la funzione di detector del contenuto di un documento analizzando ed identificando le sue proprietà individuale, e di estrarre informazioni (incluso metadato e strutture dati) da migliaia di tipi di file. Tika si adatta a diversi scopi e per questo fornisce gli strumenti necessari per eseguire registrazioni del contenuto ed estrazioni dati. La difficoltà rimane sulle librerie che sono fornite per ogni tipo di documento, questo diventa pesante e arduo quando si tratta di elaborare un numero indefinito di tipi di file. Tika può estendere le sue funzionalità per quasi tutti i tipi di file, dispone di `parser` in cui si indica come possono effettuare le estrazioni di contenuto e metadati dal documento. Tika, nasce da Apache Lucene, un progetto per la ricerca, al momento si posiziona sui livelli più alti che la Fondazione Apache Software dispone. Viene sfruttato in un contesto largo di operazioni che offre una rapida percezione per contenuti eterogenei, diventa uno strumento precursore per eseguire analisi automatiche di testo.

Siccome rileva ed estrae metadati e testo da oltre un migliaio di tipi di file diversi, per noi il suo ruolo sarà quello di utilizzarlo nel programma Java per ottenere i metadati.

Il progetto presuppone l'estrazione dei metadati, in particolare si richiede di effettuarla solo su alcuni tipi di documenti. Primo, perché dopo il processo di inserimento dati nel database avremo a disposizione vari tipi di documenti dei quali alcuni non ci interessano, in secondo luogo, non su tutti i tipi di documenti è possibile ottenere i metadati, inoltre sappiamo in partenza quali documenti verranno gestiti, sono tutti quelli il cui contenuto può servire come fonte di informazione. In particolare ci interessano documenti della categoria office, di testo e pdf. Di tipo office sarebbero - doc, docx, xlsx; testuali- odt; acrobat reader - pdf;

Scaricando le librerie di Tika si crea la classe con cui si estraggono i metadati per ogni tipo di file. Come affermato in precedenza, Tika è uno strumento ottimo ma ancora non è completo dal punto di vista di `parse` per il tipo documento, cioè non dispone di una classe/metodo che potrebbe incapsulare proprietà simile dei file per poter effettuare meno operazioni possibili per l'elaborazione dei metadati. Questo fatto ha consentito di scrivere un codice ripetitivo per ogni tipo di documento.

Capitolo 2. Valutazioni su tipi di dati e librerie Java

Scrivere un buon codice significa spendere poco tempo, sprecare poca memoria del computer ed avere un risultato impeccabile. Aver conoscenza di come viene sfruttato ogni byte del computer è importante per essere un ottimo programmatore. Non è banale neanche se si ha da fare con poche righe di codice. Per quanto riguarda il nostro compito andremo a studiare i tipi di dati che rappresentano un eventuale rischio (per l'uso eccessivo di spazio di memoria). Il rischio è di ottenere risultati incoerenti e in alcuni casi di affrontare un'insufficienza di spazio di memoria del dispositivo. Spieghiamo perché tanta attenzione per la scelta di tipi di dati e la loro l'importanza per il nostro compito.

Per quanto riguarda le librerie Java parleremo in particolare di quelle che lavorano con i file, ove le classi per realizzare l'input – output in Java e perché abbiamo

usufruito delle funzionalità dei due package, sia `java.io` che `java.nio`, nel nostro programma.

2.1 Java e strutture dati

Inizialmente si pone il problema di creare un programma scritto in Java che abbia come risultato la lista dei tutti i file, presenti in una certa directory indicata dall'utente. I documenti trovati devono essere elencati in output insieme alla directory appartenente. Si chiede inoltre di effettuare un'ulteriore stampa con le caratteristiche di ogni documento trovato.

Si discute sullo scopo del progetto e si tratta il problema di dover usare un tipo di struttura dati per rappresentare la collezione d'elementi cioè un insieme di file. Si deve scegliere tra ***ArrayList, LinkedList, Stack e Vector.***

Quale di questi potrebbe soddisfare la nostra esigenza? Come si comporta ognuna di loro per le operazioni di inserimento, aggiornamento e ricerca degli elementi nella sequenza di dati (nella collezione)? Analizzeremo anche il costo delle operazioni.

Si cerca di effettuare la migliore scelta dal punto di vista di efficienza, del tempo di esecuzione e del costo computazionale. Perché interessano tanto queste caratteristiche? Perché si tratta di lavorare con più di 250 mila file alla volta, e di effettuare operazioni di inserimento di file in una basi di dati, di ricerca dell'elemento, ed anche di un'ulteriore modifica di posizione dei dati. Lavoro per niente facile da gestire per il computer quando si tratta di allocazione della memoria.

Molto spesso, una computazione si basa su di una o più strutture di dati, di vario tipo: insiemi (collezioni), code, stack, tabelle, liste, alberi, etc. La scelta si basa sui vantaggi che ognuna di loro presenta. Questa è strettamente legata all'algoritmo che verrà applicato. Una buona struttura dati influisce inevitabilmente sull'efficienza degli algoritmi che la manipolano.

Una struttura dati è un "contenitore" di informazioni che si caratterizza per i suoi attributi "logici", quali modalità di accesso, dimensionamento, organizzazione dei suoi componenti. O per gli attributi "fisici", quali i tempi di accesso ad un componente e la persistenza delle informazioni.

Nel package java.util si trovano le classi di utilità, classi che Java predispone in automatico. Esso mette a disposizione 34 classi e 13 interfacce che implementano alcune tra le strutture dati più comuni, alcune operazioni su date e sul calendario, e altro ancora.

Il contenuto del package java.util

Collection	Comparator	Enumeration	EventListener
Iterator	List	ListIterator	Map
Map.Entry	Observer	Set	SortedMap
SortedSet			

Queste interfacce stabiliscono alcune proprietà delle nostre strutture dati, esse vengono implementate in alcune delle seguenti classi:

2.1.1 ArrayList

Una classe di questo tipo viene detta *raccolta contigua,* è una lista rappresentata mediante array, cioè ha un campo di tipo array, nelle cui singole posizioni vengono memorizzati i riferimenti agli elementi della raccolta: si tratta, quindi, di una raccolta contigua (lista) la cui capacità è la lunghezza dell'array usato per memorizzare gli elementi della lista, e vale almeno la lunghezza della lista. Quando vengono aggiunti elementi alla lista, la sua capacità varia automaticamente.

ArrayList <E> consente l'accesso veloce in lettura casuale, in modo da poter recuperare qualsiasi elemento in tempo costante. L'aggiunta e/o la rimozione a parte l'ultimo elemento, necessita lo spostamento di tutti i *n* elementi in avanti (cancellazione – operazioni per colmare le lacune), risulta un'operazione per l'*ArrayList* che ha complessità O(n) nel peggiore dei casi, ma costante, in media.

Per dimostrare come ogni tipo di classe tratta le strutture di dati, presentiamo i metodi della classe, prevalenti per l'utilizzo nel progetto, che effettua le operazioni.

Codifica ArrayList:

```
     public ArrayList(int initialCapacity) {
      super();
      if (initialCapacity < 0)
       throw new IllegalArgumentException("Illegal Capacity:
       "+ initialCapacity);
       this.elementData = new Object[initialCapacity];
     }

//costruttore vuoto con la capacità iniziale di 10 elementi
     public ArrayList() {
       this(10);
     }

//costruisce una lista di elementi dello stesso tipo.
    public ArrayList(Collection<? extends E> c) {
       elementData = c.toArray();
       size = elementData.length;
       if (elementData.getClass() != Object[].class)
         elementData = Arrays.copyOf(elementData, size,
         Object[].class);
       }

/* Rifila la capacità dell' ArrayList per essere dimensionata
  all'attuale lista.
  Un'applicazione può utilizzare quest'operazione per ridurre il
contenuto di un'istanza ArrayList
*/
     public void trimToSize() {
       modCount++;
       int oldCapacity = elementData.length;
        if (size < oldCapacity) {
          elementData = Arrays.copyOf(elementData, size);
         }
      }

/*incrementa la capicatà dell'ArrayList, quando necessario, per
  assicurare che può  contenere al minimo il numero di elementi
  specificato in OldCapacity; minCapacity- capacità minima richiesta.
*/
    public void ensureCapacity(int minCapacity) {
      modCount++;
      int oldCapacity = elementData.length;
```

```
      if (minCapacity > oldCapacity) {
        Object oldData[] = elementData;
        int newCapacity = (oldCapacity * 3)/2 + 1;
        if (newCapacity < minCapacity)
        newCapacity = minCapacity;
       elementData = Arrays.copyOf(elementData, newCapacity);
     }
     }

    public int size() {
       return size;
     }
```

Il metodo `ensureCapacity` viene invocato nel caso in cui lo spazio allocato inizialmente non fosse sufficiente, tramite la variabile `NewCapacity` ci permette di allocare nuova memoria.

Il calcolo nella riga "`newCapacity = (oldCapacity * 3)/2 + 1` " risulta che se la mia collezione inizialmente, per esempio, ha una capacità di 100 elementi, una nuova allocazione di memoria mi offrirebbe uno spazio di (100*3)/2+1. Il risultato è soddisfacente sembra anche conveniente.

2.1.2 LinkedList

`LinkedList` <E> permette inserimenti o rimozioni utilizzando iteratori a tempo costante, con l'accesso sequenziale degli elementi. Per cercare un elemento si deve scorrere la lista d'elementi avanti o indietro, cioè risulta che per trovare una posizione nella lista si richiede tempo proporzionale alla dimensione della lista. La documentazione Java specifica che "le operazioni che lavorano con gli indici di una lista, scorrano la lista dall'inizio o dalla fine, qualunque sia più vicino", tali metodi hanno complessità O (n / 4), in media, anche se O (1) per l'indice uguale a 0 (zero).

Un altro vantaggio nell'utilizzo di una *`LinkedList`* emerge quando si effettua inserimento o cancellazione di un elemento in testa alla lista, tale operazione ha complessità O (1), mentre per *`ArrayList`* O (n). Questa può essere una buona alternativa per *`LinkedList`* per aggiungere e/o rimuovere un elemento dalla testa,

comunque non è un'ottima opzione per il nostro obbiettivo. Inoltre, se si dispone di elenchi di grandi dimensioni il costo computazionale per la ricerca di un elemento diventa troppo alto, si deve invece tenere presente che l'utilizzo della memoria della classe *LinkedList* è diverso. Tuttavia, *ArrayList* occupa la quantità di memoria che viene allocata per la capacità, indipendentemente dal fatto che sono aggiunti elementi o no.

Codifica LinkedList:

```
public class  LinkedList<E>
  extends AbstractSequentialList<E> implements List<E>,
             Deque<E>, Cloneable, java.io.Serializable
 {
private transient Entry<E> header = new Entry<E>(null, null
, null);
private transient int size = 0;
public E  removeFirst() {
  return remove(header.next);
 }
public E  removeLast() {
  return remove(header.previous);
 }
public void  addFirst(E e) {
  addBefore(e, header.next);
 }
//Appends the specified element to the end of this list.
public void  addLast(E e) {
  addBefore(e, header);
 }
 //Returns true if this list contains the specified element.
public boolean  contains(Object o) {
   return indexOf(o) != -1;
 }
// Returns the number of elements in this list.
public int  size() {
   return size;
 }
 // Appends the specified element to the end of this list.
public boolean  add(E e) {
   addBefore(e, header);
   return true;
   }
public void  addFirst(E e) {
```

```
        addBefore(e, header.next);
        }

// Appends the specified element to the end of this list.
     public void  addLast(E e) {
       addBefore(e, header);
      }

//Returns true if this list contains the specified element
     public boolean  contains(Object o) {
       return indexOf(o) != -1;
      }
//Returns the number of elements in this list.
     public int size() {
        return size;
      }
//Appends the specified element to the end of this list.
     public boolean add(E e) {
         addBefore(e, header);
        return true;
      }
```

Dove < E > è il tipo di elemento passato alla collezione.

La struttura dati non prevede un calcolo dell'allocazione della memoria. Il costo computazionale invece si calcola in base alle operazioni da effettuare più frequentemente su di essa:

- get(int index) ha costo O(n)
- add(E element) haO(1)
- add(int index, E element) ha O(n)
- remove(int index) ha O(n)
- Iterator.remove() haO(1) – il vantaggio di LinkedList
- ListIterator.add(E element) ha O(1) – il vantaggio di LinkedList

2.1.3 Vector

Un `Vector` dotato di un indice è capace di crescere dinamicamente come la `Lista` sarebbe ideale per risolvere problemi che devono trattare i dati sia in sequenza (lista) che in modo indicizzato (array).

Nel codice si nota che `Vector` implementa anche l'interfaccia `List` ed è simile ad *ArrayList*. La prima differenza è che vector è sincronizzato, quindi è thread-safe, la seconda è che `Vector` contiene molti metodi che non fanno parte dalla collezione. A causa di questo, è anche leggermente più lento di *ArrayList*.

`Vector` si rivela molto utile se non si conosce la dimensione della array in anticipo o semplicemente bisogno di uno che può cambiare dimensioni per tutta la durata del programma.

Codifica Vector:

```
     public class Vector<E>
       extends AbstractList<E> implements List<E>, RandomAccess,
          Cloneable, java.io.Serializable
      {

//initialCapacity la capacità iniziale del vector
    public Vector(int initialCapacity, int capacityIncrement) {
      super();
      if (initialCapacity < 0)
        throw new IllegalArgumentException("Illegal Capacity:
        "+ initialCapacity);
        this.elementData = new Object[initialCapacity];
        this.capacityIncrement = capacityIncrement;
      }

//metodo che calcola la nuova capacità
      private void ensureCapacityHelper(int minCapacity) {
      int oldCapacity = elementData.length;
      if (minCapacity > oldCapacity) {
        Object[] oldData = elementData;
        int newCapacity = (capacityIncrement > 0) ?
        (oldCapacity + capacityIncrement): (oldCapacity * 2);
      if (newCapacity < minCapacity)
        newCapacity = minCapacity;
      }
    elementData = Arrays.copyOf(elementData, newCapacity);
      }
      }
```

Il metodo `ensureCapacityHelper` aumenta la capacità di questo vettore, se necessario, per assicurare che possa contenere almeno il numero di componenti specificati dall'argomento minima capacità. Al momento della creazione di un `Vector`, alla variabile d'istanza `elementData` viene assegnato un array di oggetti la cui

dimensione dipende dal costruttore utilizzato, il valore di default per `initialCapacity` è pari a 10;

Il metodo aumenta la capacità del vector, se necessario, per assicurare che possa contenere almeno il numero di componenti specificati dall'argomento minima capacità.

Se la capacità attuale di questo vettore è inferiore `minCapacity`, quindi la sua capacità è aumentata sostituendo il vechio vettore, mantenuto nel campo elementData, con uno nuovo più grande. La dimensione del nuovo array di dati sarà la dimensione più vecchia `capacityIncrement`, `(oldCapacity+capacityIncrement)`, se il valore dei `capacityIncrement` è minore o uguale a zero, nel tal caso la nuova capacità sarà due volte la capacità precedente `(oldCapacity * 2)`; ma se questo nuovo formato è ancora minore di `minCapacity`, allora la nuova capacità sarà `minCapacity`.

Da quelle analizzate la struttura dati *`Vector`* si rivela molto conveniente dal punto di vista d'accesso agli elementi e calcolo dimensione per la nuova allocazione di memoria. Si preferisce evitare anche il *`Vector`* a favore di *`ArrayList`* per la sincronizzazione dei metodi da parte del *`Vector`*, che non è quello che ci interessa al momento nel nostro programma, e poi si può fare in modo esplicito al bisogno.

2.1.4 Pila e Coda

Due ulteriori strutture dati usate in programmazione sono la *`Coda`* e la *`Pila`* (in inglese queue e stack). Queste due strutture dati sono di norma utilizzate come

"intermediari" tra un processo produttore di dati e un processo consumatore. In qualche modo sono analoghe al concetto di stream. Infatti, uno stream è l'entità che mette in comunicazione due processi. Allo stesso modo pila e coda sono utilizzate in programmazione per collegare un processo produttore e uno consumatore: Si nota che le strutture dati *`Pila`* e *`Coda`* hanno in lettura un comportamento molto diverso rispetto a *`List`*, *`Vector`*. Pila, Coda sono strutture nelle quali la lettura è distruttiva, nel senso che leggendo il dato lo si cancella/consuma. *`List`*, *`Vector`* sono contenitori senza lettura distruttiva, cioè consentono di leggere le informazioni più volte senza eliminarle.

Codifica Stack:

```
      public class Stack<E> extends Vector<E> {
       public  Stack() { }

//Pushes an item onto the top of this stack. This has exactly the same
//effect as: addElement(item)
        public E  push(E item) {
          addElement(item);
          return item;
      }

//removes the object at the top of this stack and returns that object
//as the value of this function. (the last item of the Vector object).
        public synchronized E pop() {
          E obj;
          int len = size();
          obj = peek();
          removeElementAt(len - 1);
         return obj;
        }

//Looks at the object at the top of this stack without removing it
from the stack. (the last item of the Vector object).
        public synchronized E peek() {
        int  len = size();
        if (len == 0)
          throw new EmptyStackException();
      return elementAt(len - 1);
       }
  }
```

Le due strutture dati ereditano la modalità di gestione della memoria da *Vector*.

2.1.5 Sintesi su le strutture dati

Come risultato del confronto possiamo decidere quali strutture dati dobbiamo scegliere da usare nel nostro algoritmo in Java.

Facendo il confronto tra le strutture dati analizzate possiamo arrivare alla conclusione che per effettuare le operazioni necessarie al nostro programma la struttura dati più conveniente è *l'ArrayList*.

La capacità iniziale di default di un *ArrayList* è piuttosto piccola (10 da Java 1,4-1,8). Per evitare l'alto costo di ridimensionamento basta costruire *l'ArrayList* con una capacità iniziale più elevata. L'analisi *l'ArrayList* rimane la struttura che si imposta più economica nei termini di spazio di memoria. Le dimensioni dell'*ArrayList* si raddoppiano al raggiungimento dei limiti. Il fatto sta che anche se la capacità diventa due volte più grande, *ArrayList* non occuperà spazio in più di quanti elementi dispone al momento. Non sarà di conseguenza un utilizzo di tutto lo spazio allocato, ma soltanto di quello necessario.

Come viene inizialmente definita la struttura dati del nostro programma

```
public class FindFile {
private ArrayList<String> listDir;
private String pathDelFile=null;
     public FindFile() {
          listDir = new ArrayList<String>();
          pathDelFile="C:\\Documents";
     }
```

L'attributo `listDir` e di tipo *ArrayList* con elementi di tipo stringa. Nell'andare avanti l'attributo assume vari valori di tipo File che nel nostro programma sono i percorsi di un file/directory; Java riconosce una path se è incluso tra le doppie virgolette esattamente come viene definita una stringa. Il metodo `readPath` effettua la ricerca di file documento, prendendo in ingresso una stringa che rappresenta il nome della directory, all'interno della quale si fa la ricerca. Nella ricerca l'attributo `listDir` ha come elementi i path delle directory in cui trova almeno un file. Gli elementi nel contenitore sono memorizzati in un array di tipo *File*. *File* e una classe in Java che rappresenta oggetti di tipo file. Dai costruttori che offre la classe *File* si usa quello che accetta in ingresso una path in quanto in seguito ne avremo bisogno. La variabile `inFile` serve per rintracciare dentro `inDir` (array di tipo *File*) se un file è una directory oppure un file documento.

2.2 Tipi di dati per SQL

Nella modellazione del database si prevede la creazione delle tabelle e la scelta dei campi da inserire per ognuna di queste. I campi sono, di fatto, le proprietà dei nostri dati che rappresentano le colonne della tabella, che siano di tipo intero o di tipo carattere si deve ad ogni modo analizzare il tipo ed il limite di rappresentabilità di ogni dato.

Per un'efficace progettazione di un database, è essenziale considerare la scelta del tipo di "contenitore" dei propri dati. Con " contenitore" intendiamo il campo che conterrà il dato. Una corretta definizione del tipo di dato nella colonna può rendere un database veloce ed economico. Studiamo la rappresentazione di un tipo di dato contenente una sequenza di caratteri, che è un campo di'interesse per il nostro progetto.

I requisiti del progetto prevedono inserimenti di campi che poterebbero avere dimensioni molto grandi. Si tratta d'inserimento dati che si riferiscono ai nomi di oggetti che assumono dimensioni notevolmente grandi. La sequenza di caratteri deve poter essere contenuta in un solo campo. Il campo `directory_sorgente` serve come esempio.

A questo proposito MySQL offre le seguente tipologie di dati per rappresentare i caratteri / sequenze di caratteri nominate anche stringhe, dimostrazione nella tabella 1.

Lo studio dei tipi di dati consiste nel fatto che il compito prevede operazioni sul database, specialmente istruzioni di update e/o insert, che SQL server deve gestire. Il problema è che SQL non può gestire un insieme di campi la cui somma sia superiore a circa 2 gigabyte.

Tipo	Dimensioni massime	Memoria occupata
CHAR(n)	255 byte	n byte
VARCHAR(n)	255 byte	LEN + 1 byte
TINYTEXT	255 byte	LEN + 1 byte
TINYBLOB	255 byte	LEN + 2 byte

TEXT	65535 byte	LEN + 2 byte
BLOB	65535 byte	LEN + 3 byte
MEDIUMTEXT	1.6 Mb	LEN + 3 byte
MEDIUMBLOB	1.6 Mb	LEN + 3 byte
LONGTEXT	4.2 Mb	LEN + 4 byte
LONGBLOB	4.2 Mb	LEN + 4 byte

Tabella 1: Tipi di dati SQL

Nella tabella 1 abbiamo tutti tipi di dati testuali, che SQL dispone, con la dimensione massima e spazio che ognuno di loro occupa nella memoria. Tra gli elencati escludiamo l'uso di MediumBlob, MediumText, LongText, LongBlob a causa del fatto che sono troppo larghi come dimensione ma anche perche non li consideriamo adatti al nostro compito.

I tipi TEXT e BLOB[6] consentono di memorizzare grandi quantità di dati. TEXT è utilizzato per dati di tipo testuale, mentre BLOB è utilizzato per ospitare dati *binary* (ad esempio il sorgente di un'immagine).

TEXT ha disponibilità di spazio per un campo, quasi illimitata. Il problema che rappresenta il TEXT è che non permette di compiere operazioni di ricerca ed ordinamento sul contenuto del campo, proprietà necessaria per lo sviluppo del nostro database.

CHAR e VARCHAR rappresentano i tipi che occupano poca memoria e sono tra i più utilizzati. La differenza tra questi due tipi è data dal fatto che mentre VARCHAR ha lunghezza variabile, CHAR ha lunghezza fissa. Questo significa che in una colonna CHAR(10) tutti i valori memorizzati saranno lunghi 10 byte anche se costituiti da 3 soli caratteri (cosa che assolutamente non accade con VARCHAR essendo la memoria occupata pari alla lunghezza del testo + 1).

La lunghezza di una variabile di tipo VARCHAR come vediamo può variare da 0 a 255 per la versione minore di MySQL 5.0.3 e da 0 a 65.535 per le versioni ulteriori. La nostra versione del DBMS è 5.6.30.

[6] BLOB-Binary Large OBject

Con la versione recente possiamo tranquillamente definire il capo `directory_sorgente` come VARCHAR (N). Ed il valore del N?

Siccome dobbiamo rappresentare una stringa che potrebbe contenere fino a 200 caratteri si puo mettere un valore più grande del numero di caratteri.

In MySQL la dimensione massima del tipo VARCHAR è la dimensione della riga dalla tabella(65.535 bytes) che viene condivisa fra tutte le colonne ed il totale di caratteri utilizzati, cioè si deve vedere quante colonne di tipo varchar sono presenti nella nostra tabella e dividere il massimo disponibile tra di loro.

Per esempio una variabile dichiarata c1 VARCHAR (21844) (valore massimo) viene accettata dalla tabella con successo. Invece una lista di variabili dello stesso tipo diminuisce la dimensione massima che una tabella puo contenere. Ad esempio con Table XXX (c1 VARCHAR (8000), c2 VARCHAR (8000), c3 VARCHAR (8000)) la creazione non avviene, viene interrotta con un errore. L'errore è causato dalla dimensione della riga che eccede i limiti disponibili.

Infine abbiamo scelto di rappresentare i quattro campi della tabella come nella figura 3.

Table Name: file Schema: **alfimporter**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
idFILE	INT(11)	✓	✓	✓		✓		✓		
NOME_FILE	VARCHAR(200)		✓							
CREATION_TIME	DATETIME		✓							
LAST_MODIFIED	DATETIME									NULL
SIZE_KB	INT(11)									NULL
daTRASFERIRE	INT(11)									'0'
DIRECTORY_SORGENTE	VARCHAR(9000)									NULL
DIRECTORY_DESTINAZIONE	VARCHAR(9000)									NULL
IS_REGULAR	TINYINT(4)									NULL
IS_SYMBOLIC	TINYINT(4)									NULL
IS_OTHER	TINYINT(4)									NULL
LAST_ACCESSED_TIME	DATETIME									NULL
FILE_EXTENTION	VARCHAR(255)									NULL

Figura 3 Campi di tipo char

Nell'immagine vengono evidenziati i campi di tipo VARCHAR della tabella, ad ognuno di loro abbiamo assunto i valori necessari/disponibili per l'elaborazione, tale che la somma dei quattro campi non superi i limiti di 21844 byte.

Per la valutazione economica delle informazioni si intende la scelta del tipo dato per il campo nella tabella e la dimensione necessaria instanziata alla sua definizione.

Serve per poter rappresentarlo nel DBMS in uno spazio di memoria sufficiente ma anche preciso per non sprecare memoria. Si tratta di prevedere situazioni di sicurezza computazionale, in cui i nostri dati non vengono persi per l'insufficienza di spazio del campo di definizione.

2.3 Le librerie di input/output in Java

Una libreria, in informatica, è un insieme di funzioni o strutture dati predefinite e predisposte per essere utilizzate in un programma attraverso un opportuno collegamento.

Lo scopo delle librerie software è fornire una collezione di entità di base pronte per l'uso ovvero riuso di codice, evitando al programmatore di dover riscrivere ogni volta le stesse funzioni o strutture dati e facilitando così le operazioni di sviluppo e manutenzione. Questa caratteristica si inserisce quindi nel più vasto contesto del riuso all'interno di programmi e applicazioni ed è presente in quasi tutti i linguaggi. I vantaggi principali derivanti dall'uso di un simile approccio sono i seguenti:

- Si può separare la logica di programmazione di una certa applicazione da quella necessaria per la risoluzione di problemi specifici, quali il calcolo di funzioni matematiche o la gestione di collezioni;

- Le entità definite in una certa libreria possono essere riutilizzate da più applicazioni;

- Si può modificare la libreria separatamente dal programma, senza limiti alla potenziale vastità di funzioni e strutture dati man mano disponibili nel tempo.

Per usufruire delle utilità delle librerie si usa la sintassi: **import nome_package.nome_classe**, dove si indica all'interprete che verremo a riutilizzare i metodi di tale classe, quindi sfrutteremo le sue proprietà nella classe. Illustreremo

all'inizio di ogni sotto paragrafo, che spiega l'uso della libreria, l'import delle librerie e quali classi abbiamo usato per il nostro compito.

2.3.1 JAVA.IO

```
import java.io.*;
```

Avendo a che fare con dei file nel nostro compito dobbiamo usare delle librerie adatte che ci serviranno per scrivere il codice che gestisce questo lavoro. Per quanto riguarda il nostro compito dalle utilità di Java ci interessa sapere quali classi possiamo includere per gestire gli oggetti di tipo file.

In effetti, cos'è un file? È una sorgente (o un deposito) d'informazioni. Per definirlo nel codice invece deve essere indicato il suo nome come una stringa. Non è una stringa semplice, ma è una stringa che ha come contenuto un percorso di un documento che ci conduce, secondo il nome definito, a localizzarlo nelle risorse del computer.

In Java viene considerato come un tipo di dato che può essere definito attraverso la classe che si trova nella libreria del package java.io.

Nella nostro compito l'uso diffuso dell'interfaccia `File` si incontra nella classe `FindFile`. La classe `File` che è una rappresentazione astratta di percorsi di file e directory, interessa in esclusivo perché, secondo il compito, dobbiamo trovare documenti presenti in una cartella e identificarli tramite le loro proprietà.

Per identificare le proprietà usiamo i metodi della classe che offrono la possibilità di ottenere informazioni supplementari sul file come ad esempio il nome, il path, la dimensione, la data di creazione ed altre di cui parleremo nel prossimo capitolo.

Di seguito sono riportati i punti importanti sulla classe `File`:

- Le istanze possono o non possono denotare un oggetto effettivo file system, ad esempio un file o una directory. Se indica un oggetto, esso si trova in una partizione. Una partizione è una porzione di specifico del file system.

- Un file system può implementare restrizioni per determinate operazioni sull'oggetto effettivo del file system, ad esempio lettura, scrittura ed esecuzione. Queste restrizioni sono conosciute collettivamente come le autorizzazioni di accesso.

- Le istanze della classe *File* non sono modificabili; vale a dire, una volta creato, il nome del percorso astratto rappresentato da un oggetto file non cambierà mai.

Il metodo `readPath` contiene un costruttore della classe *File* che riceve in ingresso il path del file, l'argomento ha come valore il nome della directory nella qualle si esegue la scansione. In effetti, nel costruttore la creazione di un'istanza di *File* non genera la creazione fisica del file sul disco fisso, però crea un oggetto di tipo file che trova oggetti simili, andando e ricercare anche nelle sottocartelle contenuti nella directory, i file trovati vengono aggiunti all'arraylist `isDir`**,** come una sequenza di oggetti di tipo file identificati dal percorso relativo.

```
public void readPath() {
File directoryInizio = null;
directoryInizio = new File(pathDelFile);

  if (directoryInizio.isDirectory()) {
    listDir.add(pathDelFile);
  }else
      while (!listDir.isEmpty()) {
      File toCheck = new File(listDir.remove(0));

    File[] inDir = null;
       inDir = toCheck.listFiles();
      }
```

2.3.2 JAVA.NIO

```
import java.nio.channels.FileChannel;
import java.nio.file.Files;
import java.nio.file.attribute.BasicFileAttributes;
```

La nuova libreria di ingresso/uscita (NIO) è stato introdotto con il JDK 1.4. Migliorando alcune caratteristiche di I/O, NIO fornisce codice standard ad alta velocità basata sui blocchi di dati. Definendo classi per contenere ed elaborare blocchi di dati, NIO sfrutta le ottimizzazioni di basso livello che I/O non supportava, senza l'utilizzo di codice nativo.

La differenza maggiore tra i due è nel modo in cui il dato viene impachettato e trasmesso. Se per un I/O si sposta un byte alla volta e l'output consuma un byte alla volte, allora NIO sposta blocchi di dati alla volta.

Gli oggetti coinvolti nelle operazioni d'input/output del NIO sono:

■ uno **stream** che incapsula il file fisico sul qaule effettivamente si opera: FileOutpurStream, che incapsula un file che può essere scritto da un canale, FileRandomAccesFile, per l'accesso casuale (non sequenziale) ai file, che consente di usage un solo canale sia per scrivere che per leggere.

■ uno o più **buffer** sui quali si scrivono o dai quali si leggono i dati e servono quindi per mantenere i dati in memoria.

■ un **canale** che fornisce la connessione col file ed effettua le operazioni di scrittura e lettura dei dati servendossi di uno o più buffer: la classe che lo rappresenta è `FileChanel` dal pacchetto `java.nio.channels`.

■ un**'istanza** di classe `File`, contenente il percorso che identifica un file o una cartella.

I multipli canali che gestisce NIO sono in effetti dei thread.

NIO può essere molto vantaggioso per i motivi:

■ per gestire centinaia di connessioni aperte contemporaneamente, ognuno invece può gestire poca quantità di dati.

■ tenere aperte connessioni con moltissimi computer (P2P usando un singolo thread).

Abbiamo comunque sfruttato le proprietà sia delle librerie I/O che NIO perché ognuno di loro ha un suo ruolo nel nostro programma. Anche se parlando di alta velocità e di modo migliore di gestione di dati, della libreria NIO, a noi, interessa che

soddisfa le richieste. Nella prima parte del paragrafo parlavamo di file come oggetto, contenitore di dati, ora NIO lo tratta come un documento sottoposto al procedimento d'inserimento nel database. Avendo da fare con dati di grande quantità che sono spostati da un dispositivo di storage ad un altro.

Codice del nostro programma che utilizza le classi del pacchetto java.nio:

```
public void copyFileUsingFileChannels(FileText source)
throws IOException {
   FileInputStream fromFileStream = new
   FileInputStream(new File(source.getDir_sorgente()));

FileOutputStream toFileStream =

   new FileOutputStream((new
   File(source.getDir_destinazione() +"\\"+
   source.getNomeFile())));

FileChannel fromChannel = fromFileStream.getChannel();

FileChannel toChannel = toFileStream.getChannel();

try {

fromChannel.transferTo(0, fromChannel.size(), toChannel);
} finally {
if (fromChannel != null)
  fromChannel.close();
if (toChannel != null)
  toChannel.close();
fromFileStream.close();
  toFileStream.close();
          }
     }
```

PARTE II. PROGETTO ED IMPLEMENTAZIONE

Capitolo 3. Creazione archivio dati

Nel seguente capitolo si discutono i dettagli che hanno preceduto alla realizzazione vera e propria del progetto partendo dalle richieste iniziali del committente, procedendo con uno studio di fattibilità utilizzato per verificare se tutte le richieste potevano essere soddisfatte, e in che modo.

Ci occuperemo della creazione di un database. Tratteremo il progetto concettuale e poi lo schema entità – relazione. In seguito sarà tradotto lo schema ER per generare le tabelle e di conseguenza renderemo reale il nostro database, includendo le varie operazioni su di esso.

Probabilmente, durante la descrizione del tema saranno porzioni di testo con contenuto ambiguo, specialmente la popolazione del database con i dati. Per questo faremo riferimento al codice Java, argomento trattato nel capitolo quattro.

3.1 Specifica Dei Requisiti

Presentiamo le richieste del progetto, descriviamo in linee generale di come si presenta la situazione.

Individuazione file da migrare - verrà creato un programma in Java che effettua la ricerca di tutti i file presenti in un "path" indicato dal utente, "path " rappresenta un indirizzo di cartella e/o directory. Come risultato si deve ottenere la lista completa di tutti i file presenti nella directory.

Per la scrematura dei duplicati - si ha come obiettivo di creare un software in Java che appoggiandosi ad un DBMS OpenSource, sarà usato per trovare eventuali duplicati dei file.

Estrazione metadati – useremo un framework per estrarre metadati su alcuni tipologie di documenti. Prima, si effettua una selezione del tipo di file da cui si vuole estrarre le informazioni, poi vengono inseriti in una delle tabelle del database anticipatamente creato.

Per prima cosa creiamo il programma in Java che realizzasse la ricerca dei file, con i dati ottenuti proseguiamo con la creazione del DB ed eventualmente inserimento dei dati al suo interno. *L'Insert* dei dati sarà possibile solo dopo aver creato il programma in Java che compie la connessione con il DB e realizza la popolazione dell'archivio. L'ultimo paragrafo include raffinamenti al database, si trova la richiede aggiungere altre nuove tabelle alla basi di dati.

3.2 Progettazione database

Per iniziare la creazione di un database la cosa più importante è la progettazione. In questa fase si stabilisce la forma che il database avrà, quale sarà l'oggetto e lo scopo.

Dopo aver verificato se la natura dei dati possa appartenere ad una delle tipologie di database si procede con l'identificazione di tutti gli elementi necessari per la realizzazione dell'archivio. Si valuta l'economicità dei tipi di dati adottati e si stabilisce il miglior modo di rappresentarli. Quando tutte le problematiche che si presentano dalla natura stessa sono risolte si può passare alla creazione del database. Il DBMS MySQL si presta adatto alle esigenze funzionali ed agli scopi proposti [2, 12].

Il progetto concettuale è il punto di partenza di ogni database, né determina il successo. Sarà comodo rivedere l'intero progetto riprendendolo dalle sue fasi iniziali se durante la sua realizzazione si dovessero rilevare degli elementi nuovi che andrebbero ad arricchire o che cambiare alcune delle impostazioni e decisioni in precedenza adottate.

Le fasi della progettazione si distinguono per il grado di astrazione che le caratterizzano. Si passa dalla progettazione del modello concettuale (scelta del DBMS e definizione dello schema Entità-Relazione), alla fase di produzione del modello logico (definizione di attributi e campi di collegamento (chiavi primarie e secondarie)) per giungere infine alla realizzazione fisica del database.

La fase di creazione dipende esclusivamente dalle seguenti tappe:

- Si definisce un modello concettuale;
- Si definisce il diagramma Entità/Relazione (E/R);
- Si definisce il modello logico

Il modello logico include:

- definizione di attributi
- identificazione chiavi primarie e secondarie
- relazione molti a molti

3.2.1. Il modello concettuale

Il modello concettuale fornisce una rappresentazione semplificata della realtà. Il grado di semplificazione dipende dalle scelte fatte per chi sviluppa il modello. Si tratta quindi di definire un insieme di dati presenti in natura e che rappresentano la natura stessa delle informazioni che si vogliono archiviare. I dati poi devono essere individuati in modo tale che rispondano allo scopo per il quale si crea il modello. Non esistono pertanto delle regole prefissate per l'individuazione dei dati e per la loro selezione, ad uno scopo diverso corrisponde un insieme di dati differenti e quindi un modello concettuale distinto.

CONTENUTO DEL PROGETTO

L'azienda specializzata in informatica intende creare un sistema informativo secondo le seguenti specifiche. Dai vari dispositivi di storage di cui dispone, si vuole gestire tutti i documenti in essi (dischi e server condivisi) contenuti. Lo scopo è che dopo la scansione di una certa directory i documenti trovati vengono inseriti all'interno di un database. Il meccanismo prevede che alcune categorie di file debbano essere copiate nelle cartelle prescelte dall'utente. Il procedimento implica di coordinare ogni spostamento di file. Ogni atto di spostamento DA una directory A una directory viene identificato con un numero intero che indica lo stato del file; per esempio 0 significa che il file è stato appena inserito nel DB; 1 significa che il file è stato copiato in una directory destinazione e 3 se ha subito un altro comportamento. Inoltre i dispositivi di storage saranno identificati come directory sorgente e le cartelle sono denominate come directory destinazione. Un file è descritto da un codice univoco, nome file, dimensione, data creazione, data modifica, data ultimo accesso, estensione, daTrasferire, isRegular, isSymbolic, isOther.

Ogni tipo directory viene identificata dal suo nome.

Viene chiesto di sviluppare un database in grado di archiviare tutte le informazioni e dal quale si possa ottenere:

- lista dei documenti con estensioni (formato del file) uguali;

- lista dei documenti di una certa dimensione raggruppati per nome;

- lista dei documenti che hanno la stessa data di creazione e/o non superi una certa data di modifica;

- elenco di certi documenti che necessita essere spostato in una cartella diversa;

- lista di documenti raggruppati per un certo anno;

3.2.2 Il Modello Entità-Relazione (ER)

Dopo aver formalizzato la semplificazione della realtà da studiare per realizzare lo scopo di archiviare un insieme di documenti, si deve decidere a quale tipo di database può appartenere. Nel nostro caso proposto la tipologia che meglio permette di gestire le informazioni è il modello relazionale.

Essendo questo il modello molto legato alla teoria degli insiemi si procederà all'identificazione delle entità o insiemi di dati, alla realizzazione del database. Le entità si definiscono come oggetti che possono essere identificati in modo chiaro in un determinato contesto, le cui proprietà chiamate attributi sono oggetto di archiviazione.

In questa fase si prevede l'individuazione d'insiemi di dati (oggetti) che caratterizzano l'archivio, non si occupa invece di analizzare le singole caratteristiche.

Dopo aver identificato gli insiemi di dati da archiviare, si definiscono i legami esistenti tra essi: le relazioni. Le relazioni rappresentano i legami o dipendenze o associazioni d'interesse informativo che esistono tra le entità che compongono il database. Tali legami sono definiti in base alla natura stessa delle entità e saranno pertanto definiti tramite lo studio stesso dei dati.
Le relazioni vengono classificate secondo la loro cardinalità in:

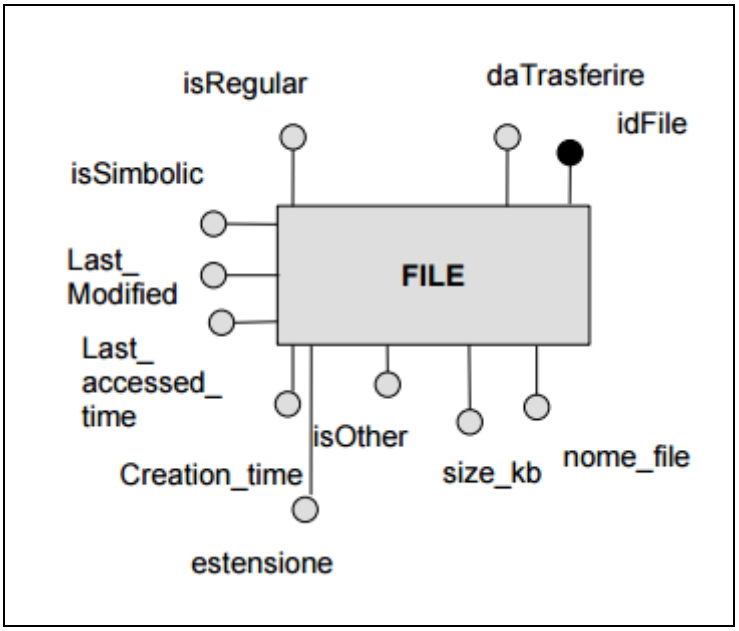
1) Relazioni uno ad uno (1: 1);

2) Relazioni uno a molti (1: M);

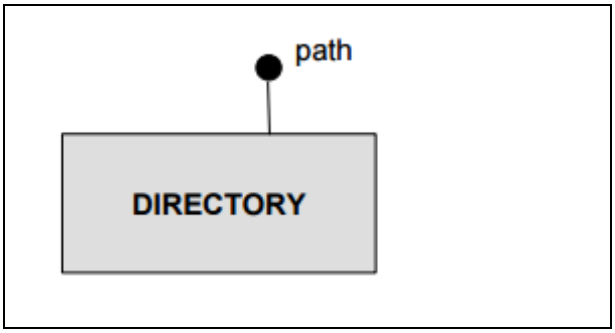
3) Relazioni molti a molti (M: M);

IDENTIFICAZIONE ENTITÀ E RELAZIONI

Dal testo con i requisiti si nota che l'oggetto *file*, invita molta attenzione per la maggior parte del contenuto. Viene facile a denominarlo come entità rappresentativa per il nostro compito. Gli attributi che descrivono l'entità sono le proprietà dell'oggetto e li raffiguriamo come segue:



Un dispositivo di storage o una cartella, termine che verrà usato per la maggior parte del capitolo, è una directory che ha un nome, lo stesso nome rappresenta il suo percorso nelle risorse del computer, detto anche **path.** Un'altra entità identificata è la *DIRECTORY*, viene semplice ad illustrare il suo schema poiché definisce un solo attributo, "*path*" che rappresenta il suo identificativo univoco.



Per quanta riguarda directory sorgente e directory destinazione rappresentarli con l'entità DIRECTORY, appartenente alla categoria e definite dallo stesso attributo

identificativo *path*. Per rappresentarli nel nostro modello relazionale si usano le associazioni, nel modo che per indicare che un file viene copiato da una directory in una cartella possiamo effettuare uno schema come segue:

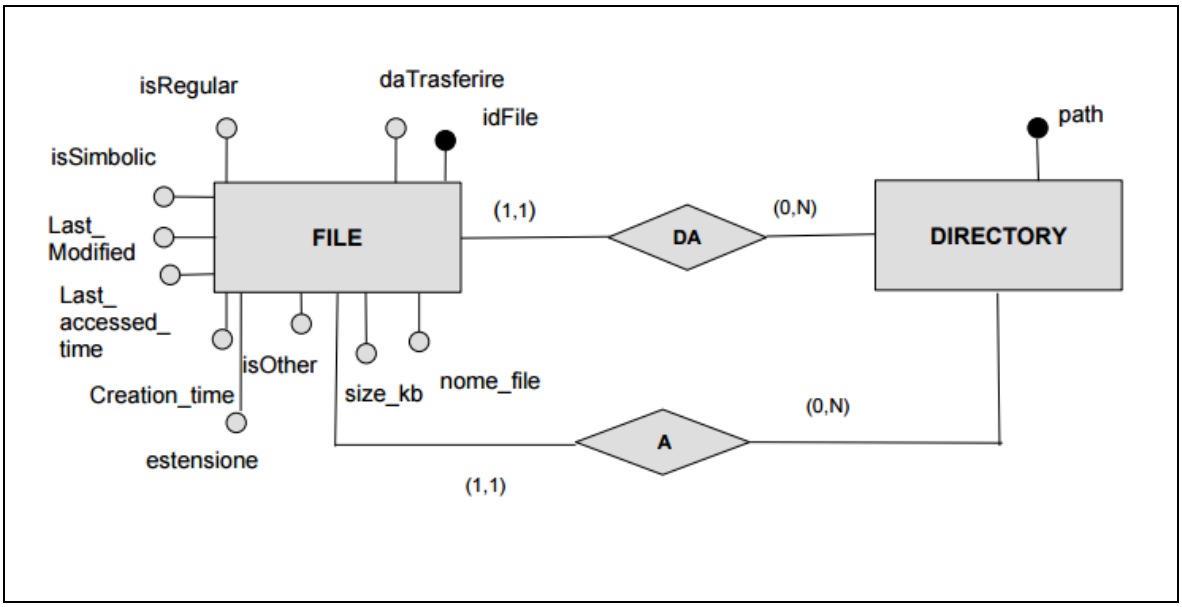


Figura 4 Schema E-R.

Per quanto riguarda la definizione delle cardinalità occorre considerare che:

- Un file viene copiato da una directory una sola volta (1,1), ove un file viene spostato in una cartella sempre una volta sola (1,1).

- Una directory sia sorgente che destinazione puo contenere fino a N file per essere copiati, e possono essere sottoposti all'atto dello spostamento più volte (0, N), cardinalità che riguarda le due parti dell'associazione *DA* e *A*.

La figura 4 rappresenta lo schema E-R prodotta per il nostro progetto. In essa abbiamo inquadrato i requisiti necessari previsti nella prima parte del contenuto del compito. Consecutivamente procediamo con la traduzione logica del modello.

3.2.3 Modello logico

Costruito lo schema concettuale (modello E-R) occorre tradurre in uno schema logico ad esso equivalente, nel nostro caso useremo il modello relazionale. Questo modello permette di definire l'organizzazione dei dati, le modalità di accesso e le viste, cioè gli schemi esterni. Nel caso di un database di tipo relazionale, il modello logico si

occuperà di stabilire le chiavi interne, chiavi primarie(PK) e secondarie (FK), della traduzione delle relazioni emerse dalla diagramma E-R. Il modello logico costituisce l'implementazione dello schema E-R e della fase finale per la creazione del database.

Per costruzione dello schema logico relazionale dobbiamo seguire le seguente fasi:

1. Eliminazione della gerarchia ISA

2. Trasformazione degli attributi e identificatori

3. Traduzione di entità e associazioni in schemi di relazioni

4. Associazioni unarie

Nella traduzione logica non abbiamo incluso tutte le fasi, svilupperemo con spiegazioni solo le regole utilizzate.

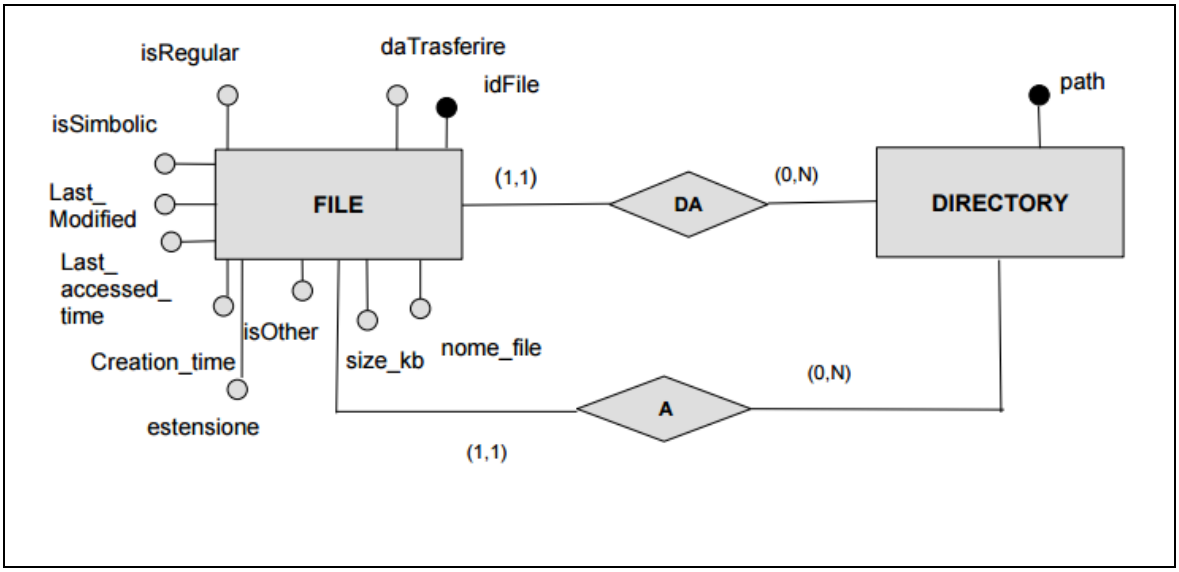
3. Traduzione di entità e associazioni in schemi di relazioni

In questa fase ogni entità è tradotta con una relazione con gli stessi attributi.

- a chiave primaria della relazione è quella dell'entità attributi.

- ogni altro identificatore dell'entità è chiave alternativa della relazione.

Analizziamo la seguente associazione:



Si effettua la traduzione con due relazioni. (Considerando la relazione *DA* per *FILE* e *DIRECTORY* per prima).

L'associazione può essere compattata nell'entità che partecipa con molteplicità unitaria, includendo in *FILE* la chiave primaria dell'entità *DIRECTORY* come foreign key:

FILE (**idFile**, **directory_sorgente**, nome_file, size_kb, Creation_time, Last_Modified, isRegular, isSymbolic, isOther, Last_Accessed_Time, estensione, daTrasferire)

FK: directory_sorgente **REFERENCE** DIRECTORY

Lo stesso procedimento vale per l'associazione unaria *A* che implica le entità *FILE* e *DIRECTORY*.

FILE (**idFile**, **directory_destinazione**, nome_file, size_kb, Creation_time, Last_Modified, isRegular, isSymbolic, isOther, Last_Accessed_Time, estensione, daTrasferire)

FK: directory_destinazione **REFERENCE** DIRECTORY

La traduzione completa dell'entità ***FILE*** è:

FILE (**idFile**, **directory_sorgente**, **directory_destinazione**, nome_file, size_kb, Creation_time, Last_Modified, isRegular, isSymbolic, isOther, Last_Accessed_Time, estensione, daTrasferire)

FK: directory_sorgente **REFERENCE** DIRECTORY

FK: directory_destinazione **REFERENCE** DIRECTORY

3.3 Raffinamenti al database

Inizialmente si prevedevano la costruzione e creazione del progetto che consiste da una sola tabella che contiene i dati necessari previsti fin'ora. Durante lo svolgimento del progetto è comparsa la necessità di effettuare aggiornamenti che implicano direttamente la basi di dati esistente, intervenuti nella seconda parte del tirocinio, si chiede di modificare la composizione del database aggiungendo altre tabelle. Realizziamo i raffinamenti al presente archivio inserendo altre nuove tabelle.

Il punto 3, presente nei requisiti, nella creazione del software, prevede delle estrazioni supplementari di dati sui documenti, chiamati metadati. I metadati sono generati dal codice Java, memorizzandoli nelle classi opportune, invece per l'implementazione nel MySQL dobbiamo ricorrere alle operazioni supportate del DBMS. Si vogliono rappresentare informazioni relative alla gestione dei metadati per certi tipi di file. Essi (i metadati) vengono estratti diversamente per ogni categorie di documento. Le categorie che ci interessano di ottenere i metadati sono tutti i file di tipo testuale, pdf ed Excel. Ogni file dispone di caratteristiche specifiche alla sua tipologia.

Si chiede che un file sia classificato per categorie, ogni documento è copiato in una specifica directory_destinazione (specifica directory- significa che non esiste una cartella prefissata prima del procedimento, si decide al momento come gestire la cartella). Non esistono vincoli per la quantità dei file che una cartella puo contenere, ne sul numero di transazioni da effettuare.

■ Un documento Excel è identificato dalle seguenti caratteristiche: doc_type, author, title, producer, content_type creation_date, last_modified e creator.

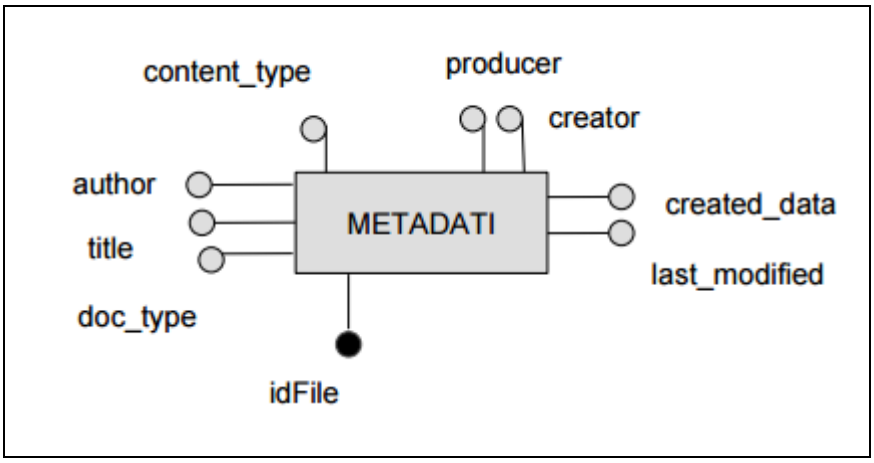
■ Un documento testuale viene descritto dalle successive proprietà: doc_type, author, title, producer, content type, creation_date, last modified, pages_nr, last author.

■ E un documento pdf viene caratterizzato dalle consecutive proprietà: doc_type, author, title, producer, content type, creation_date, last modified, pages_nr, key_words.

3.3.1 Il modello concettuale

Essendo riportati solo questi requisiti per il nostro compito, intendiamo modellare i metadati dei file utilizzando le gerarchie di generalizzazione. L'entità *METADATI* rappresenta la classe padre ed incorpora le proprietà comune per le istanze figlie, *FilePdf,FileXlsx, FileOffice*. Essa invece a sua volta crea una relazione con associazione con l'entità *File.* La classe *File* rappresenta l'astrazione di un documento ed inizialmente è stato progettata per uno scopo generico (che ha il ruolo manipolare un'insieme di file generici, ove di vari tipi) rispetto a *METADATI* che rappresenta uno specifico file.

Raffigurata in uno schema, l'entità *METADATI* si dimostra come segue:



Abbiamo creato un'entità che accoglie tutti gli attributi simili delle altre istanze, in questa maniera riusciamo a mantenere una certa consistenza dei dati. La copertura della gerarchia è ti tipo totale ed esclusiva in quanto l'entità *METADATI* genera l'appartenenza delle categorie (FileTxt, FileXlsx, FilePdf) ed è direttamente associata ad essa. Occorre modellare quindi le entità figlie: *FileOffice*(Testuale) [7], *FileXlsx* e *FilePdf*.

[7] FileTestuale- si intende documenti come file.odt, file.doc, e file.txt.

La gerarchia prodotta viene esemplificata, in maniera completa, consecutivamente:

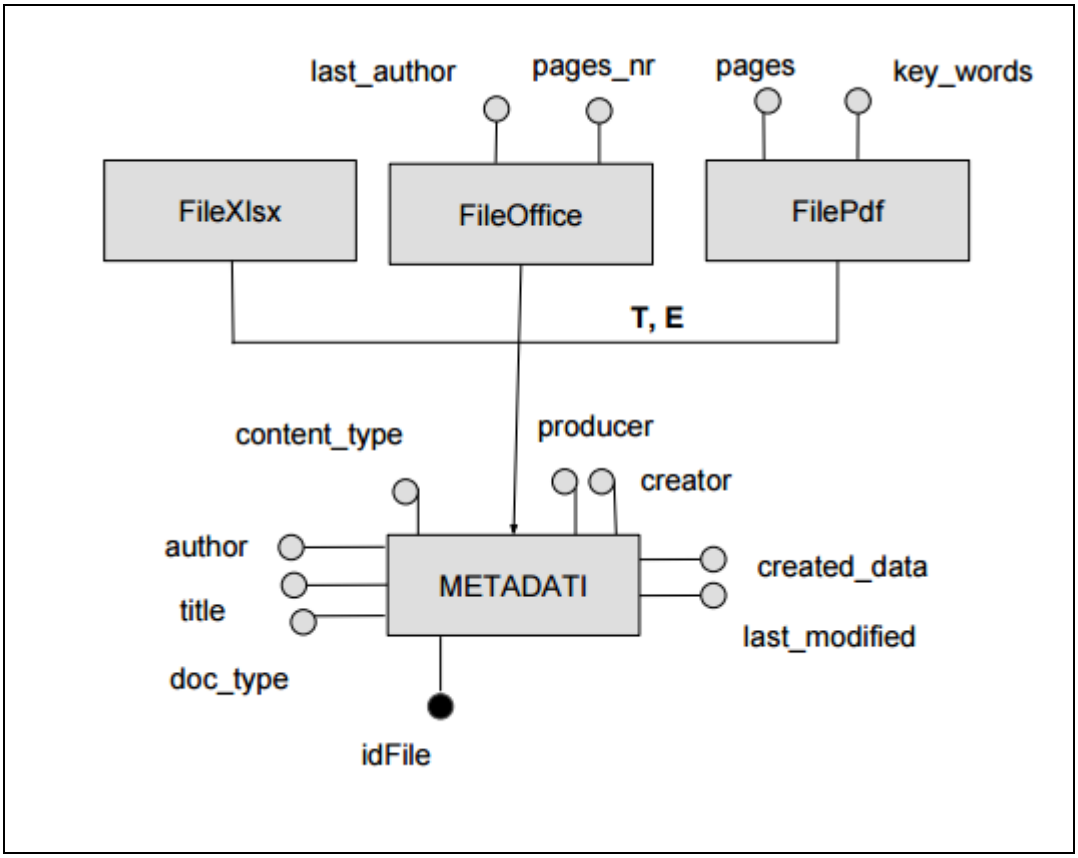


Figura 5 Gerarchia Metadati.

3.3.2 Traduzione modello concettuale, il progetto logico.

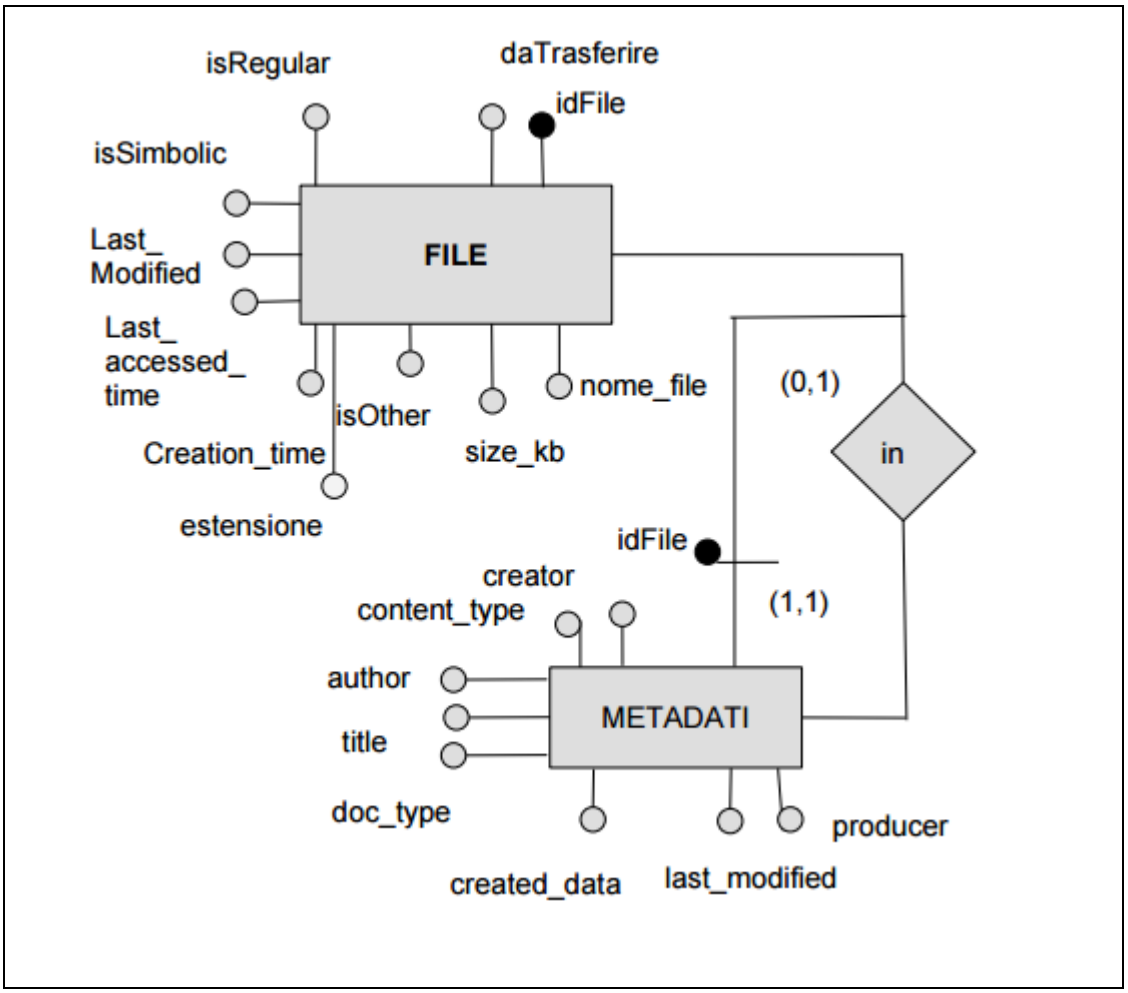
La traduzione standard del modello concettuale prevede: ogni entità è tradotta con una relazione con gli stessi attributi, cioè a chiavi primarie della relazione è quella dell'entità stessa; ogni altro identificatore dell'entità è chiave alternativa della relazione;

Tutte e due le affermazioni sono valide per noi. Poiché le entità File e METADATI sono rappresentati tramite l'associazione binaria che include la presenza di un identificatore esterno. Motivo per il quale abbiamo ricorso ad una tale corrispondenza viene spiegato dal fatto che la gerarchia di generalizzazione, ossia le entità *FileTxt, FileXlsx, FilePdf*, rappresentano una estensione dello stesso *FILE*, della prima classe creata. Tuttavia, le entità non sono altro che una specifica rappresentazione di un documento, è un record esistente nella tabella *FILE*, viene esteso in un'altra tabella per

compiere il requisito che prevede l'estrazione dei metadati per alcuni tipi di dati. Le tabelle sono caratterizzate dai metadati, devono essere però identificate dallo stesso numero identificativo della classe *FILE*.

Nel paragrafo precedente non abbiamo incluso il modello concettuale tra le entità *FILE* e *METADATI*, motivo per quale abbiamo lasciato spazio per la parte corrente, è per non ripeterne i concetti.

La rappresentazione schematica dell'entità *FILE* con l'entità *METADATI* si presenta in seguito:



Come cardinalità tra le relazioni possiamo notare che l'associazione è parziale per l'entità *METADATI* e obbligatoria per *FILE*, risulta che la chiave deve essere `idFile` dell'entità *METADATI*, ed è `UNIQUE`. La traduzione viene fatta nel modo seguente:

`METADATI (`**`idFile`**`, content_type, author, title, doc_type, created_data, last_modifed)`

FK: idFile **REFERENCES** FILE

idFile **UNIQUE**

Traduzione delle gerarchie

Il modello relazionale non rappresenta le gerarchie, esse vanno sostituite da entità e associazioni.

Le possibilità per tradurre le gerarchie sono tre:

1) mantenimento delle entità con associazioni;

2) collasso verso l'alto

3) collasso verso il basso

Le trasformazioni delle gerarchie possono modificare le cardinalità di attributi ed associazioni. L'applicabilità e la convenienza delle soluzioni dipendono dalle proprietà di copertura della gerarchia e dalle operazioni previste.

Per tradurre la gerarchia di generalizzazione si applica il collasso verso il basso, cioè tutti gli attributi dell'entità padre vengono trasferiti sulle entità figlie eliminando la classe. La traduzione prevede che un'associazione dell'entità padre si moltiplica, tante volte quante sono le entità figlie.

Questo tipo di traduzione si può fare se la copertura della gerarchia è completa ed esclusiva, e per noi lo è, abbiamo indicato prima.

Otteniamo che l'entità figlie incorporano ai presenti attributi, anche gli attributi dell'entità padre. Vediamo la generazione delle tabelle come si presenta:

FileOffice(**idfileoffice**, content_type, author, title, doc_type, created_data, last_modifed, pages_nr, last_author)

FK: idfileoffice **REFERENCES** FILE

FileXlsx (**idfilexlsx**, content_type, author, title, doc_type, created_data, last_modifed,)

FK: idfilexlsx **REFERENCES** FILE

FilePdf (**idfilepdf**, content_type, author, title, doc_type, created_data, last_modifed, pages_nr, key_words)

FK: idfilepdf **REFERENCES** FILE

Quello che abbiamo realizzato sono le tre tabelle, aggiunte al database. Concentrandoci sull'aspetto di risorse di dati otteniamo un'estensione della tabella principale *File* in tabelle secondarie con più la classificazione a seconda al formato del documento (.pdf, .doc, .xlsx, .docs, .odt).

3.3.3 Schema E-R completa

Questa fase prevede la realizzazione del database sul DBMS che rispecchia lo schema progettato con particolare riguardo alla realizzazione dei vincoli aggiuntivi e dei meccanismi di aggiornamento dei dati derivati.

Gli oggetti del database da creare sono stati raggruppati in schemi. I quattro schemi base sono quelli che derivano dalla progettazione logica.

FILE per i documenti di tutti i formati

FilePdf per i file di tipo pdf

FileOffice per i file del formato .doc, .docx, .odt

FileXlsx per i file Excel

Infine dimostriamo il risultato generato dopo il raffinamento del DB, lo schema E-R completa del nostro progetto:

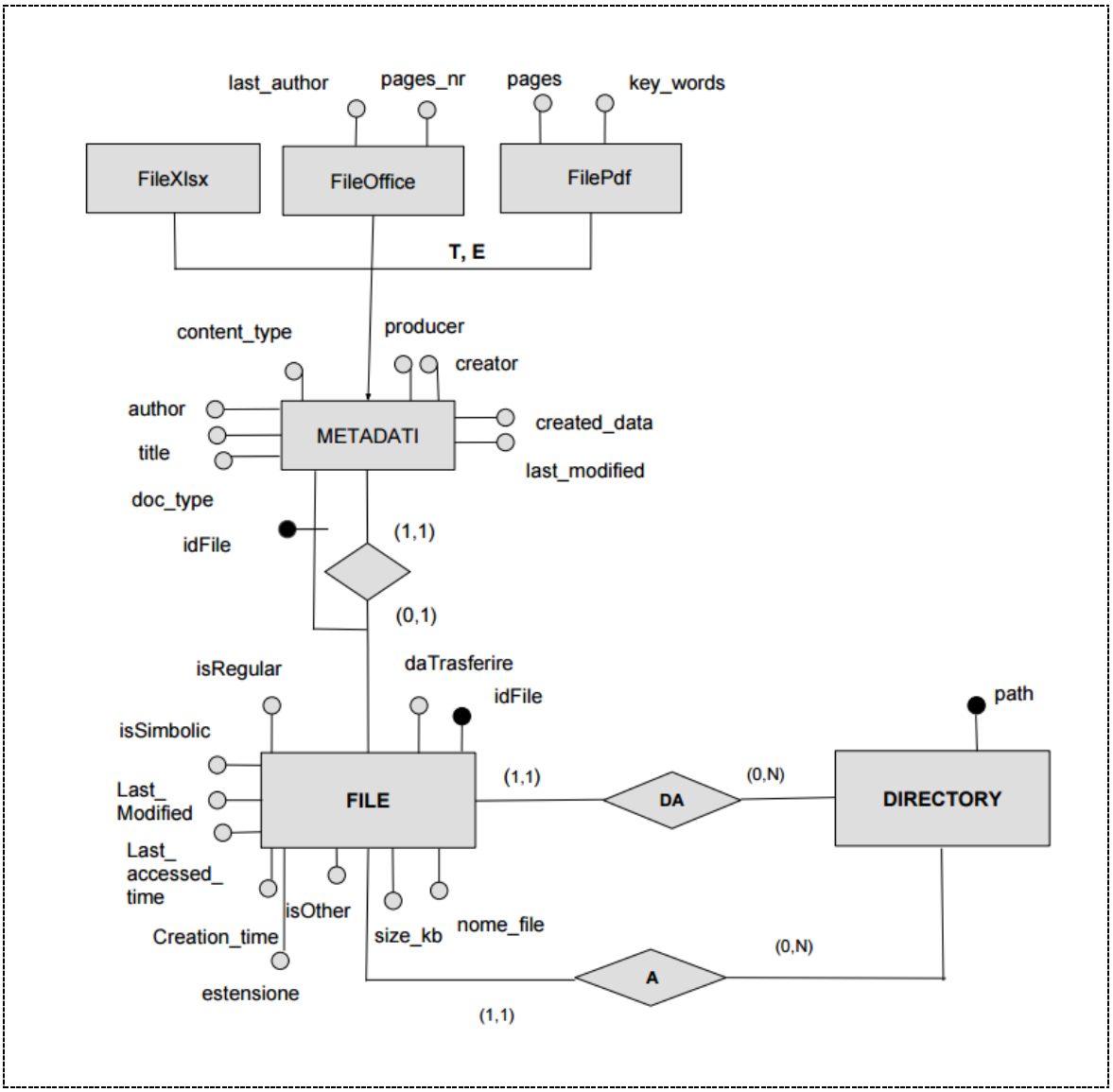


Figura 6 Schema E-R completa.

Il database è costituito dalle seguente tabelle:

FILE (**idFile**, **directory_sorgente**, **__directory_destinazione**, nome_file, size_kb, Creation_time, Last_Modified, isRegular, isSymbolic, isOther, Last_Accessed_Time, estensione, daTrasferire)

FileOffice (**idfileoffice**, content_type, author, title, doc_type, created_data, last_modifed, pages_nr, last_author)

FK: idfileoffice **REFERENCES** FILE

FileXlsx (**idfilexlsx**, content_type, author, title, doc_type, created_data, last_modifed)

FK: idfilexlsx **REFERENCES** FILE

FilePdf (**idfilepdf**, content_type, author, title, doc_type, created_data, last_modifed, pages_nr, key_words)

FK: idfilepdf **REFERENCES** FILE

La creazione delle tabelle provenienti dallo schema relazionale prodotto consiste nella stesura degli statement, *create table*. Questo porta a dover scegliere il tipo di dato dei vari attributi. Tale scelta è stata fatta nel secondo capitolo.

Detto cioè, si prosegue con la creazione e l'implementazione della nostra base di dati nel server con gli schemi e le tabelle prodotti.

3.4 Implementazione nel DBMS

In questo sotto paragrafo tratteremo l'implementazione, la connessione e le varie operazioni da effettuare sulla nostra basi di dati, manipolata in MySQLWorkbench. *AlfImporter*, il nome del nostro database relazionale viene costruito con le tabelle *FILE*, *FilePdf*, *FileXlsx*, *FileOffice*. La struttura interna del database immagazzina i dati nelle tabelle e permette intervenire su di essa con operazioni specifiche del linguaggio SQL.

3.4.1 Connessione al server

La connessione al server del database avviene tramite i soliti dati necessari per rendere privato un collegamento. Il nome dell'utente e la parola sono i credenziali per ogni utente. Impone che ogni utente venga riconosciuto individualmente da SQL Server. La seguente riga indica che l'utente con il nome "XXXX" e parola "'xxxx" ha tutti i diritti per eseguire ogni tipo d'operazione sul database *alfimporter*.

GRANT ALL PRIVILEGES ON alfimporter.* to XXXX IDENTIFIED BY 'xxxx';

3.4.2 Operazioni su Database

Avendo il database popolato con dei dati, andremo a recuperarli effettuando interrogazioni scritti nel linguaggio SQL, cioè eseguire dei query. Le varie possibilità di operare su un database sono le esecuzioni delle query. Si può interrogare al minimo una tabella ed estrarre dati di quanti campi si desidera.

Esistono diversi tipi di query:

Query di selezione: è possibile estrarre i dati (*select*) da una o più tabelle e visualizzarli in una nuova tabella.

Query di accodamento: consente di aggiungere (*insert*) a tabelle già esistenti un gruppo di record in base a dei criteri specifici.

Query di aggiornamento: consente di modificare (*update*) il valore di uno o più campi in corrispondenza di un intervallo selezionato di records esistenti.

Query di eliminazione: consente di cancellare (*delete*) uno o più record indipendentemente dai criteri inseriti nella creazione della query.

Query a campi incrociati: permette di "incrociare" i campi di più tabelle in modo tale da ottenere una matrice in cui le righe corrispondono normalmente a campi di tipo descrittivo e riepilogativo, mentre le colonne corrispondono a totali o conteggi.

Vengono chiamate anche *query pivot*, poiché i dati che risultano alla fine della creazione sono simili alle tabelle pivot di un foglio di calcolo.

Query di *creazione tabella*: se lo si desidera, le righe prodotte da una query possono anche alimentare i record di una nuova tabella mediante una query di creazione tabella, facendo sempre riferimento alla scheda Struttura, gruppo Tipo di query

Esistono inoltre, le query *annidate* dove il filtro della query è dato da un'altra query.

Ci sono anche le *query di raggruppamento* dove è possibile raggruppare righe in base ad uno o più campi. In questo caso, ogni gruppo è costituito da righe aventi dati uguali nel campo indicato (detto appunto campo raggruppato) ed occuperà una sola riga

dell'elenco prodotto dalla query. Le principali funzioni native di accesso da applicare ad ogni raggruppamento sono quelle di somma, conteggio, media, minimo e massimo per cercare di capire un algoritmo di selezione dati in base a qualche algoritmo.

Per le interrogazioni d'inserimento si effettua in maniera denotata sotto, usando java per inserire i dati la semantica non cambia.

QUERY DI INSERIMENTO

Inserisci valori nella tabella XX per le colonne specificate nelle parentesi.
`INSERT INTO table_name( column1, column2....columnN)` `VALUES ( value1, value2....valueN);`

Per la query d'inserimento dei dati in una tabella con il linguaggio SQL è illustrata soltanto la semantica perché l'operazione effettivamente viene effettuata dalla classe *`FileTextDao`*, nel seguente modo:

```
public int insert(FileText filetext) {
  String sql = "insert into file( nome_file, creation_time,
    last_modified, size_kb, daTrasferire, directory_sorgente,
    directory_destinazione, is_regular,is_symbolic, is_other,
    last_accessed_time, file_extention) values ( ?, ?, ?, ?,
    ?, ?, ?, ?, ?, ?, ?, ?)";
jdbcTemplate.update(sql, new Object[] {
    filetext.getNomeFile(),
new Timestamp(filetext.getDataCreazione().toMillis()),
new Timestamp(filetext.getultimaModifica().toMillis()),
    filetext.getSizeFile(),
filetext.getDaTrasferire(),filetext.getDir_sorgente(),filetext
    .getDir_destinazione(), filetext.isRegular(),
    filetext.isSymbolic(), filetext.isOther(),
new Timestamp(filetext.getUltimoAcceso().toMillis()),
```

```
        filetext.getFileExtention()});
            return 0;
        }
```

Figura 7 visualizza la tabella *FILE* con i dati inseriti al suo interno. Il risultato dell'inserimento viene dimostrato eseguendo la query "SELECT * FROM file", abbiamo la risposta di aver presente nella tabella 135073 documenti. Tuttavia, il tempo per l'esecuzione è ottimo (circa 6 decimi di secondi), operazione avvenuta con successo.

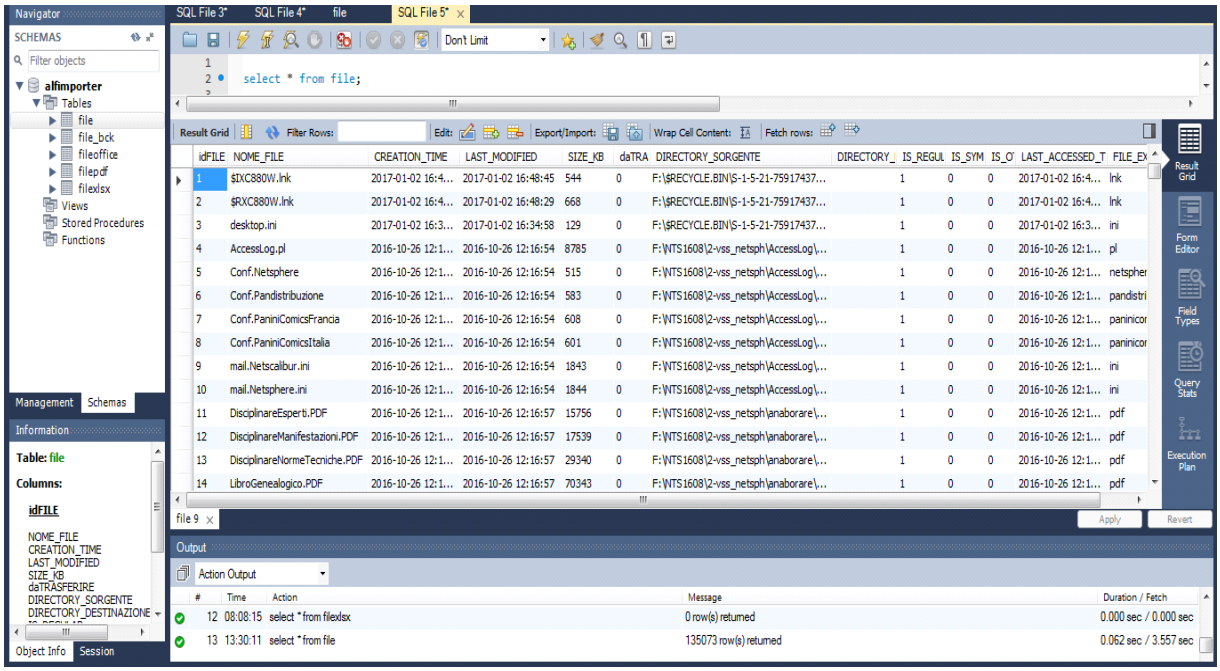


Figura 7 Tabella File.

QUERY DI SELEZIONE

Seleziona tutti i file distinti raggruppati per il nome e la dimensione.

```
SELECT *, count(*) FROM file
GROUP BY nome_file, size_kb
HAVING count(*)>1;
```

```
SELECT *, count(*)
 FROM file
 group by nome_file, size_kb
 having count(*)>1;
```

idFILE	NOME_FILE	CREATION_TIME	LAST_MODIFIED	SIZE_KB	daTRASFERIRE	DIRECTORY_SORGENTE	DIRECTORY_DESTINAZIONE	IS_REGULAR	IS_SYMBOLIC
242030	!ReadMe.txt	2016-11-02 08:17:15	2009-09-11 11:31:57	753	0	F:\NTS1608\1-SVN\Progetti\Manutencoop\XXX_...		1	0
70519	-.-	2016-11-02 08:36:30	2008-07-04 16:39:37	3	0	F:\NTS1608\1-SVN\Progetti\Manutencoop\XXX_...		1	0
84345	.actionScriptProperties	2016-11-02 11:41:52	2010-05-10 11:27:03	836	0	F:\NTS1608\1-SVN\Progetti\NewMedia\LegaCal...		1	0
84336	.actionScriptProperties	2016-11-02 11:41:50	2010-05-06 20:27:06	842	0	F:\NTS1608\1-SVN\Progetti\NewMedia\LegaCal...		1	0
14462	.checkstyle	2016-11-02 09:40:31	2016-03-30 15:38:20	313	0	F:\NTS1608\1-SVN\Progetti\Alfresco\Framewor...		1	0
14606	.classpath	2016-11-02 08:06:48	2014-03-21 12:23:03	382	0	F:\NTS1608\1-SVN\Progetti\Atlas\Alfresco.4.2\...		1	0
14533	.classpath	2016-11-02 08:07:00	2012-12-10 10:03:46	388	0	F:\NTS1608\1-SVN\Progetti\Atlas\Alfresco\Grup...		1	0
14377	.classpath	2016-11-02 09:34:09	2014-11-19 14:39:37	411	0	F:\NTS1608\1-SVN\Progetti\Admenta\Sorgenti\...		1	0
14767	.classpath	2016-11-02 08:05:49	2012-11-26 14:45:40	443	0	F:\NTS1608\1-SVN\Progetti\BPER\Alfresco\Itali...		1	0

Result 8

Output

#	Time	Action	Message	Duration / Fetch
6	09:53:46	SELECT * FROM file	0 row(s) returned	0.016 sec / 0.000 sec
7	10:09:03	SELECT * FROM file	0 row(s) returned	0.000 sec / 0.000 sec
8	10:20:47	SELECT * FROM file	251934 row(s) returned	0.000 sec / 0.811 sec
9	11:00:22	SELECT count(*) FROM file group by nome_file, size_kb having count(*)>1	20813 row(s) returned	2.371 sec / 0.031 sec
10	11:00:35	SELECT *, count(*) FROM file group by nome_file, size_kb having count(*)>1	20813 row(s) returned	166.765 sec / 59.452 sec

Seleziona tutti i file che hanno l'estensione 'pdf';

```
SELECT * FROM file

WHERE file_extention="pdf";
```

SQL File 3* SQL File 4*

```
select * from file
where FILE_EXTENTION="pdf";
```

CREATION_TIME	LAST_MODIFIED	SIZE_KB	daTRAS	DIRECTORY_SORGENTE	DIRECTORY_DESTINAZIC	IS_REGULAF	IS_SYM	IS_OTHE	LAST_ACCESSE	FILE_EXTENTION
2016-06-14 21:52:42	2016-06-14 21:52:51	820782	0	C:\Users\utente\Documents\BD\14.AlgebraRel...		1	0	0	2016-06-14 ...	pdf
2016-02-29 15:16:02	2016-02-29 15:16:02	11977821	0	C:\Users\utente\Documents\BD\A1-introduzio...		1	0	0	2016-02-29 ...	pdf
2016-02-29 15:06:21	2016-02-29 15:06:22	8538435	0	C:\Users\utente\Documents\BD\A2-tecnologia...		1	0	0	2016-02-29 ...	pdf
2016-03-09 18:50:39	2016-03-09 18:50:39	849021	0	C:\Users\utente\Documents\BD\B1-introduzion...		1	0	0	2016-03-09 ...	pdf
2016-03-09 18:50:25	2016-03-09 18:50:25	679632	0	C:\Users\utente\Documents\BD\B2-ER.pdf		1	0	0	2016-03-09 ...	pdf
2016-03-09 18:50:56	2016-03-09 18:50:56	673078	0	C:\Users\utente\Documents\BD\B3-proprietà.pdf		1	0	0	2016-03-09 ...	pdf
2016-03-09 18:51:19	2016-03-09 18:51:19	691657	0	C:\Users\utente\Documents\BD\B4-associa1.pdf		1	0	0	2016-03-09 ...	pdf
2016-03-09 18:51:35	2016-03-09 18:51:35	693297	0	C:\Users\utente\Documents\BD\B5-associa2.pdf		1	0	0	2016-03-09 ...	pdf
2016-03-09 18:51:52	2016-03-09 18:51:52	700645	0	C:\Users\utente\Documents\BD\B6-identifica.pdf		1	0	0	2016-03-09 ...	pdf
2016-03-09 18:52:35	2016-03-09 18:52:35	696081	0	C:\Users\utente\Documents\BD\B7-ISA.pdf		1	0	0	2016-03-09 ...	pdf
2016-05-03 14:31:22	2016-05-03 14:31:22	66868	0	C:\Users\utente\Documents\BD\BD160419R.pdf		1	0	0	2016-05-03 ...	pdf
2016-07-19 20:10:33	2016-07-19 20:10:41	194295	0	C:\Users\utente\Documents\BD\BD160707R.pdf		1	0	0	2016-07-19 ...	pdf

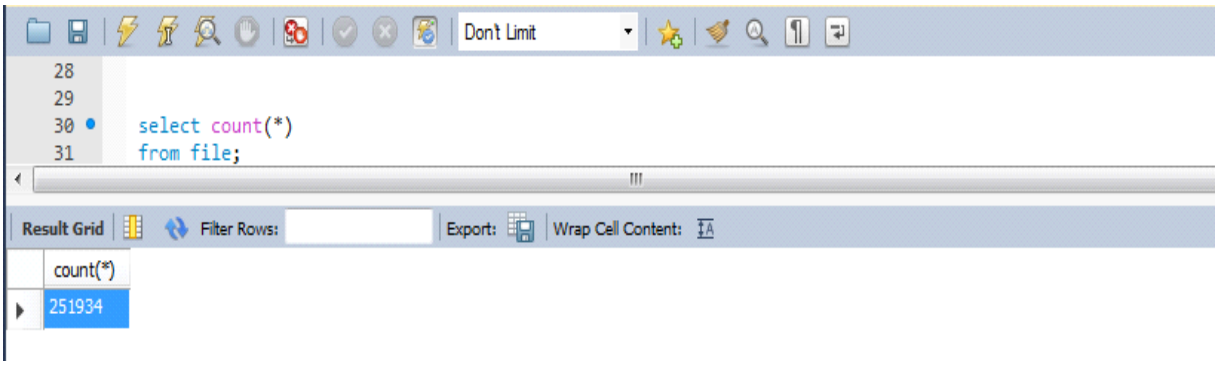
file 42

Output

#	Time	Action	Message	Duration / Fetch
1	14:03:32	select * from file where FILE_EXTENTION="pdf"	59 row(s) returned	0.000 sec / 0.000 sec

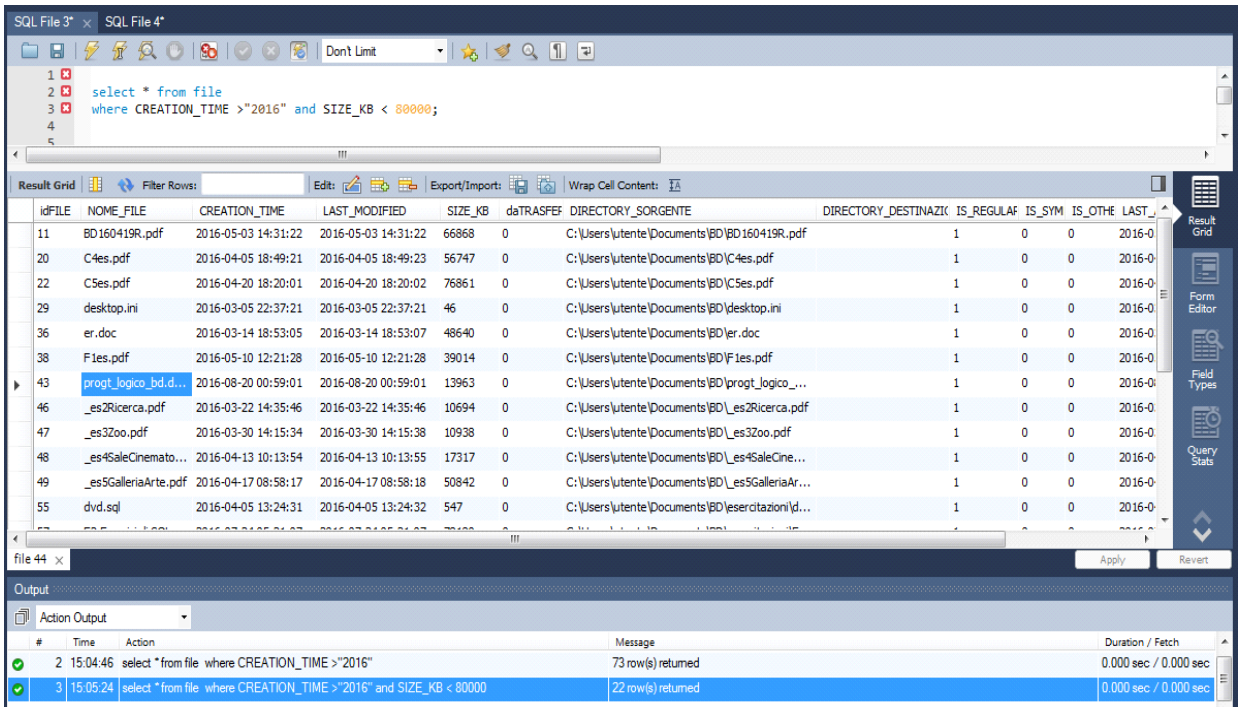
Numero di tutti i file distinti

```
SELECT count(*) FROM file;
```



Seleziona tutti i file che sono stati creati cominciando con 2016 e non superano la dimensione di 80 kb;

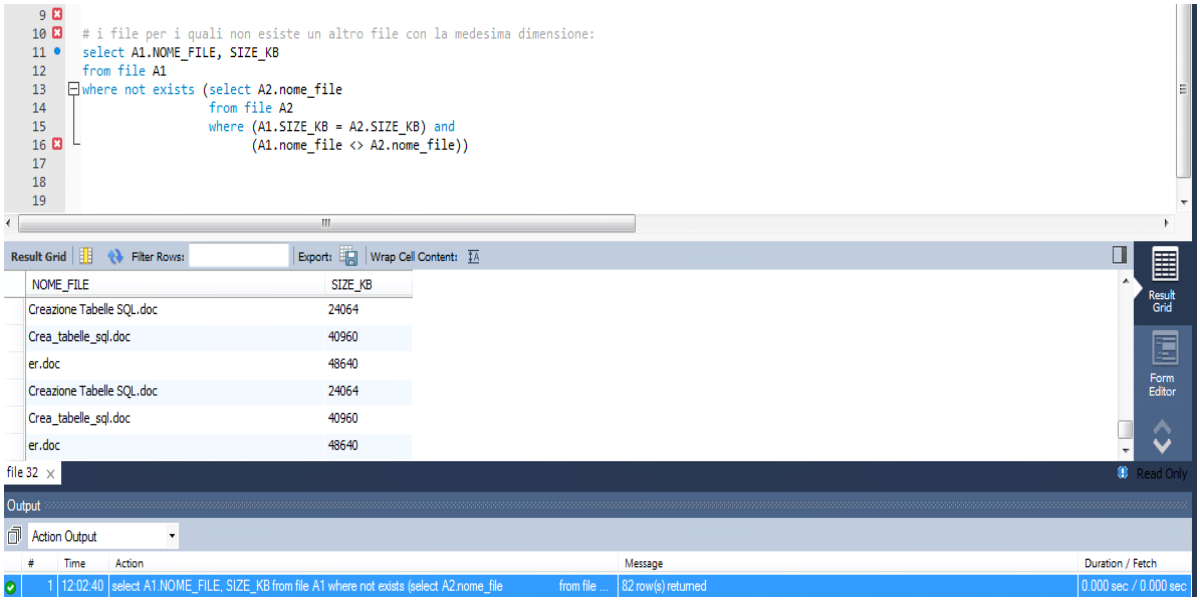
```
SELECT * from file
WHERE CREATION_TIME >"2016" and SIZE_KB < 80000;
```



Vediamo ora una soluzione che usa una query nidificata e l'operatore exists/ not exists. Questi operatori hanno come argomento una query interna e restituisce vero/falso se e soltanto se il risultato della query argomento contiene qualche elemento:

Per i file per i quali non esiste un altro file con la medesima dimensione.

```
SELECT A1.nome_file, size_kb
FROM file A1
WHERE NOT EXISTS (select A2.nome_file
 FROM file A2
 WHERE (A1.size_kb = A2.size_kb) and
 (A1.nome_file <> A2.nome_file));
```



File per la quale esiste un altro file con la medesima size_kb.

```
SELECT idFILE, nome_file, size_kb
FROM file A1
WHERE EXISTS (select A2.nome_file
 FROM file A2
 WHERE (A1.size_kb = A2.SIZE_KB) AND
 (A1.nome_file <> A2.nome_file))
```

```
select * from file;

#file per i quali esiste un altro file con la medesima size_kb.
select idFILE, nome_file, SIZE_KB
from file A1
where exists (select A2.nome_file
              from file A2
              where (A1.size_kb = A2.SIZE_KB) and
                    (A1.nome_file <> A2.nome_file))
```

idFILE	nome_file	SIZE_KB
104	Progetto.doc	662016
105	Progetto_logico - Copia.doc	36352
106	Progetto_logico.doc	36352
107	Query Inserimento - Copia....	26112
108	Query Inserimento.doc	26112
109	Schema_ER - Copia.doc	89088
110	Schema_ER.doc	89088
111	Tesina - Copia.doc	941056
112	Tesina.doc	941056

#	Time	Action	Message	Duration / Fetch
10	12:18:41	select nome_file from file A1 where exists (select A2.nome_file from file A2 where (A...	30 row(s) returned	0.000 sec / 0.000 sec
11	12:18:58	select idFILE, nome_file, SIZE_KB from file A1 where exists (select A2.nome_file from file A2 ...	30 row(s) returned	0.000 sec / 0.000 sec

QUERY DI MODIFICA

Per tutti i file che hanno l'estensione 'doc' aggiorna il campo "directory_destinazione" inserendo il percorso di una directory

```
UPDATE file SET DIRECTORY_DESTINAZIONE =
"C:\\Users\\utente\\Desktop\\file_copiati_db"
WHERE file_extention="doc";
```

idFILE	NOME_FILE	CREATION_TIME	LAST_MODIFIED	SIZE_KB	daTRASFERIR	DIRECTORY_SORGENTE	DIRECTORY_DEST	IS_REGULAR	IS_SYMBOLI	IS_OTHE	LAST_ACCESSE	FILE_EXTENTION
1	Dispensa1_Tecnica fotogra...	2016-08-19 11:37:24	2016-08-19 11:37:28	3452722	0	C:\Users\utente\Desktop\gioiel...	C:\Users\utent...	1	0	0	2016-08-19 ...	doc
2	markup-gioielli-reputation-...	2016-08-18 20:30:13	2016-08-18 20:30:23	216480	0	C:\Users\utente\Desktop\gioiel...		1	0	0	2016-08-18 ...	png
3	NaDi_fotografico per nego...	2016-08-20 21:03:18	2016-08-20 20:58:47	10840469	0	C:\Users\utente\Desktop\gioiel...		1	0	0	2016-08-20 ...	pdf

Altri esempi di query si trovano nell'appendice.

Capitolo 4. Implementazione in Java

Dopo aver accennato in diverse occasioni ad alcune funzionalità richieste dal committente, il lettore dovrebbe essersi fatto ormai un'idea del software che verrà esposto nel cuore di questa tesi. Si esporranno dettagliatamente le principali funzioni implementate con riferimento al codice Java. Si cercherà di seguire l'implementazione della classe Java con il percorso in ordine temporaneo della procedura d'inserimento nelle basi di dati e la consultazione del risultato inserito. Ci fermeremo di più sulle spiegazioni delle classi che costituiscono l'essenza del progetto. Dopo aver prodotto la DB, interveniamo con le classi di Java per accedere ad esso (DB) tramite il protocollo JDBC[8]. Per prima cosa creiamo il modello di dati (la classe da persistere il database)

[8] JDBC- Java Database Connectivity

basilare, rappresentata dalla tabella *File*. Dopo aver creato la connessione al server, applicheremo le metodologie d'inserimento dei dati nelle tabelle.

Alla fine del capitolo presenteremo l'estrazione di metadati dai documenti specifici.

4.1 Modello di dati

Finora abbiamo discusso di tutto ma non della provenienza dei dati della tabella e come sono stati forniti per il progetto. Facciamo un passo indietro ritornando al requisito il quale richiede: "*l'individuazione file da migrare*".

Concettualmente si crea una classe per cercare tutti i documenti. La classe *FindFile* svolge tale compito ed eseguirà una ricerca nella cartella e/o sottocartella memorizzata nella variabile *pathDelFile* di tutti i documenti in essa contenuti. *PathDelFile* è valore di tipo Stringa, indica il percorso (path) di un file.

Il metodo *readPath* esegue la ricerca e memorizza nella variabile *inDir[]* (variabile di tipo ArrayList) tutti i documenti trovati:

```
public void readPath() throws SQLException {
File directoryInizio = null;

//condizione iniziale da verificare (che il parametro sia un
    directory)
if (directoryInizio.isDirectory()) {

// aggiunge alla lista i percorsi del file
listDir.add(pathDelFile);
}
else
    logger.debug("Non è una directory");
// cerca i file tra i directory
while (!listDir.isEmpty()) {

File toCheck = new File(listDir.remove(0));

File[] inDir = null;
```

```
    try {

// array of files and directory
inDir = toCheck.listFiles();
logger.debug("Directory: " + toCheck.getAbsolutePath());
        }
//se la cartella in cui siamo arrivato a cercare il file è
    //un documento
        // null stampa il messaggio
//  altrimenti  esegue  il  ramo  else  (torna  in  un  altro
    directory a
// cercare i files)
    if (inDir == null) {
    logger.debug("Directory  vuota  o  non  accessibile  "  +
    toCheck.getAbsolutePath());
              } else {
for (File inFile: inDir) {
if (inFile.isFile()){
fileProprietY(inFile);
        else{
//altrimenti inFile e un directory
listDir.add(inFile.getAbsolutePath());
                        }
        }
```

La lista dei file ottenuti contiene gli oggetti necessari per la migrazione, per l'inserimento nel database. Il metodo *`filePropriety`* cha prende come parametro di ingresso il "*`myfile`*" (l'oggetto file), che rappresenta l'astrazione della tabella *`File`* estrae tutti gli attributi possibili e necessari dall'oggetto file e li memorizza nell'attributo *attr*. I campi della tabella *`File`* saranno a punto quelli definiti dall'attributo *`attr`*.

```
public File fileProprietY(File myfile){
BasicFileAttributes attr=null;
attr = Files.readAttributes(myfile.toPath(),
    BasicFileAttributes.class);

logger.info("file: "+myfile.getName());
logger.info("file path = " + myfile.getAbsolutePath());
logger.info("file name = " +
    myfile.getName().toLowerCase());
logger.info("creationTime = " + attr.creationTime());
logger.info("lastAccessTime = " + attr.lastAccessTime());
logger.info("lastModifiedTime = " +
```

```
        attr.lastModifiedTime());
    logger.info("isDirectory = " + attr.isDirectory());
    logger.info("isOther  = " + attr.isOther());
    logger.info("isRegularFile = " + attr.isRegularFile());
    logger.info("isSymbolicLink = " + attr.isSymbolicLink());
    logger.info("size  = " + attr.size());
    logger.info("estensione del file=" +ext2);
```

4.2 Sviluppo delle classi Java

La Figura 8, mostra la sequenza dello sviluppo di una applicazione Java con un database, sono i passi necessari da compiere per la realizzazione del progetto.

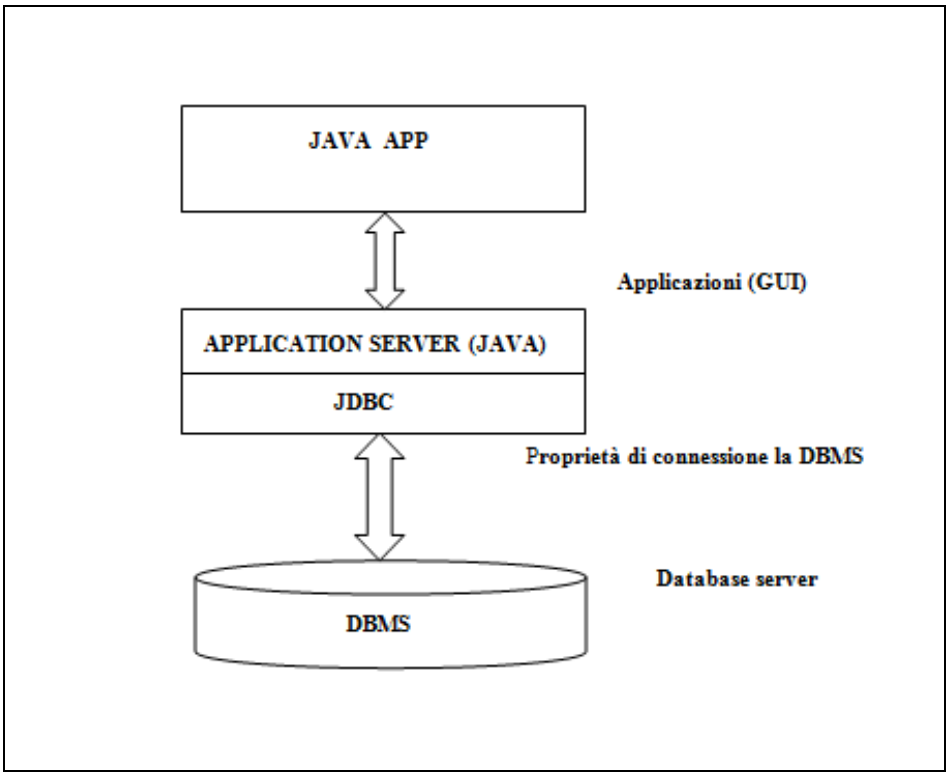


Figura 8 Java Applicazioni con DB server

4.2.1 Classe di dominio

Cominciamo quindi con la creazione della prima classe, che rappresenta la nostra tabella *FilePer* arrivare ad inserire i dati nella database dobbiamo creare l'oggetto. Il nostro oggetto d'interesse è un documento che ha certe proprietà. Le proprietà sono gli attributi dell'oggetto file ottenuti attraverso i costruttori della classe File, rilevati nel paragrafo 4.1.

Il nostro modello da rappresentare è la classe *FileText*, con i relativi metodi accessori (getters e setters) i cui attributi sono i campi della tabella *File*.

Errore. L'origine riferimento non è stata trovata.

```
public class FileText{
  int idFile;
  long sizeFile;
  String nomeFile;
  String dir_destinazione;
  String dir_sorgente
  FileTime dataCreazione;
  FileTime ultimaModifica;
  FileTime ultimoAcceso;
  int daTrasferire=0;
  boolean isRegular;
  boolean isSymbolic;
  boolean isOther;
  String fileExtention;
}
```

Il secondo passo richiede la creazione di un'interfaccia, chiamata *FileFill*. In essa dichiariamo alcuni metodi sull'oggetto *file*: inserimento, cancellazione, modifica, ricerca per ID, ricerca per nome e ricerca per *path*. Se necessario si possono definire degli altri metodi sempre nella stessa interfaccia..

```
/**
 * interfaccia FileFill dichiara le operazione che verranno
    effettuate sulla database.
*/
public interface FileFill {
    public int insert(FileText filetext);
    public int delete(int idfile);
```

```
        public int update(FileText filetext);
        public FileText searchById(int idfile);
        public FileText searchByNome(String nome_file);
        public FileText searchByPath(String percorso_file);  }
```

Potremo definire i metodi direttamente nella classe che li implementa, è vero, però l'uso di un'interfaccia nel nostro caso diventa inevitabile, poiché essa costituisce una pura specifica di comportamenti [1]. Tuttavia, vogliamo organizzare il codice che sia riutilizzabile, quindi l'uso di un'interfaccia serve per separare le definizioni dall'implementazione dei metodi, di cui abbiamo fatto uso.

4.2.2 Classi DAO

Una delle peculiarità principali di Spring è quella di favorire la scrittura di codice modulare permettendone il riuso. Per questa ragione Spring propone l'utilizzo del DAO (Data Access Object), un pattern architetturale che ha come scopo di separare le logiche di business da quelle di accesso ai dati.

L'idea alla base di questo pattern è di descrivere le operazioni necessarie per la persistenza del modello in un'interfaccia e di implementare la logica specifica d'accesso ai dati in apposite classi.

Le classi Data Access Objects/Persistence gestiscono l'accesso ai dati, incapsulando le modalità di comunicazione con il DB. Le DAO si fanno carico di gestire il codice SQL. DAO esegue il flusso di lavoro JDBC di base, lasciando al codice dell'applicazione fornire ed estrarre risultati di SQL. I metodi implementati da questa classe eseguono operazioni sul database mediante la variabile *jdbcemplate*.

Nel nostro progetto abbiamo quattro classi che adottano la strategia DAO. Faremo riferimento con esempi e dettagli su *FileTextDao*, la classe predominante del nostro progetto. Le altre ripetono lo stesso principio.

FileTextDao contiene i metodi d'interrogazione e manipolazione della classe di dominio, *FileText*. I metodi che espone sono:

- *insert (FileText)*, che invoca l'inserimento sul DB

• *update(FileText)*, che invoca l'aggiornamento sul DB.

• *delete(int)*, che invoca la cancellazione dal DB.

•*searchByNome(String)*,*searchById(int)*,*searchByPath(string)*, che invoca il caricamento dal DB.

Prendiamo il metodo per l'inserimento dei dati per discuterne il funzionamento:

```
public int insert(FileText filetext) {
   String sql = "insert into file( nome_file, creation_time,
   last_modified, size_kb, daTrasferire, directory_sorgente,
   directory_destinazione, is_regular,is_symbolic, is_other,
   last_accessed_time, file_extention) values ( ?, ?, ?, ?, ?,
   ?, ?, ?, ?, ?, ?, ?)";

   jdbcTemplate.update(sql, new Object[] {
   filetext.getNomeFile(), new
   Timestamp(filetext.getDataCreazione().toMillis()), new
   Timestamp(filetext.getultimaModifica().toMillis()),
   filetext.getSizeFile(),
   filetext.getDaTrasferire(),filetext.getDir_sorgente(),filet
   ext.getDir_destinazione(), filetext.isRegular(),
   filetext.isSymbolic(), filetext.isOther(), new
   Timestamp(filetext.getUltimoAcceso().toMillis()),
   filetext.getFileExtention()});
return 0;
```

Il metodo *insert*, come menzionato prima, usa l'oggetto di classe *FileText*, modello di dati della tabella ***File*** nel nostro DB. L'interrogazione *insert into file* comporta l'inserimento dei dati nella tabella, rispettando rigorosamente le regole sintattiche del linguaggio SQL. Per accedere allo spazio del DBMS (nel nostro caso l'inserimento di dati nella tabella) serve l'intervento della variabile *Jdbctemplate*. Essa permette di fornire ed estrarre codice SQL per i metodi. È usata per ogni metodo che implica la necessità di intervenire con il DB. *Jdbctemplate* gestisce la connessione delle nostre operazioni con il server.

L'operazione d'inserimento si conclude con successo se il valore del return equivale a 0, che corrisponde a true.

Se invece dobbiamo lavorare con diverse istanze di dati si deve creare una classe DAO per ogni classe che rappresenta entità del dominio di applicazione.

Mostriamo un altro esempio che effettua l'operazione di eliminazione di un record dal DB.

Stavolta abbiamo come parametro d'ingresso un intero che rappresenta il codice identificativo del record che sarà eliminato.

```
      Delete query for a record

public int delete(int idfile) {
  jdbcTemplate.update("delete from file where
        idfile = ?", idfile);
return 0;
          }
```

4.2.3 Classi RowMapper(recuperare multipli oggetti)

Java usa una classe speciale per estrarre i dati da un database, conosciuta come *RowMapper*. Nel paragrafo precedente, 4.2.2 Classi DAO, abbiamo utilizzato dei metodi che operano con oggetti Java nel DB. Gli oggetti non sono altro che i record del database che andiamo a recuperare utilizzando i metodi che eseguono SELECT query(). Quindi ai metodi `query`() e `queryForObject`() di `Jdbctemplate` viene passata un'istanza di *RowMapper* (vedi oltre per il sorgente).

L'interfaccia `RowMapper` permette di associare alle righe del ResultSet le istanze di oggetti del dominio dell'applicazione, nel nostro caso le classi modello: `FileText, FilePf, FileXlsx, FileOffice`.

L'interfaccia usa un solo metodo `mapRow` che accetta come parametri l'istanza del ResultSet e un int che rappresenta la riga della tabella in cui si trova nel DB.

Questa è la classe, `FileRowMapper`, che fa da mapper per la tabella `File`:

```
Class FileRowMapper.java
public class FileTextMapper implements RowMapper<Object>
{
public Object mapRow(ResultSet rs, int rowNum) throws
     SQLException {
     FileText filetext = new FileText();
     filetext.setIdFile(rs.getInt("idFile"));
     filetext.setNomeFile(rs.getString("nome_file"));
     filetext.setSizeFile(rs.getInt("size_kb"));
     filetext.setultimaModifica(FileTime.
    fromMillis(rs.getTimestamp("last_modified").getTime()));
     filetext.setUltimoAcceso(FileTime.fromMillis
         (rs.getTimestamp("last_accessed_time").getTime()));
     filetext.setDataCreazione(FileTime.fromMillis
         (rs.getTimestamp("creation_time").getTime()));
     filetext.setDir_sorgente(rs.getString
         ("directory_sorgente"));
     filetext.setDir_destinazione(rs.getString
         ("directory_dest inazione"));
     filetext.setFileExtention(rs.getString("file_extention");
     filetext.setOther(false);
     filetext.setRegular(false);
     filetext.setSymbolic(false);
     filetext.setDaTrasferire(rs.getInt("daTrasferire"));
return filetext;
         }
     }
```

4.2.4 Recuperare singoli oggetti

In precedenza abbiamo parlato di come si estraggono i dati con la *RowMapper*, una classe destinata ad eseguire SELECT query per ottenere un set di risultati con oggetti di un unico tipo.

Esiste un'altra alternativa che ci permette di recuperare i dati. Stavolta si tratta di *ExtractResultSet*.

Supponiamo di voler importare il risultato della query in un unico oggetto in cui, verificando un valore di un campo (*daTrasferire*), desideriamo modificarlo con un

altro. Non possiamo farlo direttamente con il *RowMapper* perché non abbiamo accesso a una singola riga dal ResultSet. Per raggiungere quest'obiettivo dobbiamo scrivere un *ResultSetExtractor*. Ecco un esempio:

```
String sql ="select * from file where daTrasferire =1";
  rs= stmt.executeQuery(sql);
  while(rs.next()){
    FileText filecopy= (FileText)(new
    FileTextMapper().mapRow(rs, 0));
      fff.copyFileUsingFileChannels(filecopy);
      filecopy.setDaTrasferire(2);

    FileTextDao g = new FileTextDao();
      g.setJdbcTemplate(template);
      rs.getInt("daTrasferire");
      rs.updateInt( "daTrasferire", 2);
      rs.updateRow();
                    }
```

Per concludere, la differenza tra *RowMapper* e *ResultSetExtrator* è che con il *RowMapper* possiamo accedere al risultato di una singola riga, però con il *ResultSetExtrator* si può utilizzare l'intero set di risultati. Se abbiamo bisogno di mappare diverse righe restituite dalla query per un singolo oggetto, dovremo usare *l'Extractor* però in altri casi, un mapper dovrebbe essere sufficiente.

Sia per *RowMapper* sia per *ResultSetExtractor* l'oggetto *ResultSet* mantiene una connessione ad un database, per questo non può essere serializzato e imoltre non possiamo passare l'oggetto *ResultSet* da una classe all'altra attraverso la rete. A questo proposito c'e bisogno di creare un'interfaccia che gestisca una connessione al database- JDBC Statement (4.3.1 L'invio dei comandi.)

4.3 JDBC Connection

JDBC è l'API che permette alla nostra applicazione in Java di accedere al database. Definita nel package java.sql (javax.sql per le funzionalità server) dispone

dell'oggetto di tipo java.sql.Connection che rappresenta una connessione (sessione) con un database. I comandi al DBMS ed i risultati sono inviati e ricevuti tramite la connessione.

L'architettura di JDBC prevede l'utilizzo di un "driver manager", che espone alle applicazioni un insieme d'interfacce standard e si occupa di caricare a run-time i driver opportuni per "pilotare" gli specifici DBMS. Le applicazioni Java utilizzano le JDBC API per comunicare con il JDBC driver manager. Il driver manager usa le JDBC API per parlare con i singoli driver che pilotano gli specifici DBMS.

Java JDBC usa due interfacce che aiutano a creare una connessione: Driver Manager e DatatSource. Esse sono responsabile dell'esecuzione delle query necessarie per il database.

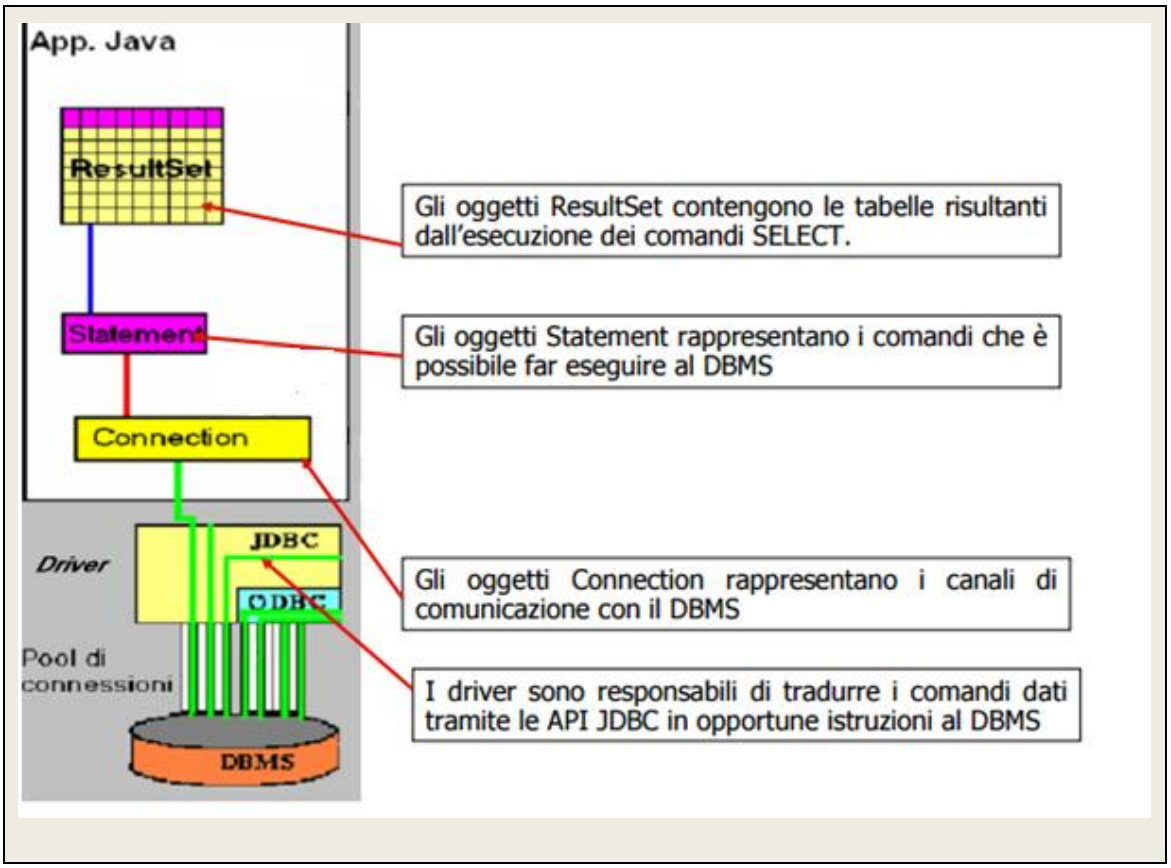


Figura 9 DB Connection

Per connettersi al server si devono usare i parametri locali, passati al DBMS quando inizializzano il collegamento per la prima volta.

```
Connect database URL=
    'jdbc:mysql://localhost:nr_porta/nome_database'

   Si esecuta una query DB_name=INNER DB;
```

DriverManager: Questa interfaccia è utilizzata per contenere le informazioni riguardanti il database come nome della classe, URL connection, username e password. Come viene esemplificato nell'immagine 10.

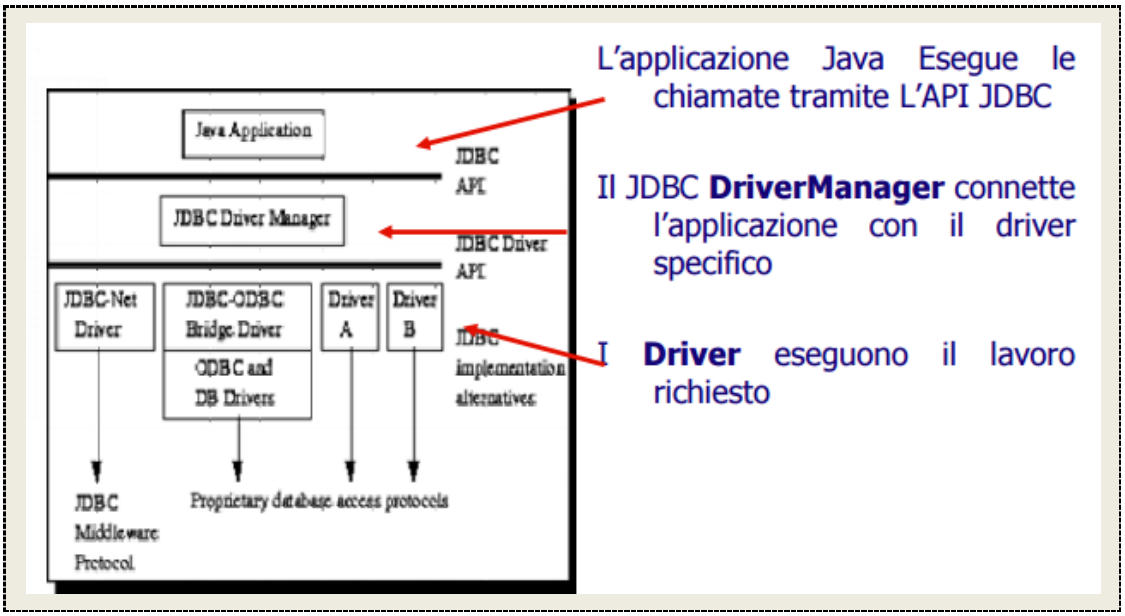


Figura 10 JDBC uso dei driver.

DataSource: Questa interfaccia è preferita invece da chi richiede connessioni particolari. Le proprietà di un oggetto `DataSource` sono impostate in modo tale che essa rappresenta una particolare `DataSource`. Questa funzionalità è implementata nel nostro file di configurazione `(bean.xml)` nel modo corrispondente:

```
<!-- Initialization for data source -->
<bean id="dataSource"
   class="org.apache.commons.dbcp2.BasicDataSource" >
   <property name="driverClassName" value="com.mysql.jdbc.Driver"/>

<property name="url"
```

```
      value="jdbc:mysql://10.0.1.247:3306/alfImporter" />
     <property name="username" value="alfImporter" />
     <property name="password" value="xxxxx" />
   </bean>
```

Driver class: è la classe per la connessione di MySQL database è **com.mysql.jdbc.Driver**.

Connection URL: l'URL per la connessione di the MySQL database è **jdbc:mysql://localhost:3306/alfimporter** dove jdbc è il nostro tipo di applicazione, MySQL è il database, localhost è il nome del server su quale MySQL sta girando, 3306 è il numero di porta, alfimporter è il nome del database.

Ciascuna connessione al database permette l'esecuzione di comandi SQL del tipo: CREATE, INSERT, UPDATE, DELETE e SELECT, usati da ResultSet.

4.3.1 L'invio dei comandi

Tramite una connessione è possibile inviare al database comandi modellati da tre diversi tipi di oggetti:

1. Statement: creati dal metodo `createStatement`, rappresentano comandi SQL senza parametri.

2. `PreparedStatement`: creati dal metodo `prepareStatement`, rappresentano comandi SQL che possono essere precompilati nel database ed accettare parametri (di IN) al momento dell'esecuzione

3. `CallableStatement`: creati dal metodo `prepareCall`, che possono essere usati per eseguire stored procedure ed accettano anche parametri di OUT e INOUT.

Prima di poter utilizzare un oggetto Statement per eseguire un'istruzione SQL, è necessario creare l'oggetto *Connection* che utilizza il metodo *createStatement()*, come nel seguente codice:

```
Statement stmt = null;
Connection con = DriverManager.getConnection();
try {
stmt = con.createStatement();
}
catch (SQLException e) {
 . . .
}
finally {
 . . .
}
```

I tre metodi che usano Statement sono:

■ **public ResultSet executeQuery(String sql):** utilizzato per eseguire SELECT query. Restituisce un oggetto di tipo ResultSet.

■ **public int executeUpdate(String sql):** utilizzato per eseguire query specifici di tipo create, drop, insert, update, delete, etc.

■ **boolean execute (String SQL)**: restituisce un valore booleano se un oggetto ResultSet può essere ottenuto, altrimenti è false. Tale metodo può essere utilizzato per eseguire SQL DDL[9] statements o quando vi è la necessità di usare query dinamiche.

Il ResultSet è il responsabile per la creazione del collegamento, vari esempi sono stati esposti nei paragrafi Classi RowMapper(recuperare multipli oggetti) e 4.2.4 Recuperare singoli oggetti

[9] DDL- Data Definition Languagge, è uno dei tipi di statement che dispone il lingiaggio SQL, effettua operzioni di creazione, modifica e cancellazione tabela; analisa informazioni su tabelle, etc.

4.4 File di configurazione

Nel codice Java sono creati i metodi che compongono un'applicazione. Invece in Spring, c'è la necessità di creare un file di configurazione in cui ritrovare la cosiddetta "regola" che dà all'interprete regole su come eseguire il codice.

Gli oggetti importanti che costituiscono l'applicazione e che sono gestiti dal contenitore IoC (contenitore Spring) sono chiamati bean. Un bean è un oggetto che crea un'istanza, assemblato e gestito da un contenitore IoC bean. Questi bean sono inseriti in un file di configurazione, chiamato *bean.xml*.

Il file di configurazione rappresenta un contenitore usato per incapsulare la logica di costruzione tra l'oggetto in una o più classi. È utilizzato per costruire una configurazione tra gli oggetti presenti nelle varie classi che hanno creato tra di loro una certa dipendenza. Il tipo di rapporto che queste istanze creano viene rappresentato nel *file.xml*.

Il file *bean.xml* viene inserito nella stessa cartella che contiene tutte le altre classi di Java.

```
<!-- Definition of beans -->
<bean id="jdbcTemplate"
  class="org.springframework.jdbc.core.JdbcTemplate">
  <property name="dataSource" ref="dataSource" />
</bean>

<bean id="findfile"
  class="com.netsphere.alfimporter.FindFile">
  <property name="jdbcTemplate" ref="jdbcTemplate"/>
  <property name="pathDelFile" value="F:\\" />
  <property name= "operazione" value= "1" />
</bean>

<bean id="filesearch"
  class="com.netsphere.alfimporter.FileTextDao">
  <property name="jdbcTemplate" ref ="jdbcTemplate"/>
</bean>

<bean id="fileduplicate"
  class="com.netsphere.alfimporter.TrovaDuplicatiReport">
```

```
        <property name="template" ref ="jdbcTemplate"/>
      </bean>
```

Analizzando il file di configurazione possiamo notare che è stato chiesto al container di gestire le istanze di tre classi:

■ com.netsphere.alfimporter.FindFile – che ha come bean l'oggetto *findfile*

■ com.netsphere.alfimporter.FileJDBCDao – ha il bean chiamato *filesearch*

■ com.netsphere.alfimporter.TrovDuplicatiReport - ha il bean con il nome *fileduplicati*

Ora spieghiamo come sono interpretate le dipendenze tra gli oggetti.

Al bean *findfile* è stata iniettata, tramite un'injection di tipo setter (setJdbcTemplate), un riferimento alla classe *DataSource* (Data source e la classe che crea la connessione con il DB, paragrafo precedente) Il bean stesso dispone di tre parametri: *Jdbctemplate*, *pathDelFile*, *operazione*, i quali vengono istanziati con valori.

La procedura implica che l'utente non deve entrare nel codice Java per passare valori alle variabili, può però inserire, modificare e cancellare direttamente dal file di configurazione. Questa è la caratteristica più vantaggiosa dell'uso del *file.xml*.

Al bean *filesearch* è iniettato il bean *Jdbctemplate*, con un'Injection di tipo setter, che si riferisce sempre all'istanza della classe *DataSource*.

Al bean *fileduplicati* è stato iniettato la stessa istanza per il parametro *Jdbctemplate* trovato nella classe *TrovaDuplicatiReport*, che si riferisce sempre all'istanza della classe *DataSource*.

Infine istanziamo il container e vediamo come interagire con esso per utilizzare i bean configurati. La regola prevede di eseguire *l'ApplicationContext* (classe per la configurazione del file di *bean.xml*) dalla classe principale, che nel nostro caso è classe *Test*.

```
ClassPathXmlApplicationContext context = new
  ClassPathXmlApplicationContext("bean.xml");

BasicDataSource ds =
  (BasicDataSource)context.getBean("dataSource");
```

Anche se `Jdbctemplate` costituisce un bean, esso si riferisce sempre alla classe `DataSource` che come detto prima è responsabile per la configurare il collegamento tra le classi di Java ed il DMBS.

4.5 Estrazione metadati

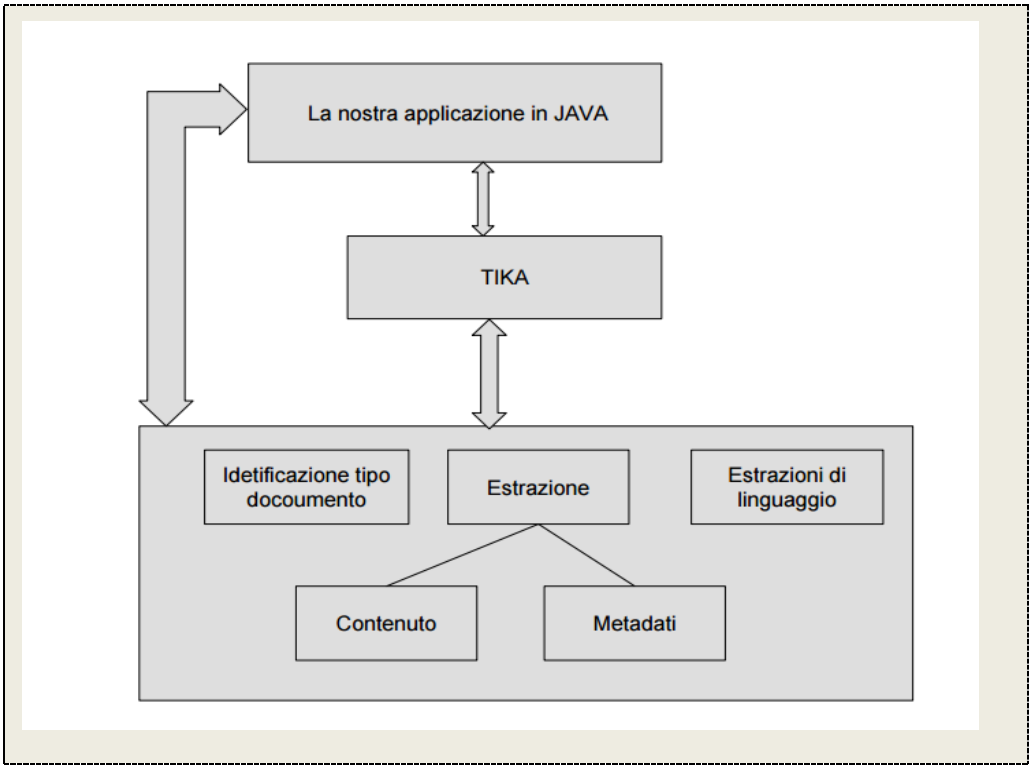


Figura 11 Tika structure.

Per l'estrazione dei metadati abbiamo incorporato Tika nella nostra applicazione. Tika utilizza vari `parser` per l'identificazione del tipo di documento usando le sue tecniche per rilevare ed estrarre i dati. I metadati non sono altro che

ulteriori informazioni di file. Se consideriamo un file audio, il nome dell'artista, nome dell'album, il titolo, data registrazione sono esempi di metadati [13,14].

La figura 11, illustra come funziona l'estrazione dei dati da un programma in Java.

Nel nostro caso invece dovremo estrarre i dati dei tutti i file con un contenuto testuale. Tra di loro si elencano documenti di tipo pdf, doc, otd, docx. Altri tipi di file non ci interessa.

Per questo si esegue prima il *parsing* del file, viene identificato di che tipo sia. Se per esempio è di tipo pdf, si leggono i metadati si crea l'oggetto *DocumentoPDF* e siinseriscono nella tabella di basi dati corrispondente. *DocumentPDF* è la classe che rappresenta il modello di dati, la tabella *filePdf* nel nostro database.

Sotto si presenta un pezzo di codice, della classe *FileContent,* che estrai i metadati per un documento di tipo pdf.

```
public class FileContent {
  if (filetype.equals("pdf")){
   DocumentoPDF docpdf= new DocumentoPDF();
     docpdf.setIdfileContent(idfilecollegato);
     docpdf.setDocument_type(filetype);
     docpdf.setAuthor(metadata.get(PDF.DOC_INFO_CREATOR));
     docpdf.setTitle(metadata.get(PDF.DOC_INFO_TITLE));
     docpdf.setProducer(metadata.get(PDF.DOC_INFO_PRODUCER);
     docpdf.setContent_type(metadata.get("Content-Type"));
      docpdf.setLast_modified(metadata.getDate(PDF.DOC_INFO_
      MODIFICATION_DATE));
      docpdf.setCreated_date(metadata.getDate(PDF.DOC_INFO_C
      REATED));
     docpdf.setInfo_subject(
      metadata.get(PDF.DOC_INFO_SUBJECT));
      docpdf.setKey_words(metadata.get(PDF.DOC_INFO_KEY_WORD
      S));
      docpdf.setPages_nr(Integer.parseInt(metadata.get("xmpT
      Pg:NPages")));
     docpdf.setPages_nr(0);
            }
scriviSuDbPdf(docpdf);
                  }
```

Alla fine dell'esecuzione del *parsing* chiama il metodo *scriviSuDbPdf* che inserisci i dati nella tabella corrispondente. Le tabelle del basi di dati sono risultato di questo pezzo di programma, in quanto per ogni tipo di documento abbiamo creato una tabella che le dimostra l'autenticità dei dati. Invece per i documenti di tipo .doc, .docx, e .odt abbiamo creato un'unica tabella in quanto le caratteristiche e l'estrazione dei metadati sono identiche per tutte.

Conclusioni

Nella scrittura di questa la tesi abbiamo incluso le informazioni che permettessero di capire come si è svolto lo sviluppo dell’applicazione. Abbiamo introdotto la parte teorica necessaria per lo sviluppo del programma nella prima parte della tesi. E nella seconda invece ci siamo concentrati sull’implementazione del programma.

Si è partiti dall'analisi e dalla ricerca dalla creazione di un database che avesse lo scopo di raccogliere file presenti nei vari punti di storage del server aziendale. In seguito, abbiamo progettato e sviluppato un software che fosse in grado di gestire i dati inseriti nel database con lo scopo di essere sottoposti a eventuali operazioni di scrematura e spostamenti nelle cartelle prescelte dal gestore.

Grazie allo studio di nuove tecnologie di sviluppo e strumenti di elaborazione è stato possibile creare l’applicazione che implementa le funzionalità ad hoc per lo scopo prefissato. Concludendo, possiamo dire di aver compiuto il lavoro necessario, ed elenchiamo i punti realizzati:

- ☑ È stato creato un nuovo database con le tabelle

- ☑ È stato effettuato l'inserimento dei dati nel DB con successo
- ☑ È stato creato il programma per la scrematura dei duplicati
- ☑ È stato creato il programma per la migrazione dei file
- ☑ È stato creato il programma per l'estrazione dei metadati.

Per quanto riguarda gli sviluppi futuri si potrebbero migliorare i seguenti aspetti:

- ☑ Creazione di nuove aree documentale di accesso ai dati
- ☑ Importazione dei metadati su Alfresco

Per quanto riguarda gli sviluppi futuri dei due aspetti riportati possiamo specificare le cause per cui la situazione si presenta problematica.

Tra i punti da realizzare nei requisiti si definiva "Individuazione di massima di quante nuove aree documentali sono da creare". Lo scopo era che dopo la migrazione dei documenti si pensava di evolvere il software adattandolo a un nuovo gestionale. Invece durante l'esecuzione del progetto abbiamo tolto l'individuazione di massima delle nuove aree documentali, in quanto, ad un'analisi più approfondita è risultato non automatizzabile. I metadati di base dei file erano poco rappresentativi. Abbiamo allora aggiunto un'estrazione più approfondita dei metadati di tutti i contenuti di tipo "documento Office" usando le librerie TIKA e creando tabelle ad hoc per i metadati aggiuntivi. Questa permetterà l'importazione in Alfresco.

Appendici

Codice 1: Tabella filepdf

```
select * from filepdf;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

idflexlsx	doc_type	author	title	producer	content_type	created_date	last_modified	creator
283	xlsx	NULL	NULL		application/vnd.openxmlf...	2006-09-16 00:00:00	2016-10-11 06:55:16	NULL
346	xlsx	carlo	NULL	NULL	application/vnd.openxmlf...	2014-01-29 11:08:37	2014-02-05 11:17:50	carlo
357	xlsx	Leonardo Mattioli	NULL	NULL	application/vnd.openxmlf...	2015-04-29 15:49:40	2015-05-05 12:05:54	carlo
360	xlsx	NULL	NULL		application/vnd.openxmlf...	2006-09-16 00:00:00	2014-11-20 15:02:35	NULL
387	xlsx	carlo	NULL	NULL	application/vnd.openxmlf...	2015-04-29 15:49:40	2015-11-18 15:22:54	carlo
392	xlsx	Leonardo Mattioli	NULL	NULL	application/vnd.openxmlf...	2015-04-29 15:49:40	2015-05-07 10:28:46	carlo
2307	xlsx	lcosta	NULL	NULL	application/vnd.openxmlf...	2014-08-11 09:09:48	2014-10-24 14:37:30	lcosta
2494	xlsx	daniele	NULL	NULL	application/vnd.openxmlf...	2013-01-18 10:00:01	2013-01-18 10:09:55	daniele
5145	xlsx	Daniele	NULL	Microsoft	application/vnd.openxmlf...	2013-08-22 13:16:41	2013-08-22 14:19:07	Daniele
5193	xlsx	Leonardo Mattioli	NULL		application/vnd.openxmlf...	2015-06-01 07:17:24	2015-06-01 09:25:13	Leonardo Mattioli
5194	xlsx	Leonardo Mattioli	NULL	NULL	application/vnd.openxmlf...	2012-07-10 10:08:07	2015-01-07 16:59:07	Biancoli Simone
5195	xlsx	Leonardo Mattioli	NULL	Enginee...	application/vnd.openxmlf...	2006-01-24 21:06:54	2013-06-12 07:32:04	Engineering.IT S...
5624	xlsx	ATTI	NULL	NULL	application/vnd.openxmlf...	2013-03-15 13:32:26	2013-03-15 13:45:35	ATTI

filexlsx 10 | Apply | Revert

Output

Action Output

#	Time	Action	Message	Duration / Fetch
12	08:08:15	select * from filexlsx	0 row(s) returned	0.000 sec / 0.000 sec
13	13:30:11	select * from file	135073 row(s) returned	0.062 sec / 3.557 sec
14	17:27:30	select * from filexlsx	1192 row(s) returned	0.093 sec / 0.016 sec

Codice 2: Query- all from fileoffice

Don't Limit

```
select * from fileoffice
where pages_nr between 20 and 50
and last_author="luca";
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idfileoffice	doc_type	author	title	producer	content_type	created_date	last_modified	pages_nr	last_author
3098	doc	luca	NULL		application/msword	2008-11-10 14:44:00	2008-11-11 10:33:00	28	luca
17090	docx	luca	NULL	NULL	application/vnd.ope...	2012-11-19 14:01:00	2012-12-05 16:49:00	29	luca
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

fileoffice 15 | Apply

Output

Action Output

#	Time	Action	Message	Duration
1	17:33:34	select * from fileoffice where pages_nr between 20 and 50	202 row(s) returned	0.015 se
2	17:34:14	select * from fileoffice where pages_nr between 20 and 50 and last_author="luca"	2 row(s) returned	0.000 se

Codice 3: Count all from tabella fileoffice

```
select count(*) from fileoffice;
```

count(*)
5055

Codice 4: Tabella fileoffice order desc

```
select * from filepdf;

group by content_type;
```

idfilepdf	doc_type	author	title	producer	content_type	created_date	last_modified	pages_nr	key_wor
11	pdf	NULL	NULL	GNU Ghostscript 6.51	application/pdf	NULL	NULL	3	NULL
12	pdf	NULL	NULL	GNU Ghostscript 6.51	application/pdf	NULL	NULL	3	NULL
13	pdf	NULL	NULL	GNU Ghostscript 6.51	application/pdf	NULL	NULL	6	NULL
14	pdf	NULL	NULL	GNU Ghostscript 6.51	application/pdf	NULL	NULL	14	NULL
15	pdf	NULL	NULL	GNU Ghostscript 6.51	application/pdf	NULL	NULL	3	NULL
17	pdf			PDFConverter	application/pdf	2005-12-21 10:58:53	NULL	1	
125	pdf	sld006	Microsoft Word - GestPay - Specifiche tecniche ...	Acrobat Distiller 5.0.5 (Windows)	application/pdf	2007-06-13 12:04:36	2007-06-13 12:04:36	44	NULL
127	pdf	NULL	NULL	NeoOffice 1.2 Beta	application/pdf	2006-02-03 12:45:17	NULL	16	NULL

Output

#	Time	Action	Message	Duration / Fetch
1	17:37:16	select * from fileoffice order by idfileoffice desc	5055 row(s) returned	0.016 sec / 0.047 sec
2	17:38:35	select * from pdf	Error Code: 1146. Table 'alfimporter.pdf' doesn't exist	0.000 sec
3	17:38:46	select * from filepdf	1885 row(s) returned	0.000 sec / 0.016 sec

Codice 7: Conta i file dalla tabella file

```
select count(*)
from file;
```

count(*)
251934

Codice 8: All from tabella filepdf

```
select * from fileoffice
order by idfileoffice desc;
```

idfileoffice	doc_type	author	title	producer	content_type	created_date	last_modified	pages_nr	last_author
156453	docx	NULL	NULL	NULL	application/vnd.ope...	2012-03-29 08:51:00	2012-04-10 15:28:00	21	NULL
156444	docx	NULL	NULL	NULL	application/vnd.ope...	2012-11-29 13:29:00	2012-12-06 09:01:00	4	NULL
156439	docx	NULL	NULL	NULL	application/vnd.ope...	2012-07-17 07:50:00	2012-07-19 13:35:00	52	NULL
156438	docx	NULL	NULL	NULL	application/vnd.ope...	2012-06-15 13:44:00	2012-06-15 13:44:00	52	NULL
156437	docx	NULL	NULL	NULL	application/vnd.ope...	2012-03-29 08:51:00	2012-05-09 15:56:00	1	NULL
156436	docx	NULL	NULL	NULL	application/vnd.ope...	2012-03-29 08:51:00	2012-05-07 12:04:00	1	NULL
156432	docx	G. Ripamonti	NULL	NULL	application/vnd.ope...	2014-01-08 07:49:00	2014-01-08 09:03:00	65	G. Ripamonti
156431	doc	G. Ripamonti	NULL	BRT	application/msword	1999-03-30 21:01:00	2000-06-20 13:00:00	1	G. Ripamonti
156424	doc	Davide Zucchetti	NULL	Coster T.E. S.p.A.	application/msword	2005-01-25 16:41:00	2005-01-25 16:41:00	116	Davide Zucchetti

#	Time	Action	Message	Duration
1	17:37:16	select * from fileoffice order by idfileoffice desc	5055 row(s) returned	0.016 sec

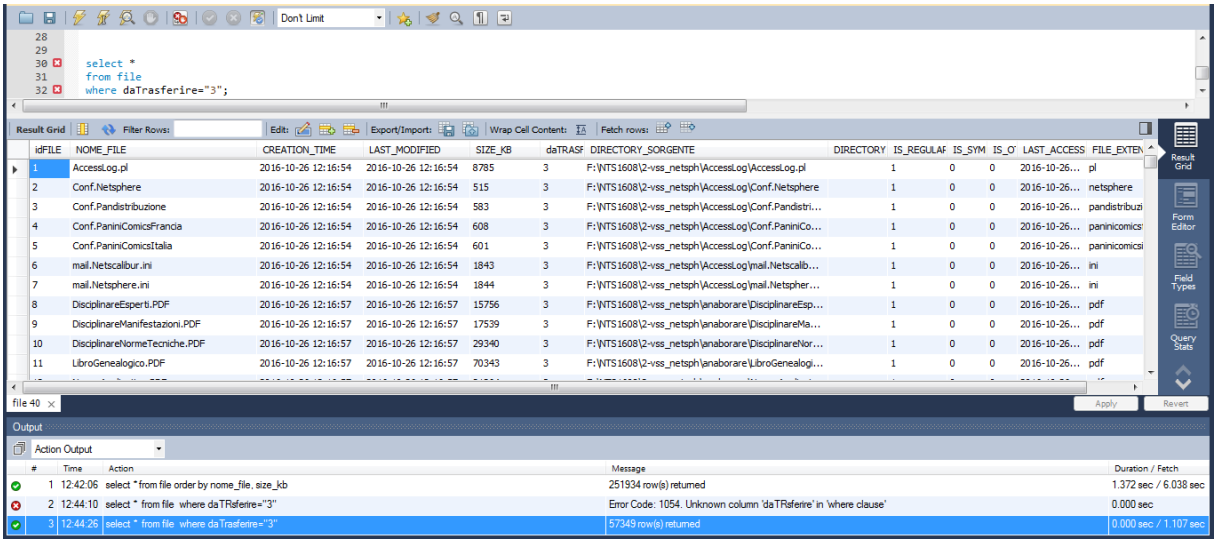
Codice 9: Select condizionale from tabella filepdf

```
select * from filepdf
where key_words like "Alfresco%";
```

idfilepdf	doc_type	author	title	producer	content_type	created_date	last_modified	pages_	key_w
5635	pdf	NULL	Alfresco Developer Series: Advanced Workflows	LibreOffice 3.4	application/pdf	2012-02-21 15:03:23	NULL	47	Alfresc
14441	pdf	NULL	Alfresco Developer: Developing Custom Actions	OpenOffice.org 2.0	application/pdf	2007-09-27 06:06:38	NULL	18	Alfresc
14444	pdf	Jeff Potts	Alfresco Developer Series: Working with Custo...	LibreOffice 3.4	application/pdf	2012-01-09 21:38:37	NULL	55	Alfresc
14616	pdf	NULL	Alfresco Developer Series: Advanced Workflows	LibreOffice 3.4	application/pdf	2012-02-21 15:03:23	NULL	47	Alfresc
14624	pdf	NULL	Alfresco Developer: Developing Custom Actions	OpenOffice.org 2.0	application/pdf	2007-09-27 06:06:38	NULL	18	Alfresc
28142	pdf	Luis Sala	Sala - Alfresco on AWS - Cost-Effective, Scalabl...	Mac OS X 10.9 Quartz PDFContext	application/pdf	2013-11-04 15:47:49	2013-11-04 15:47:49	47	alfresc
47082	pdf	NULL	Alfresco Developer: Developing Custom Actions	OpenOffice.org 2.0	application/pdf	2007-09-27 06:06:38	NULL	18	Alfresc
47085	pdf	Jeff Potts	Alfresco Developer Series: Working with Custo...	LibreOffice 3.4	application/pdf	2012-01-09 21:38:37	NULL	55	Alfresc

#	Time	Action	Message	Duration / Fetch
1	17:44:01	select * from filepdf where key_words like "Alfresco%"	8 row(s) returned	0.000 sec / 0.000

Codice 5: Tutti i record duplicati



Bibliografia

Riportiamo qui alcuni riferimenti bibliografici che hanno contribuito al completare ed integrare la presente tesi.

[1] G.Cabri, F.Zambonelli " Programmazione ad oggetti in Java: dai fondamenti a internet", Pitagora Bologna,2003.

[2] D.Beneventano, S.Bergamaschi, F.Guerra, M.Vincini "Progetto di Basi di Dati Relazionali: lezioni ed esercizi", Pitagora Bologna, 2007

[3] Mark Pollack, O. Gierke, T.Risberg"Spring Data: Modern Data Accesed for Enterprise Java", O'Reilly 2012

[4] Craig Walls "Spring in Action", third edition, Manning Publications 2011

[5] Wen-Chen-Hu, Naimu Taobouch, “Big Data Management Technologies and Applications”, IGI Global, 2014;

[6] Luca Vetti Tagliati, “Java Best Practice - I migliori consigli per scrivere codice di qualità “, tecniche nuove, 2008;

[7] Spring Web Flow 2 Web Development.pdf

[8] www.mysql.org

[9] http://titanic.fauser.edu/database/bibliog.htm

[10] http://www.javatpoint.com/java-jdbc

[11] http://www.html.it/pag/18720/

[12] http://dolly.fim.unimore.it/2015/course/view.php?id=28

[13] https://www.tutorialspoint.com/tika/tika_extracting_pdf.htm

[14] https://github.com/apache/tika/blob/master/tika-parsers/src/main/java/org/apache/tika/parser/microsoft/WordExtractor.java

[15] http://www.mkyong.com/tutorials/

[16] http://www.appuntisoftware.it/apache-poi-manipolare-documenti-microsoft-in-java/