UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica

Database spaziali e loro interazione con QuantumGIS

S. Ten. Pietro Russo

Tesi di Laurea

Relatore:
Prof.sa Letizia Leonardi

Correlatori:
Prof.sa Federica Mandreoli, Prof. Riccardo Martoglia

Anno Accademico 2009/2010

RINGRAZIAMENTI

Ringrazio tutti i professori del mio corso di laurea ed
in particolar modo la prof.sa Leonardi Letizia,
la prof.sa Mandreoli Federica e il prof. Martoglia Riccardo
per la loro continua, instancabile e preziosissima disponibilità.
Ringrazio inoltre il Cap. Criscuolo, tutti i miei colleghi
di Corso e i miei amici per avermi sempre sostenuto.
Un ringraziamento speciale va infine alla mia famiglia
che mi è sempre stata vicina e mi ha sempre aiutato ogni volta
che ne ho avuto bisogno.

PAROLE CHIAVE
Database spaziali
QuantumGIS
JDBS

INDICE

ELENCO DELLE FIGURE

Introduzione

Obiettivo di questo elaborato è quello di andare a studiare l'importanza dei database spaziali e la loro graficazione attraverso QuantumGIS, per poi andare a creare un'applicazione Java che possa essere utile sia in campo civile che, soprattutto, in campo militare e le cui funzionalità siano quelle di caricare dei file all'interno di un database e di visualizzarli.

L'interesse sia per i database che per la programmazione in Java è stato fondamentale per la scelta di questo progetto. A questo si deve poi aggiungere la mia passione per la topografia militare che mi ha portato a sceglier un progetto basato su dati geografici.

Si è partiti dapprima con lo studio approfondito del mondo dei database spaziali per poi passare a quello di PosgreSQL, PostGIS e QuantumGIS, programmi che, come vedremo nei prossimi capitoli, sono stati fondamentali per lo sviluppo del progetto.

In secondo luogo si è provato a livello pratico, utilizzando tali programmi, ciò che era stato studiato sugli SDBMS (Spatial DataBase Management System) per poi formulare delle interessanti query per sfruttare al massimo le loro capacità.

Infine si è passati alla programmazione vera e propria in Java cercando di riprodurre tutte quelle operazioni che erano state fatte in precedenza. Quindi si è passati allo sviluppo dell'applicazione vera e propria andando a creare le classi necessarie alla connessione al database, al caricamento dei dati in esso e alla loro visualizzazione, alla creazione dell'interfaccia.

Come quindi si può facilmente capire il nostro progetto ha seguito essenzialmente tre fasi che sono:

- ✓ Studio degli SDBMS e di altri programmi
- ✓ Utilizzo di tali programmi
- ✓ Programmazione in Java

E proprio questa è la struttura che ho voluto dare al mio elaborato. Nella prima parte ho cercato di sintetizzare tutta la fase di studio che è stata fatta. Nella seconda mi sono concentrato nel riassumere le operazioni svolte con PostgreSQL, PostGIS e QuantumGIS. Nella terza ho sintetizzate le operazioni seguite nella programmazione dell'applicazione.

In particolare la tesi è organizzata in 5 capitoli:

- ✓ Capitolo 1: comprende tutta la fase di studio degli SDBMS, dei dati che utilizzano e delle loro potenzialità.
- ✓ Capitolo 2: comprende la fase di studio di PostgreSQL, PostGIS e QuantumGIS e delle loro funzioni.
- ✓ Capitolo 3: comprende la fase progettuale della nostra applicazione riportando use case diagram, activity diagram e class diagram della nostra applicazione.
- ✓ Capitolo 4: vengono riportate tutte le varie fasi che hanno portato all'implementazione della nostra applicazione.
- ✓ Capitolo 5: vengono riportati i codici Java delle varie classi

Capitolo 1

Database spaziali

1.1 Che cosa sono

Un database spaziale (Spatial Database Management System o SDBMS) è un database ottimizzato per memorizzare e interrogare i dati relativi agli oggetti nello spazio, includendo punti, linee e poligoni. A differenza dei tipici database, in grado di analizzare vari tipi di caratteri numerici e di dati, in questo nuovo tipo di database sono necessarie funzionalità aggiuntive, per processare i nuovi tipi di dati, chiamate geometrie o funzionalità. Gli indici che i database tradizionali usano per velocizzare la ricerca di dati e di valori, nel caso dei database spaziali sono chiamati indici spaziali (le tipologie più usate sono gli R-tree). Oltre alle classiche tipologie di query (come ad esempio quelle di SELECT), in questo caso ne sono presenti anche delle altre come, ad esempio, funzioni di misurazione delle distanze, e di calcolo delle aree. Nella letteratura classica, diversi sono stati i nomi che sono stati dati a questi tipi di database come ad esempio pittorici, geometrici, geografici, tutti nomi che evidenziano la loro particolarità fondamentale: il fatto di trattare e memorizzare dati spaziali come mappe, immagini, linee e poligoni. Il termine database spaziale è entrato in uso solo negli ultimi anni in seguito ad un ciclo di conferenze, chiamato "Symposium on Large Spatial Databases (SSD)", a partire dal 1989, e sta proprio ad evidenziare l'idea che tale database sia un contenitore di oggetti nello spazio invece che di immagini o figure di uno spazio.

Ma per quale motivo si è sentita la necessità di creare questa nuova tipologia di database? Ovviamente perché ci si era resi conto di avere qualcos'altro da dover rappresentare che non poteva essere rappresentato con i metodi tradizionali. Ma cosa era necessario rappresentare? Secondo la letteratura tradizionale, abbiamo due tipologie di visioni:

- <u>Oggetti nello spazio</u>: siamo interessati a diverse entità distinte fra loro, ognuna delle quali ha la proprio descrizione geometrica.
- <u>Spazio</u>: vogliamo descrivere lo spazio stesso, che sarebbe a dire essere in grado di descrivere ogni suo punto.

Quando si parla di oggetti nello spazio si intende ad esempio città, foreste, strade.
In questo caso si vanno quindi a considerare come astrazioni le linee, i punti e le regioni (vedi fig. 1.a)

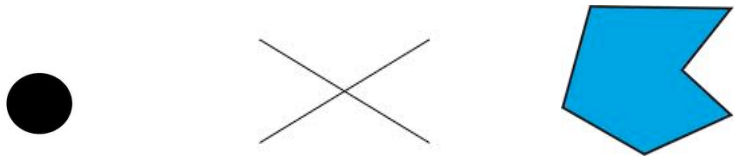


Fig. 1.a: punti, linee, regioni

Mentre la seconda visione è quella della mappe tematiche o delle partizione delle città in distretti. In questo caso le due astrazioni principali saranno la partizione di una regione e le reti (vedi fig. 1.b)

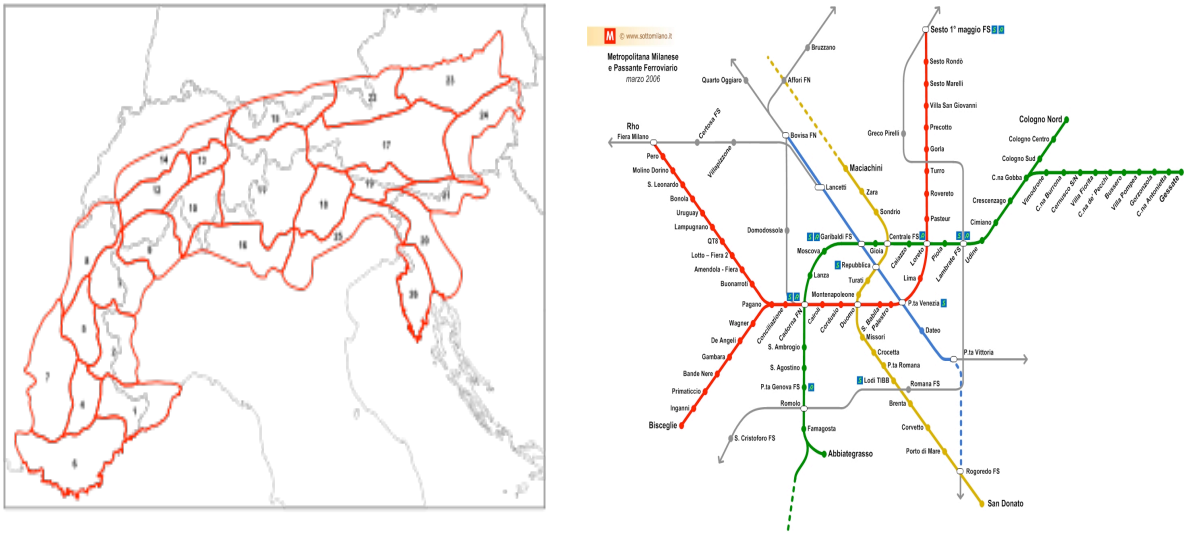


Fig. 1.b: partizione di una regione, reti

E sono proprio queste astrazioni alle fondamenta del SDBMS (Spatial DataBase Management System) usato dal GIS, che andremo meglio ad analizzare nel prossimo capitolo, per cercare, immagazzinare, fare query su dati spaziali.

1.2 Tipi di dati, relazioni e query

Per quanto riguarda i tipi di dati utilizzati ci si rifà alle astrazioni che prima abbiamo accennato. Quindi ancora avremo dei tipi chiamati punti, linee e regioni e attraverso questi è possibile andare a definire le altre operazioni algebriche come l'intersezione o l'unione. Ma tra queste operazioni sicuramente quelle più importanti oltre che più usate sono le *relazioni spaziali*. Possiamo considerare tre classi differenti:

- Le relazioni topologiche: come ad esempio *adiacente, dentro* che non cambiano in base alle trasformazioni topologiche come la rotazione o la traslazione.
- Le relazioni dirette: come ad esempio *sopra, sotto, a nord di, ecc..*
- Le relazioni metriche: *distanza < 300.*

E proprio basandoci su queste relazioni possiamo andare a considerare il concetto di *query* il cui compito principale è quello di connettere le operazioni dell'algebra spaziale con i servizi del linguaggio DBMS. Così possiamo andare a considerare quattro fondamentali tipologie di operazioni:

- Selezione spaziale: una selezione è un'operazione che dato un gruppo di oggetti va a considerare solo quelli che rispettano un determinato predicato. Un esempio può essere quello di andare a considerare tutte le città con più di 300'000 abitanti che distano meno di 100 km da Hagen che viene tradotta con

```
cities select[dist(center, Hagen) < 100 and pop > 300000]
```

- Join spaziale: è un join che confronta degli oggetti attraverso un predicato sui loro valori. In questo caso come esempio possiamo considerare quello di combinare tutte le città con il loro stato:

```
cities states join[center inside area]
```

- Applicazioni su funzioni spaziali: in questo caso abbiamo l'utilizzo di un set di funzioni di algebra spaziali usate all'interno di un'unica query. Ad esempio si chiede tutti i fiumi che passano all'interno della Bavaria, quale parte di essi l'attraversa e quanto è lunga. Tradotto si avrebbe:

```
rivers select[route intersects Bavaria]
extend[intersection(route, Bavaria) {part}]
extend[length(part) {plength}] project[rname, part,
plength]
```

la cui graficazione la possiamo vedere in fig. 1.c

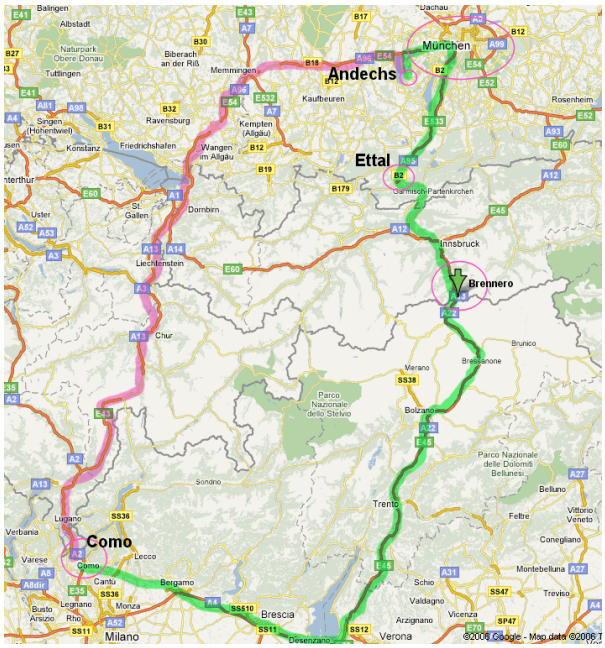


Fig. 1.c: graficazione della precedente query

- Altri set di operazioni: tutte quelle altre operazioni che manipolano dati in altri modi.

Attraverso le query è quindi possibile andare ad interrogare il nostro database ed andare ad effettuare una qualsiasi operazione algebrica. Quello che però non deve sfuggire è che in questo caso si ha un'integrazione della geometria all'interno del linguaggio delle query, integrazione questa che passa attraverso tre aspetti fondamentali che sono quello di considerare i valori di SDT (Spatial Data Type) come costanti all'interno delle query, quello di esprimere le quattro classi di operazioni all'interno dell'algebra spaziale, e infine quello di descrivere la presentazione dei risultati.

1.3 Tool degli SDBMS

Fin ad ora ci siamo limitati a cercare di capire cosa fossero i database spaziali, quali tipologie di dati utilizzassero e su cosa si basava il nuovo linguaggio di query. Adesso invece cerchiamo di capire quali sono gli algoritmi che vengono usati all'interno degli SDBMS.

Il problema generale da risolvere è quello dell'implementazione dell'algebra spaziale, problema questo che passa innanzitutto dalla rappresentazione dei valori e in secondo luogo dalla determinazione di algoritmi per il supporto delle sue operazioni. Nei prossimi paragrafi ci occuperemo di che caratteristiche deve avere la rappresentazione dei valori e degli indici spaziali.

1.3.1 Rappresentazione dei valori

La rappresentazione di un valore di un tipo di dato spaziale deve essere simultaneamente compatibile con due differenti visioni che sono quelle del DBMS e quella dell'algebra spaziale. Per quanto riguarda la prospettiva del DBMS tale rappresentazione deve essere la stessa sia per gli attributi che per tutti gli altri valori, deve avere molteplici e possibili dimensioni, deve poter facilmente essere caricata in vari tipi di memorie, deve essere in grado di offrire un elevato numero di implementazioni di generiche operazioni algebriche di cui necessita il DBMS. Per quanto riguarda la prospettiva dell'algebra spaziale deve essere in grado di supportare algoritmi per le operazioni di algebra spaziale.

1.3.2 Indici spaziali

Un *indice spaziale* è un tipo di indice esteso che consente di indicizzare una colonna spaziale che è una colonna della tabella che contiene dati di tipo spaziale. Gli indici spaziali possono essere creati utilizzando i comuni B tree. Ma cosa sono questi B-tree? Sono strutture dati ad albero comunemente utilizzati nell'ambito di database per accedere a dati alfanumerici conservati in memoria secondaria. Essi derivano dagli alberi di ricerca, in quanto ogni chiave appartenente al sottoalbero sinistro di un

nodo è di valore inferiore rispetto a ogni chiave appartenente ai sottoalberi alla sua destra; derivano anche dagli alberi bilanciati perché tutte le foglie si trovano alla stessa distanza rispetto alla radice. Il vantaggio principale dei B-Tree è che essi mantengono automaticamente i nodi bilanciati permettendo operazioni di ricerca in tempi ammortizzati logaritmicamente. Il vantaggio per cui vengono utilizzati nell'ambito dei database è il fatto che, con la loro implementazione, si ottimizzano i tempi di lettura/scrittura in memoria di massa, in quanto nei B-tree ogni nodo corrisponde ad un blocco di memoria secondaria (detta anche pagina) e quindi il bilanciamento fornisce un limite superiore ai blocchi a cui si deve accedere per raggiungere un dato in memoria di massa. La radice dell'albero è sempre in memoria centrale, mentre gli altri nodi sono in memoria di massa; da questo è facile dedurre che la radice contiene solamente i link agli altri nodi. Un esempio di tale tipologia di albero lo possiamo vedere in fig. 1.d.

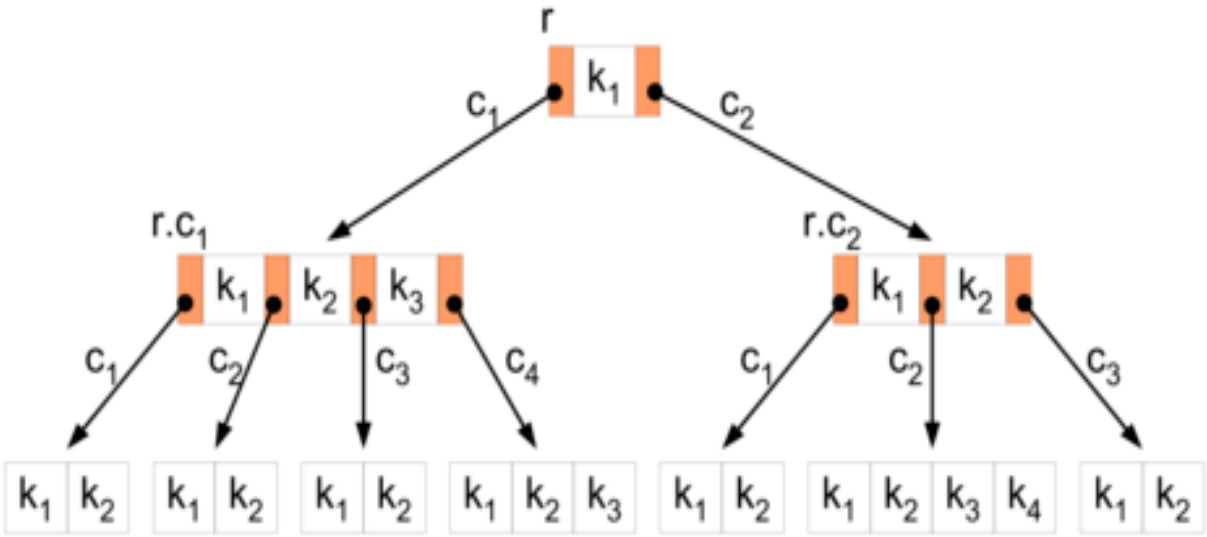


Fig. 1.d: esempio di B-tree

Quindi ogni pagina (che altro non è che una piccola porzione di memoria) del B-tree di un indice viene definita nodo. Il nodo di livello superiore viene definito nodo radice. I nodi di livello inferiore dell'indice vengono definiti nodi foglia. I livelli dell'indice compresi tra il nodo radice e i nodi foglia vengono definiti livelli

intermedi. In un indice cluster il livello foglia include le pagine di dati della tabella sottostante. I nodi di livello radice e intermedio contengono pagine di indice che includono le righe dell'indice. Ogni riga di indice contiene un valore di chiave e un puntatore a una pagina di livello intermedio nel B-tree o a una riga di dati nel livello foglia dell'indice. Le pagine di ogni livello dell'indice sono collegate in un elenco collegato doppiamente.

Questi tipi di alberi vanno bene fin quando si lavora a una o al massimo due dimensioni. Quando però il numero delle dimensioni aumenta allora si passa ad utilizzare gli R-tree che sono un tipo di albero (grafo) simile al B-tree, ma sono usati per indicizzare spazi multidimensionali, ad esempio le coordinate spaziali (X, Y) per dati geografici. Un esempio di richiesta che usi un R-tree potrebbe essere "Trova tutti i musei entro 2 km dalla mia posizione attuale". La struttura dati divide lo spazio in MBR (minimum bounding rectangles, infatti R-tree deriva proprio da Rectangle) annidati gerarchicamente e quando possibile sovrapposti. Ogni nodo dell'R-tree ha un numero variabile di entry (fino ad un massimo predeterminato). Ogni entry che non sia un nodo foglia contiene due entità: una identifica il nodo figlio, l'altra l'MBR che contiene tutte le entry del nodo figlio. L'algoritmo di inserimento e cancellazione di entry dagli MBR assicura che elementi "vicini" siano posizionati nello stesso posto (nodo foglia): un nuovo elemento andrà nel nodo foglia che richiede il minor numero di estensioni delle dimensioni dell'MBR. Gli algoritmi di ricerca usano gli MBR per decidere se cercare o meno nel nodo figlio del nodo corrente. In questo modo la maggior parte dei nodi non viene esplorata dagli algoritmi. Per questo motivo, come per i B-tree, ciò rende gli R-tree adatti ai database, dove i nodi possono essere copiati in memoria solo quando necessario. Un esempio di tale tipologia di albero lo possiamo vedere in fig. 1.e .

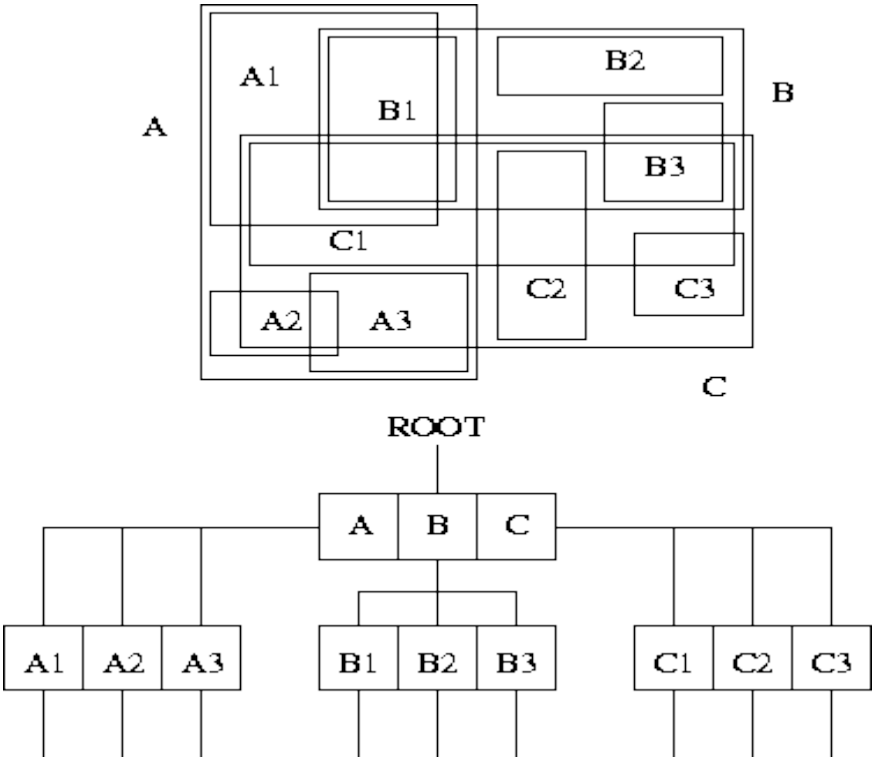


Fig. 1.e: esempio di R-tree

1.4 Requisiti

Bisogna anche evidenziare quali siano i requisiti che il sistema deve avere per poter supportare tale tipologia di database. Un sistema deve almeno essere in grado di recuperare da una grande collezione di oggetti in uno spazio quelli compresi all'interno di una particolare area senza analizzare l'insieme. Quindi l'indicizzazione spaziale è obbligatoria. Essa dovrebbe inoltre supportare la connessione di oggetti con classi diverse attraverso alcune relazioni spaziali in un modo migliore rispetto a quello del prodotto cartesiano. In generale quindi il sistema dovrebbe essere in grado di garantire:

- La rappresentazione dei tipi di dato dell'algebra spaziale
- Le strutture degli indici spaziali
- Le operazioni di accesso a tali indici
- I costi delle funzioni per tutte queste operazioni
- Tecniche di filtro e rifinitura

1.5 Alcune ultime considerazioni su SDBMS

Nelle pagine precedenti si è cercato di evidenziare quali sono le particolarità di questa tipologia di database che diversamente da quelli classici lavora su tipi di dati completamente diversi che possiamo raggruppare in due diverse categorie:

- Geometry: supporta dati planari o euclidei.
- Geography: che archivia dati ellissoidali, ad esempio coordinate di latitudine e longitudine, GPS.

Non più quindi solo parole o numeri ma anche punti, linee e poligoni. Proprio per questa sua caratteristica l'SDBMS viene usato dal GIS che sarà l'argomento del prossimo capitolo.

Capitolo 2

GIS: PostgreSQL, PostGIS, QuantumGIS

2.1 GIS e SDBMS

Prima di cominciare con la vera e propria trattazione di questo capitolo è utile prima soffermarsi su quali siano le particolarità di GIS e SDBMS. Innanzitutto bisogna dire che GIS (Geografical Information System) è un software usato per visualizzare ed analizzare dati spaziali usando funzioni di analisi spaziale come la misurazione, la ricerca, la distribuzione. Un SDBMS al contrario focalizza i suoi obiettivi sull'immagazzinamento efficiente di grandi dati spaziali e sull'utilizzo di indici spaziali per velocizzare le query. Quindi un GIS non fa altro che appoggiarsi a un SDBMS per ricercare, immagazzinare e fare query su dati spaziali. Perciò questi due strumenti potrebbero essere utilizzati da qualcuno che volesse rispondere alla domanda: "quali sono le regioni d'Italia che confinano con più stati stranieri?". Un GIS è differente da un SDBMS, in quanto si occupa essenzialmente dell'elaborazione e manipolazione dei dati georeferenziati, che a loro volta possono essere memorizzati in un SDBMS o in singoli file. Risulta chiaro quindi che un SDBMS può tranquillamente essere usato da altre applicazioni diverse da GIS come ad esempio quelle di astronomia e di geometria. Fissati quindi questi concetti si può iniziare con la trattazione principale del capitolo.

2.2 Introduzione ai GIS

Ci possono essere molte diverse definizioni di Geographic Information System (GIS). Spesso un GIS è definito come un insieme organizzato di hardware, software, dati geografici, e persone tecnicamente preparate per acquisire, memorizzare, aggiornare, manipolare, analizzare, e visualizzare in modo efficiente informazioni che siano geograficamente referenziate. Ma il termine GIS è spesso utilizzato per identificare il software che viene utilizzato per rappresentare ed analizzare cose che succedono sulla terra. Indipendentemente dalla sua definizione, un GIS permette di integrare operazioni comuni di un database quali interrogazioni ed analisi statistiche, con i vantaggi unici offerti dalla possibilità di visualizzazione ed analisi geografica offerti da una mappa. Queste possibilità distinguono un GIS da altri sistemi informativi e lo rendono adatto a vari campi di utilizzo sia nel campo pubblico che privato per spiegare eventi, pianificare strategie. Ma a questo punto ci si chiede: cosa possiamo fare con un GIS? Molti problemi del mondo reale possono essere risolti attraverso GIS, come gli esempi in fig. 2.a:

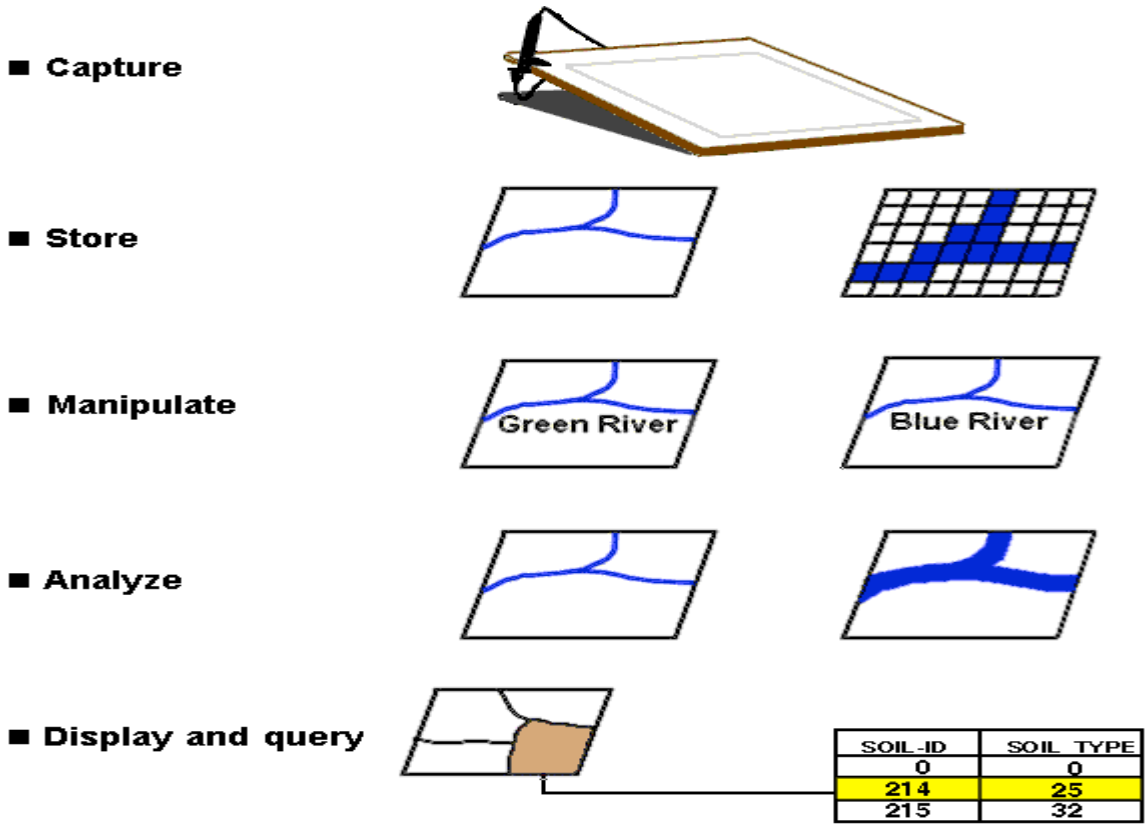


Fig. 2.a: problemi reali risolvibili attraverso GIS

E si potrebbero fare ancora tantissimi esempi. Quello che però e la cosa più interessante è che un GIS permette di visualizzare informazioni geografiche che vengono poi utilizzate ad esempio nei GPS, nelle cartine. Ora ci sono due componenti dell'informazione geografica: i dati spaziali e dati descrittivi (attributi). Su una mappa simboli e testo servono per "comunicare" l'informazione descrittiva. Spesso l'informazione testuale fornisce un modo di accedere all'informazione addizionale organizzata in file correlati. La mappa diventa quindi un potente strumento per distribuire tale informazione. Lo stesso concetto viene applicato ai modelli di dati spaziali. Una potente possibilità offerta da uno strumento GIS è la sua capacità di "legare" informazione spaziale e dati descrittivi (attributi). L'analisi geografica è un processo che aiuta a trovare risposte o soluzioni a particolari problemi geografici. La chiave per un'efficiente analisi geografica è la capacità di analizzare le relazioni geografiche di oggetti che si trovano in livelli informativi separati. Questo viene eseguito unendo spazialmente oggetti geografici da layers diversi utilizzando un processo denominato overlay spaziale. Dopo un overlay spaziale sia gli "attributi" che le "forme" dei dati originali vengono "fusi" in un unico layer creando nuove relazioni spaziali. Una volta terminato è possibile interrogare i nuovi oggetti spaziali così ottenuti utilizzando sia interrogazioni spaziali, sia interrogazioni sugli attributi. Nella fig. 2.b riportiamo un esempio di overlay spaziale.

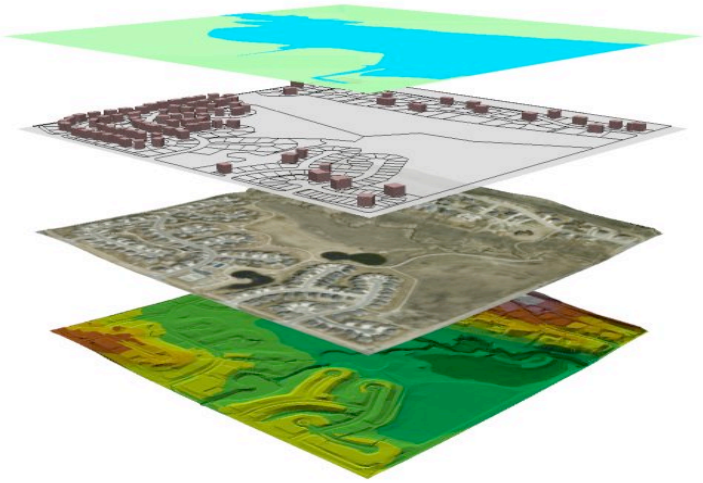


Fig. 2.b: esempio di overlay spaziale

Alla fine quindi possiamo andare a rappresentare la nostra informazione geografica o anche andare solo a rappresentare i dati spaziali o i loro attributi. Entrambi i tipi di dati possono essere visualizzati su uno schermo di un video o inviati su una stampante o plotter. I dati spaziali (punti, linee e poligoni) possono essere memorizzati come layers separati nel database geografico. È possibile visualizzare tutti gli oggetti o solo alcuni di essi. I dati descrittivi (attributi) permettono di comunicare ciò che gli oggetti geografici rappresentano. È possibile produrre elenchi in formati tabellari, utilizzare i dati descrittivi (attributi) per determinare come oggetti specifici saranno visualizzati, o usare l'informazione degli attributi per selezionare e visualizzare specifici oggetti. Scegliendo opportunamente la simbologia, è possibile visualizzare diversi attributi per un singolo layer (ad esempio, la larghezza della linea indica il numero di corsie di una strada e il colore della linea il tipo della strada stessa) come possiamo vedere in fig. 2.c.

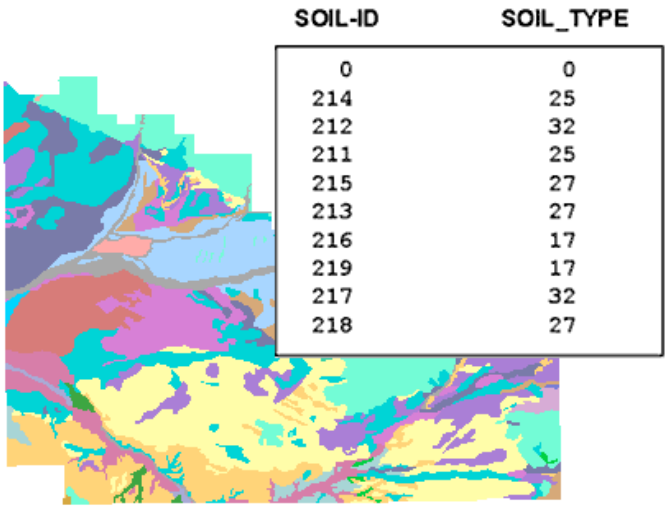


Fig. 2.c: rappresentazione degli attributi di un layer

A questo punto possiamo cominciare a prendere in esame quali sono stati quei programmi fondamentali per la stesura di questo progetto che sono postgreSQL, postGIS e quantumGIS la cui trattazione avverrà nei prossimi paragrafi.

2.3 postgreSQL

L'utilizzo di tale applicazione è stata fondamentale per la stesura di questo lavoro. Infatti è proprio attraverso postgreSQL che è stato possibile creare tutti i database su cui andare caricare (tramite Java) le mappe e su cui andare ad eseguire delle query. I risultati di tali query venivano poi visualizzate all'utente attraverso quantumGIS. Entriamo più nel dettaglio per cercare di capire come funziona postgreSQL.

2.3.1 Cenni storici

PostgreSQL (il cui logo è mostrato in fig. 2.d) , conosciuto anche come Postgres, è il più completo database server open source e fra questi è l'unico a supportare il modello object-relational. È stato sviluppato in open-source fin dall'inizio; l'antenato di PostgreSQL è stato Ingres, sviluppato alla Università di Berkeley in California (dal 1977 al 1985). Michael Stonebraker, anche lui alla UC Berkeley, guidò lo sviluppo di un database object-relational chiamato Postgres (dal 1986 al 1994). Due studenti post-laurea di Berkeley, Jolly Chen e Andrew Yu, aggiunsero più tardi un parser SQL a Postgres. Nell'estate del 1996, in risposta all'esigenza di un database SQL open-source, si formò un gruppo per continuare lo sviluppo di Postgres che sta evolvendo velocemente grazie al supporto di una comunità di sviluppatori.

Fig 2.d: logo di postgreSQL

Il server Postgres è supportato da diversi sistemi operativi sia open-source che proprietari: Linux, i sistemi BSD, Solaris, HP-UX, AIX, Irix e Windows NT/2000.

PostgreSQL è totalmente compatibile con lo standard ANSI-SQL-92, ed implementa la maggior parte dei costrutti SQL. Il database può lavorare in due modalità:

- **fsync:** In questa modalità ogni transazione viene salvata su disco, in modo da garantire i dati da crash di sistema o cadute di corrente. Sacrificando però le performance in fase di inserimento, cancellazione o modifica dei dati.
- **no-fsync:** In questo caso un crash di sistema potrebbe causare una perdita di dati ma le performance sono più elevate.

Esistono due driver ODBC per Postgres, PsqlODBC (incluso nella distribuzione) e OpenLink ODBC oltre ai driver JDBC, di interfacciamento a Java (anche questi compresi nella distribuzione). È dotato anche di un'interfaccia utente grafica, pgaccess, compresa nella distribuzione e di un terminale interattivo Psql.
Ma la caratteristica fondamentale è che è completamente gratuito, anche per usi commerciali e il codice è a disposizione dello sviluppatore.

2.3.2 Descrizione

Un rapido esame di PostgreSQL potrebbe suggerire che sia simile agli altri database. PostgreSQL usa il linguaggio SQL per eseguire delle query sui dati. Questi sono conservati come una serie di tabelle con chiavi esterne che servono a collegare i dati correlati. La programmabilità di PostgreSQL è il suo principale punto di forza ed il principale vantaggio verso i suoi concorrenti: PostgreSQL rende più semplice costruire applicazioni per il mondo reale, utilizzando i dati prelevati dal database. I database SQL conservano dati semplici in "flat table", richiedendo che sia l'utente a prelevare e raggruppare le informazioni correlate utilizzando le query. Questo contrasta con il modo in cui sia le applicazione che gli utenti utilizzano i dati: come ad esempio in un linguaggio di alto livello con tipi di dato complessi dove tutti i dati correlati operano come elementi completi, normalmente definiti oggetti o record (in base al linguaggio). Convertire le informazioni dal mondo SQL a quello della programmazione orientata agli oggetti, presenta difficoltà dovute principalmente al

fatto che i due mondi utilizzano modelli di organizzazione dei dati molto differenti. L'industria chiama questo problema "impedance mismatch" (discrepanza di impedenza): mappare i dati da un modello all'altro può assorbire fino al 40% del tempo di sviluppo di un progetto. Un certo numero di soluzioni di mappatura, normalmente dette "object-relational mapping", possono risolvere il problema, ma tendono ad essere costose e ad avere i loro problemi, causando scarse prestazioni o forzando tutti gli accessi ai dati ad aver luogo attraverso il solo linguaggio che supporta la mappatura stessa. PostgreSQL può risolvere molti di questi problemi direttamente nel database permettendo agli utenti di definire nuovi tipi basati sui normali tipi di dato SQL e al database stesso di comprendere dati complessi. Per esempio, si può definire un indirizzo come un insieme di diverse stringhe di testo per rappresentare il numero civico, la città, ecc. Da qui in poi si possono creare facilmente tabelle che contengono tutti i campi necessari a memorizzare un indirizzo con una sola linea di codice. PostgreSQL, inoltre, permette l'ereditarietà dei tipi, uno dei principali concetti della programmazione orientata agli oggetti. Ad esempio, si può definire un tipo codice_postale, quindi creare un tipo cap (codice di avviamento postale) o un tipo us_zip_code basato su di esso. Gli indirizzi nel database potrebbero quindi accettare entrambi i tipi, e regole specifiche potrebbero validare i dati in entrambi i casi. Nelle prime versioni di PostgreSQL, implementare nuovi tipi richiedeva scrivere estensioni in C e compilarle nel server di database. Dalla versione 7.4 è diventato molto più semplice creare ed usare tipi personalizzati attraverso il comando "CREATE DOMAIN". La programmazione del database stesso può ottenere grandi vantaggi dall'uso delle funzioni. La maggior parte dei sistemi che utilizzano SQL permette agli utenti di scrivere una procedura che è un blocco di codice SQL che le altre istruzioni SQL possono richiamare. Comunque SQL rimane inadatto come linguaggio di programmazione, pertanto gli utenti possono sperimentare grandi difficoltà nel costruire logiche complesse. Ancora peggio, SQL non supporta molti dei principali operatori di base dei linguaggi di programmazione, come le strutture di controllo di ciclo e condizionale. Pertanto ogni venditore ha

scritto le sue estensioni al linguaggio SQL per aggiungere queste caratteristiche, e pertanto queste estensioni non per forza operano su piattaforme diverse di database.

In PostgresSQL, i programmatori possono implementare la logica in uno dei molti linguaggi supportati che sono:

- Un linguaggio nativo chiamato PL/pgSQL simile al linguaggio procedurale di Oracle PL/SQL, che offre particolari vantaggi nelle procedure che fanno un intensivo uso di query.
- Wrapper per i più diffusi linguaggi di scripting come Perl, Python, Tcl, e Ruby che permettono di utilizzare la loro potenza nella manipolazione delle stringhe e nel link ad estese librerie di funzioni esterne.
- Le procedure che richiedono prestazioni maggiori e logiche di programmazione complesse possono utilizzare il C ed il C++.
- Inoltre è disponibile anche un interfacciamento all'esoterico linguaggio R, ricco di funzioni statistiche e per il calcolo matriciale.

Il programmatore può inserire il codice sul server come funzioni, che rendono il codice riutilizzabile come stored procedure, in modo che il codice SQL possa richiamare funzioni scritte in altri linguaggi (come il C o il Perl).

Punti di forza della programmabilità di PostgreSQL sono:

- Incremento delle prestazioni, in quanto la logica viene applicata direttamente dal server di database in una volta, riducendo il passaggio di informazioni tra il client ed il server.
- Incremento dell'affidabilità, dovuto alla centralizzazione del codice di controllo sul server, non dovendo gestire la sincronizzazione della logica tra molteplici client e i dati memorizzati sul server.
- Inserimento di livelli di astrazione dei dati direttamente sul server, per cui il codice del client può essere più snello e semplice.

Questi vantaggi fanno di PostgreSQL, probabilmente, il più avanzato sistema database dal punto di vista della programmabilità, il che aiuta a spiegarne il successo.

Utilizzare PostgreSQL può ridurre il tempo totale di programmazione di molti progetti, con i vantaggi suddetti che crescono con la complessità del progetto stesso, ovviamente se utilizzato da utenti esperti nel suo utilizzo.

2.3.3 Tipi di dato

Fondamentalmente PostgreSQL gestisce quattro tipologie di dato: le stringhe, i valori numerici, le date e i valori booleani o logici. La capacità di distinguere tra tipi diversi è particolarmente importante, dato che consente di ottimizzare l'allocazione dei dati risparmiando spazio e velocizzando le operazioni di elaborazione. I tipi di dato numerici si riferiscono a due principali sottocategorie: i numeri interi e i numeri decimali, questi ultimi sono detti anche numeri in "virgola mobile". Nello specifico, riportiamo alcune tipologie di dato intero:

- INTEGER: numero intero costituito al massimo da 9 cifre.
- SMALLINT: numero intero per definizione più piccolo di quello esprimibile tramite il tipo INTEGER.

Per quanto riguarda invece i decimali elenchiamo:

- FLOAT: espresso singolarmente indica un numero in virgola mobile; FLOAT(n): indica invece un numero in virgola mobile della lunghezza di n bit.
- REAL: è anch'esso un numero in virgola mobile ma espresso in teoria con maggiore precisione rispetto a FLOAT.
- DOUBLE PRECISION: dei REAL con però il doppio della precisione

I tipi di dato stringa si riferiscono a sequenze di caratteri: per stringa infatti si intende un insieme di caratteri composto da numeri che vanno da 0 a 9, da lettere dell'alfabeto, da spazi vuoti, punteggiatura, simboli e caratteri speciali. Nello stesso modo un'immagine, un video o un Mp3 sono delle sequenze di caratteri che restituiscono un output multimediale. Riportiamo alcuni esempi di dato stringa:

- CHAR: indica un singolo carattere; CHAR(n) indica invece una stringa della lunghezza fissa di n caratteri.

- VARCHAR(n): indica una stringa di lunghezza variabile composta al massimo da n caratteri.

Le date vengono viste da PostgreSQL come se fossero un tipo a se stante, non un numero né una stringa; ciò consente di operare calcoli cronologici con un alto livello di precisione, quasi come se l'ORDBMS (Object-Relational DataBase Management system) avesse "coscienza" del tempo che passa. Per quanto riguarda questa tipologia segnaliamo:

- DATE: indica una data intera completa di giorno, mese ed anno.
- TIMESTAMP: indica un'informazione contenente data e ora.
- TIME: indica un orario completo di ore, minuti e secondi.
- INTERVAL: indica un intervallo di tempo.

I valori booleani rispondono alla logica binaria 0/1 oppure TRUE/FALSE, da cui abbiamo il valore BOOLEAN che può assumere soltanto uno tra questi due valori. La regola fondamentale nella rappresentazione del tipo di dato in PostgreSQL è che essi vanno rappresentati in forma costante e gli unici delimitatori ammessi sono gli apici singoli.

I tipi di dato numerici vanno rappresentati in questo modo: `numero'. Per cui avremo, per esempio, `1000', nel caso di un INTEGER (oppure `-1000' se negativo); nello stesso modo avremo, per esempio `- 58.9' nel caso di un numero in virgola mobile, sia esso FLOAT o REAL.

Diversa la modalità di rappresentazione utilizzata per le stringhe: `' stringa ". Per cui avremo ``albero" o `'} ☻ ôqÕ" identificati come stringhe; la modalità non cambia al variare del tipo stringa, quindi non avremo differenze tra CHAR e VARCHAR. Anche le date hanno una propria sintassi rappresentativa: : `' data ". Per cui avremo ``22.11.2005" nel caso di una data (dato DATE), ``17:57:51" nel caso di un orario (dato TIME) oppure, ``22.11.2005 17:57:51" nel caso di un TIMESTAMP.

2.4 postGIS

PostGIS (il cui logo e rappresentato in fig. 2.e) è l'estensione spaziale del server PostgreSQL che introduce i tipi di dato geometrico e le funzioni per lavorare con essi. Fornisce i tipi di dati specificati negli standard dell'Open Geospatial Consortium. In particolare è un geodatabase e fornisce il sistema di gestione dati sui quali è basato un GIS. Consente di poter archiviare su database anche dati geografici e di eseguire operazioni su di essi.

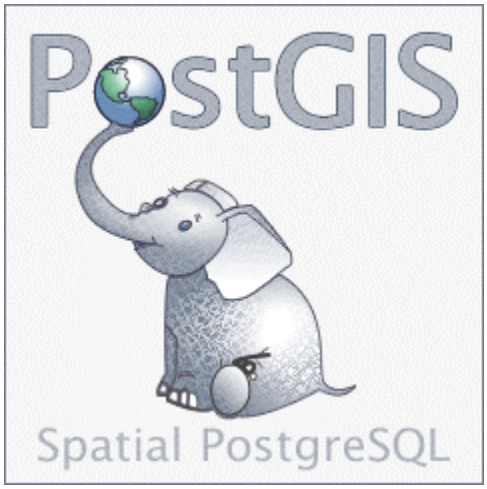


Fig. 2.e: logo di postGIS

Le geometrie che si possono manipolare ed usare grazie a PostGis sono le seguenti: point, line, polygon, multipoint, multiline, multipolygon e geometry collections. Tra le sue caratteristiche più importanti possiamo elencare:

- È disponibile per Windows e sistemi GNU/Linux.
- Permette l'archiviazione di dati vettoriali con coordinate 2D e 3D.
- Consente la gestione di circa 200 sistemi di riferimento.
- Consente l'esportazione dati in SVG e GML.

Molto interessanti sono anche le funzioni di postGIS oltre che l'analisi di alcune tra le più importanti operazioni che è possibile eseguire che sono l'importazione e l'esportazione di shapefile (.shp) di cui si parlerà però nel capitolo 4.

2.5 QuantumGIS

QuantumGIS (logo in fig. 2.f) è stato molto importante per lo sviluppo della tesi. Attraverso questo programma è stato possibile infatti andare a visualizzare i risultati delle query che andavamo a fare sui database precedentemente caricati attraverso postgreSQL. Inoltre permette di eseguire un elevatissimo numero di query attraverso i suoi tool (come ad esempio caricamento su una mappa di un numero N di punti, o ricerca di regioni che avevano determinate caratteristiche, ecc.).

Fig. 2.f: logo di quantumGIS

QuantumGIS (QGIS) è un Sistema Informativo Geografico a codice aperto (Open Source). Il progetto è nato nel maggio 2002 ed è stato ospitato su SourceForge nel giugno dello stesso anno. QGIS gira attualmente su molte piattaforme Unix, su Windows, e OSX. QGIS è sviluppato in Qt e C++. Ciò fa sì che QGis appaia comodo nell'uso e piacevole e semplice da usare nell'interfaccia grafica (graphical user interface) GUI. L'obiettivo iniziale era di fornire un visualizzatore di dati GIS, ma attualmente QGIS ha oltrepassato questo punto nel suo sviluppo, ed è usato da molti per il loro lavoro quotidiano nel campo GIS. QGIS supporta nativamente un considerevole numero di formati raster e vettoriali (di questi due tipi di dato si parlerà nel paragrafo 2.5.2): il supporto a nuovi formati può essere facilmente inserito mediante plugin.

2.5.1 Descrizione

QGIS offre nativamente e mediante plugin molte funzioni GIS di uso comune. È possibile offrirne una panoramica raggruppandole sinteticamente in sei categorie:

- Visualizzazione di dati: possono essere visualizzati e sovrapposti dati vettoriali e raster in diversi formati e proiezioni senza necessità di conversioni verso un formato comune interno. Tra i formati supportati sono inclusi tra gli altri shapefile ESRI, MapInfo, SDTS, GML, dati spaziali forniti da servizi di mappa online conformi agli standard OGC (Open Geospatial Consortium) quali Web Map.
- Esplorazione dati e creazione di mappa: possono essere composte mappe e esplorati interattivamente dati spaziali con una semplice interfaccia grafica. I molti strumenti utili disponibili nell'interfaccia includono: proiezione al volo, compositore di mappa, pannello vista panoramica, etichetta elementi segnalibri geospaziali, modifica/vedi/cerca elementi, salva e ricarica progetti.
- Creazione, modifica, gestione ed esportazione di dati: possono essere creati, modificati, gestiti ed esportati dati vettoriali in molteplici formati. I dati raster devono essere importati in GRASS (Geographic Resources Analysis Support System) per poter essere editati ed esportati in altri formati. QGIS offre le seguenti funzioni: strumenti per digitalizzare formati supportati da OGR e layer vettoriali GRASS, creazione e modifica di shapefiles e georeferenziazione di immagini con l'apposito plugin.
- Analisi di dati: Possono essere eseguite analisi spaziali di dati PostgreSQL/PostGIS e di altri formati supportati da OGR per mezzo del plugin python ftools. QGIS offre attualmente strumenti per l'analisi, il campionamento, il geoprocessamento, la gestione delle geometrie e del database di dati vettoriali. Possono inoltre essere usati gli strumenti GRASS integrati, che includono l'intera gamma delle funzioni di GRASS di oltre 300 moduli.

- Pubblicazione di mappe su Internet: QGIS può essere usato per esportare dati in un mapfile che può essere pubblicato su Internet mediante un webserver sul quale sia installato UMN MapServer.
- Estensione delle funzioni di QGIS per mezzo di plugin (un programma non autonomo che interagisce con un altro programma per ampliarne le funzioni): QGIS può essere adattato a particolari esigenze grazie all'architettura estensibile per mezzo di plugin. QGIS fornisce le librerie che possono essere usate per creare i plugin. Fondamentalmente abbiamo due tipi di plugin che sono quelli inclusi nel software e quelli Python. Questi ultimi sono ospitati sul repository ufficiale PyQGIS, e possono essere facilmente installati usando il gestore di plugin python. Mentre tra i plugin inclusi nel software (la cui attivazione avviene attraverso il plugin manager mostrato in fig. 2.g) abbiamo ad esempio la cattura delle coordinate, l'aggiunta layer WFS, le decorazioni, gli strumenti GPS, il generatore di reticolo, il plugin per interpolazione, esportazione verso Mapserver, e altri.

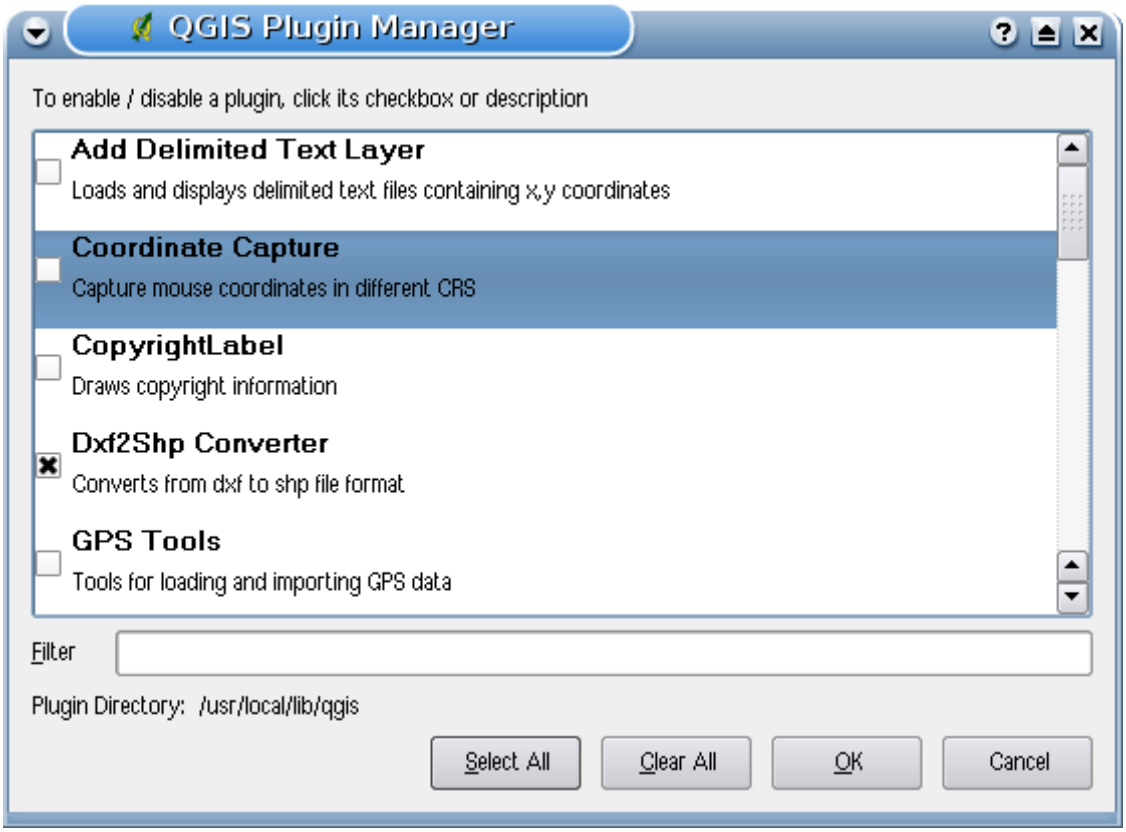


Fig 2.g: plugin manager

2.5.2 Tipi di dato

QuantumGIS è un programma che fondamentalmente viene utilizzato per creare, visualizzare, interrogare e analizzare dati geospaziali. I dati geospaziali riportano informazioni inerenti la posizione geografica di un oggetto che spesso implica l'uso di coordinate geografiche, quali valori di latitudine e longitudine. Dato spaziale è un altro termine comunemente usato, così come lo sono: dato geografico, dato GIS, mappa, location, coordinate e geometrie spaziali. Le applicazioni che usano dati geospaziali eseguono varie funzioni. Quella più conosciuta e facilmente compresa è la produzione di mappe. I programmi per la realizzazione di mappe impiegano tali dati e li rappresentano in una forma che sia visibile, tipicamente su uno schermo o stampati su carta. Si possono presentare mappe statiche (una semplice immagine) o mappe dinamiche che sono personalizzate dall'utente che ne usufruisce attraverso un'applicazione desktop o una pagina web. Ma a questo punto ci chiediamo come sono archiviati i dati geospaziali? In estrema sintesi, ci sono due tipi di dati geospaziali attualmente di uso comune (in aggiunta al tradizionale dato in forma tabellare che è comunque diffusamente impiegata) e sono i dati raster e i dati vettoriali.

Un dato raster è una griglia regolare fatta di celle o nel caso di semplici immagini, di pixel. Essi hanno un numero fisso di righe e colonne. Ogni cella ha un valore numerico e una certa dimensione geografica (ad es. 30x30 metri). Più raster sovrapposti sono utilizzati per rappresentare immagini che utilizzano più di un colore (ad esempio un raster per ogni set di rosso, verde e blu viene combinato per creare il colore dell'immagine). I più noti dati raster sono l'immagine satellitare, la foto aerea, le ombreggiature altimetriche o i modelli digitali di terreno (Digital Elevation Model, DEM). Qualunque elemento di mappa, con le opportune limitazioni, può essere rappresentato come dato raster. Inoltre le immagini satellitari sono anche un esempio di dati in bande multiple. Ogni banda è essenzialmente un livello sovrapposto al precedente dove vengono salvati i valori della lunghezza della luce. Come è facile

immaginare, un raster di grosse dimensioni occupa maggiore spazio su disco. Un raster con celle piccole può fornire maggior dettaglio ma richiede anche più spazio. Per questo si cerca di trovare il giusto equilibrio tra la dimensione della cella ai fini dell'archiviazione e la dimensione della cella ai fini analitici o di mappatura. Infine possiamo dire che diversamente dai dati vettoriali, i dati raster di solito non hanno associato un database contenente i dati descrittivi di ogni cella e sono geocodificati in base alla risoluzione del pixel e alle coordinate x/y di un angolo del raster che ne consente un corretto posizionamento nella vista mappa. QGIS utilizza le informazioni di georeferenziazione incorporate nel layer raster (ad es. GeoTiff) o in un apposito world file per visualizzare correttamente i dati.

L'altra tipologia di dati sono i dati vettoriali. Nel suo senso più semplice, i vettori sono un metodo di descrizione di una posizione utilizzando un insieme di coordinate. Ogni coordinata si riferisce ad una posizione geografica utilizzando un sistema di valori xy (piano cartesiano). I vettori vengono usati per diagrammare decrescenti risparmi per la pensione o crescenti interessi per il mutuo, ma gli stessi concetti sono alla base dell'analisi e della mappatura del dato geospaziale. Ci sono vari modi di rappresentare queste coordinate geografiche secondo lo scopo che ci si è prefissi. I dati vettoriali si presentano in tre forme, di complessità crescente:

- Punti: una coppia di coordinate (x y) rappresenta una precisa posizione geografica.
- Linee: Coordinate multiple (x1 y1, x2 y2, x3 y4, ... xn yn) collegate tra loro in un certo ordine. Equivale a disegnare una linea dal punto (x1 y1) al punto (x2 y2) e così via. Queste parti fra ogni punto sono considerate segmenti. Hanno una lunghezza ed ad essi si può attribuire una direzione basata sull'ordine dei punti. Tecnicamente, una linea è data da una singola coppia di coordinate collegate insieme; una polilinea è costituita da linee multiple collegate insieme.
- Poligoni: Quando le linee sono collegate tra loro da più di due punti, con l'ultimo punto coincidente con il primo, denominiamo questa un poligono.

Triangoli, cerchi, rettangoli ecc. sono tutti poligoni. La caratteristica principale di un poligono è che essi racchiudono un'area.

2.5.3 Plugin

Quantum GIS è stato progettato con un'architettura a plugin. Ciò permette di aggiungere nuove caratteristiche e funzioni all'applicazione. Molte delle caratteristiche in QGIS sono in effetti implementate come plugin di base Core o contribuiti dagli utenti Esterni:

- Plugin Core (nella barra delle applicazioni in fig. 2.f) sono mantenuti dal team di sviluppo di QGIS e fanno parte automaticamente di ogni distribuzione di QGIS. Sono scritti in uno dei due linguaggi: C++ or Python.
- Plugin esterni sono attualmente tutti scritti in Python. Sono immagazzinati in archivi dei pacchetti esterni e mantenuti dai singoli autori. Possono essere aggiunti a QGIS usando il plugin core chiamato Plugin Installer.

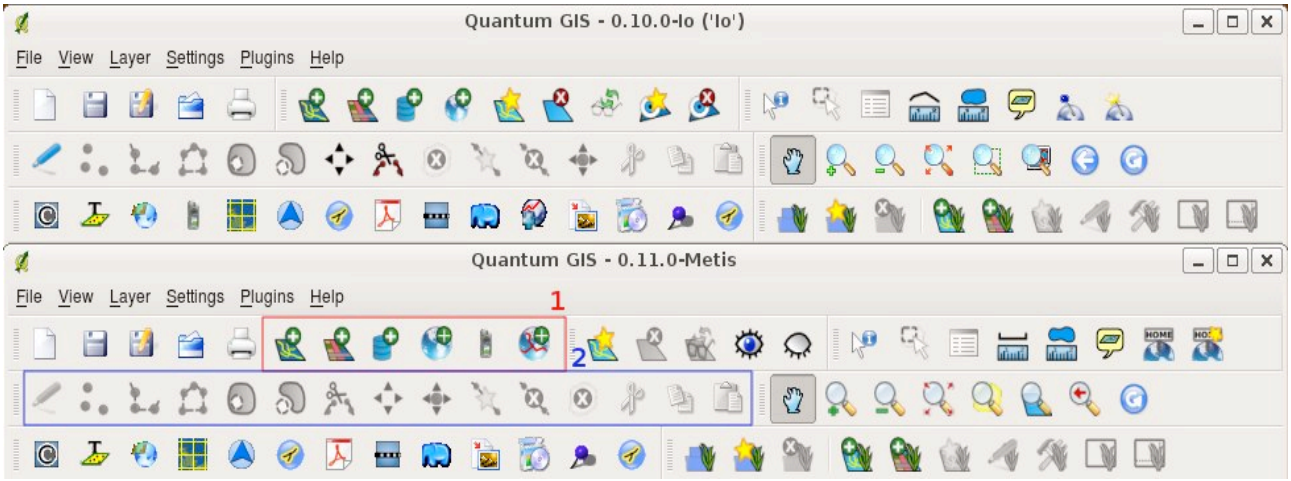


Fig. 2.h: barra dei plugins

Il plugin che sicuramente è servito maggiormente per questo progetto è senza dubbio il plugin ftool. L'obiettivo del plugin python ftool è quello di fornire una risorsa per

molti processi basati sui GIS, senza bisogno di software, librerie, o soluzioni complesse aggiuntive. Attraverso questo plugin è possibile effettuare la gestione di dati spaziali e l'analisi di funzioni in maniera sia veloce che funzionale e fornisce funzioni che vanno dai tool di analisi e ricerca a quelli di geometria e geoprocessing. ftool ora è automaticamente installato e attivato nelle nuove versioni di QGIS, e come con tutti i plugin, può essere disattivato e attivato con il Plugin Manager.

Capitolo 3

Progetto

3.1 Alcune considerazioni: dove ci troviamo

Fino ad ora ci siamo limitati ad analizzare quali sono stati gli elementi teorici che era necessario possedere per la realizzazione del nostro programma (nel capitolo 1) e quali sono state le tecnologie usate (i programmi che sono stati descritti nel capitolo 2). Ci proponiamo ora di andare ad analizzare la fase di progettazione e di implementazione vera e propria della nostra applicazione.

La progettazione del programma ha seguito varie fasi. Principalmente si è partiti dall'analisi di possibili utilizzatori del programma e i suoi possibili usi. Poi è stata fatta un'analisi delle classi che servivano alla realizzazione del programma stesso e alle relazioni fra di esse (mediante class diagram). Tale fase è stata infine seguita dall'implementazione vera e propria del nostro programma. In questo capitolo andiamo a considerare la fase di progettazione (attraverso le fasi prima citate) mentre nel capitolo successivo analizzeremo la fase di implementazione dell'applicazione.

3.2 Possibili utilizzatori del programma

Giunti a questo punto della nostra trattazione ci chiediamo: quali sono i possibili utilizzatori del nostro programma? E i suoi possibili usi?

Questo programma è rivolto a tutti coloro che utilizzano un GPS (Global Position System). Permette infatti di andare a caricare all'interno di un database spaziale degli shapefile .shp (un esempio è rappresentato in fig. 3.a) contenenti mappe, strade, fiumi, punti di interesse che poi possono essere visualizzati direttamente attraverso l'interfaccia del GPS stesso. Ovviamente questa operazione può essere fatta sia da chi programma il database del GPS e sia dall'utilizzatore. Il programmatore infatti inserisce tutti quei dati che poi l'utente andrà a visualizzare, invece l'utilizzatore può aggiungere a questi dati degli altri che secondo lui possono essere di particolare interesse (come ad esempio dei ristoranti in cui si è trovato bene, o dei luoghi belli da vedere) che magari il programmatore può usare successivamente.

Fig. 3.a: esempio di shapefile rappresentante una mappa delle autostrade italiane

Da un punto di vista spiccatamente militare tale programma può invece essere usato nei vari teatri operativi all'estero oltre che in patria. Molto spesso infatti nasce l'esigenza di conoscere quali sono le posizioni di basi alleate (fig. 3.b), checkpoint (dei punti sensibili alla cui protezione vengono posti dei militari), posti di blocco, ecc. Tale programma può anche essere utilizzato nella scelta degli itinerari per conoscere quali sono i punti critici da oltrepassare, strade, ponti, acquedotti, ecc.. Come quindi nel campo civile anche nel campo militare tale programma può essere utilizzato sia da un punto di vista prettamente strategico - organizzativo che da un punto di vista tattico. Può infatti inserire dati all'interno del database sia il comando, per segnalare ai reparti schierati quali sono gli itinerari da seguire o i punti critici da controllare, e sia la pattuglia schierata per segnalare al comando eventuali variazioni dello scenario iniziali o conseguimento dell'obiettivo (eliminazione di un bersaglio, ecc.).

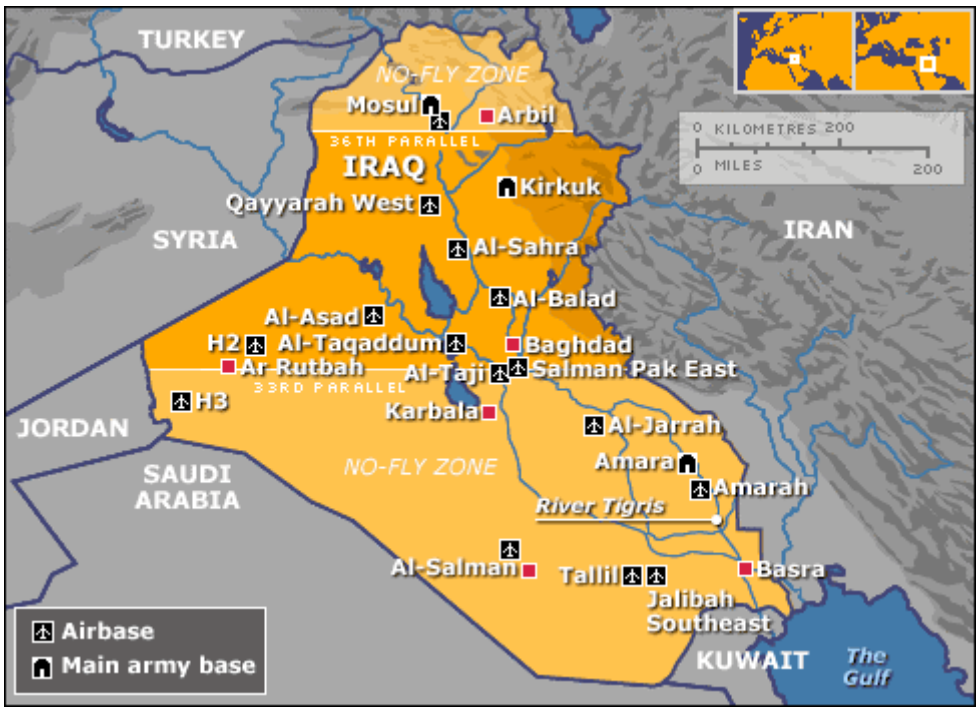


Fig. 3.b: disposizione delle basi militari in Iraq

Passiamo ora ad analizzare degli use case e degli activity diagram della nostra applicazione.

3.3 Use case e activity diagram

In questo paragrafo andremo ad analizzare dapprima uno dei possibili casi d'uso che si potrebbero verificare nell'utilizzo della nostra applicazione per poi andarne ad analizzare le relative tabelle.

3.3.1 Scenari

Uno dei possibili scenari (fig. 3.c) è quello in cui un programmatore utilizzi l'interfaccia grafica del programma per visualizzare dei dati già esistenti o per caricarne di nuovi.

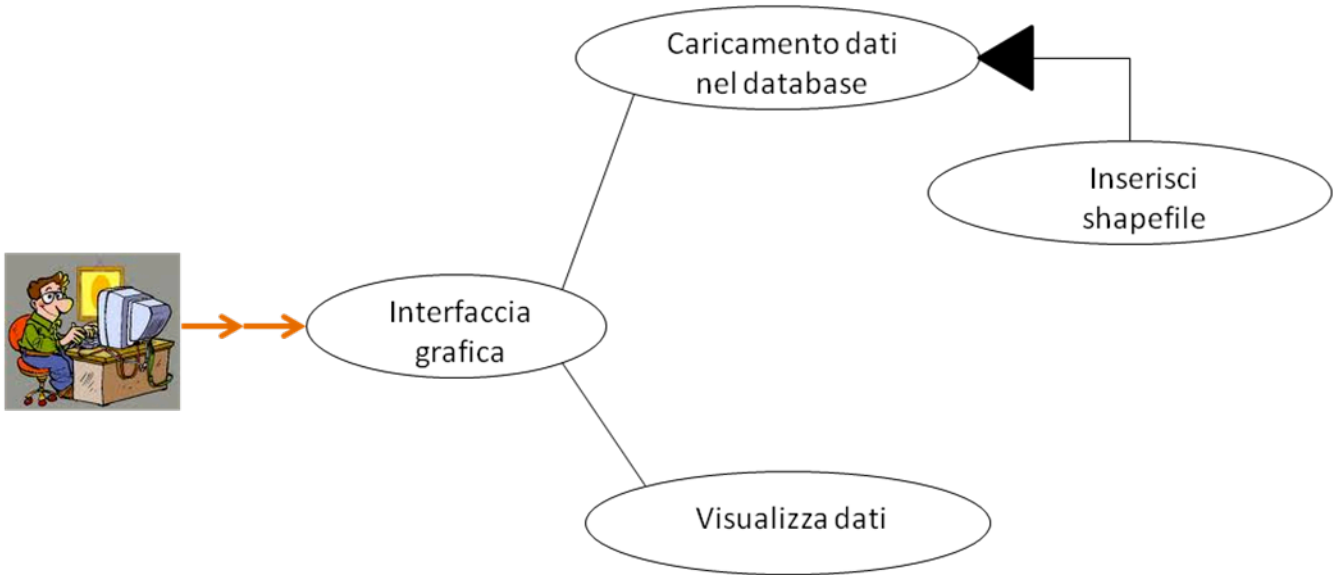


Fig. 3.c: scenario dell'utilizzo dell'interfaccia grafica
da parte di un programmatore

Un altro scenario, dal punto di vista grafico praticamente uguale a quello del programmatore, è quello di un soldato che in operazione utilizza l'interfaccia grafica della nostra applicazione per visualizzare dei dati già caricati o per aggiungerne di nuovi in un database già fornito dal programmatore (in fig. 3.d):

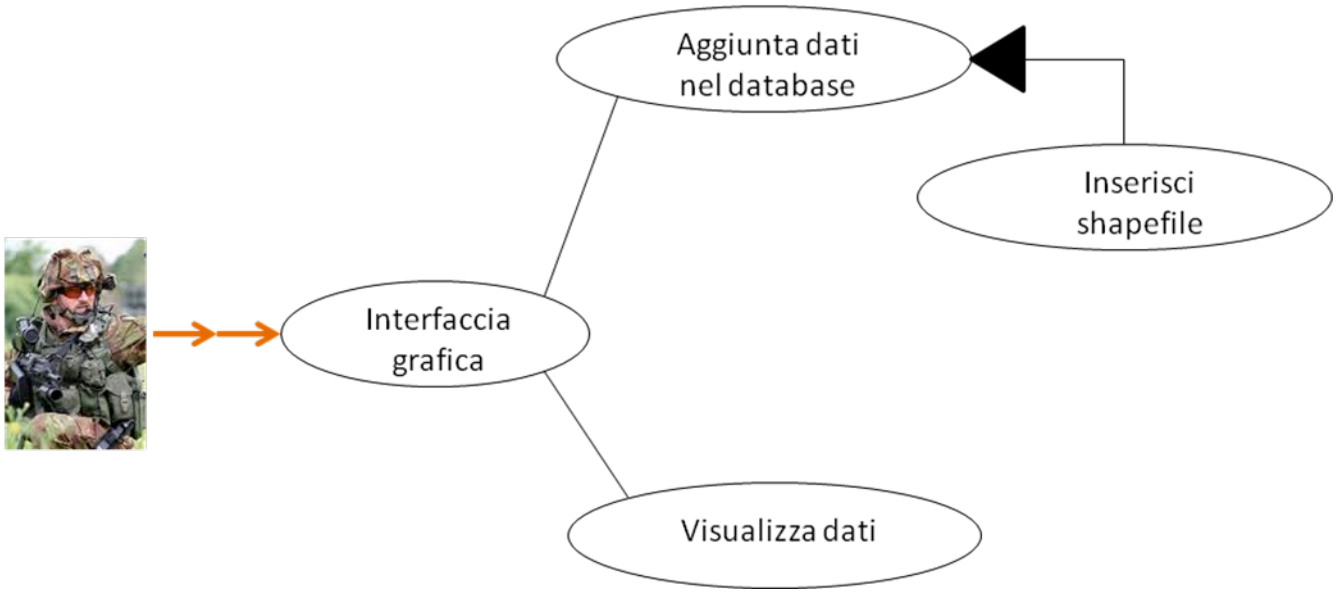


Fig. 3.d: scenario dell'utilizzo dell'interfaccia grafica
da parte di un utente

Questo proprio a sottolineare la doppia utilizzazione della nostra applicazione che può appunto essere impiegato sia dai programmatori per la creazione del database, sia dagli utilizzatori (un soldato ad esempio) per l'inserimento di dati nel GPS.

3.3.2 Tabelle dei casi d'uso

Vediamo alcune tabelle di alcuni casi d'uso.

- Caricamento di dati all'interno del database da parte dell'utente:

Obiettivo	Attraverso questa funzione è possibile andare a inserire degli shapefile all'interno del database spaziale
Attore primario	Utente
Ambito	Software in questione
Precondizioni	Il database deve esistere, avere a disposizione ancora memoria ed essere stato già precaricato
Evento scatenante	L'utente decide di inserire nel database dei dati
Sequenza	1) Clicca sul tasto 'user' 2) Clicca sul tasto 'sfoglia' 3) Scegliere tra le varie cartelle il file che si desidera caricare nel database e una volta scelto cliccare sul relativo file che verrà evidenziato dal sistema 4) Cliccare sul tasto 'apri'

Postcondizioni di successo	Il sistema fa comparire a video il seguente messaggio di conferma:
	
	`Shapefile type: Point`
	`Postgis type: POINT[2]`
	
	memorizza i nuovi dati nel database e inserisce il percorso del file nella tabella di visualizzazione file

Postcondizioni di fallimento	viene visualizzato a video un messaggio di errore e le informazioni non sono caricate

- Caricamento di dati all'interno del database da parte del programmatore:

Obiettivo	Attraverso questa funzione è possibile andare a inserire deglii shapefile all'interno del database spaziale
Attore primario	Programmatore
Ambito	Software in questione
Precondizioni	Il database deve esistere e avere a disposizione ancora memoria
Evento scatenante	Il programmatore decide di inserire nel database dei dati
Sequenza	1) Il programmatore si identifica attraverso nome utente e password e clicca sul tasto 'ok' 2) Clicca sul tasto 'crea' e all'apertura della nuova schermata sul tasto 'sfoglia' 3) Scegliere tra le varie cartelle il file che si desidera caricare nel database e una volta scelto cliccare sul relativo file che verrà evidenziato dal sistema 4) Cliccare sul tasto 'apri'

Postcondizioni di successo	Il sistema fa comparire a video il seguente messaggio di conferma:
	
	`Shapefile type: Point`
	`Postgis type: POINT[2]`
	
	memorizza i nuovi dati nel database e inserisce il percorso del file nella tabella di visualizzazione file

Postcondizioni di fallimento	viene visualizzato a video un messaggio di errore e le informazioni non sono caricate

- Visualizzazione dei dati da parte dell'utente:

Obiettivo	Attraverso questa funzione è possibile andare a visualizzare i dati presenti all'interno del database spaziale
Attore primario	Utente
Ambito	Software in questione
Precondizioni	Il database deve esistere e contenere dei dati
Evento scatenante	L'utente decide di voler visualizzare i dati presenti nel database
Sequenza	1) Clicca sul tasto 'user' 2) Cliccare sul tasto 'Visualizza dati' 3) Il sistema visualizza a video una tabella con i file caricati nel database 4) Dopo che l'utente clicca sul tasto di chiusura il sistema chiude l'applicazione
Postcondizioni di successo	Le informazioni richieste vengono visualizzate
Postcondizioni di fallimento	Viene visualizzato a video un messaggio di errore

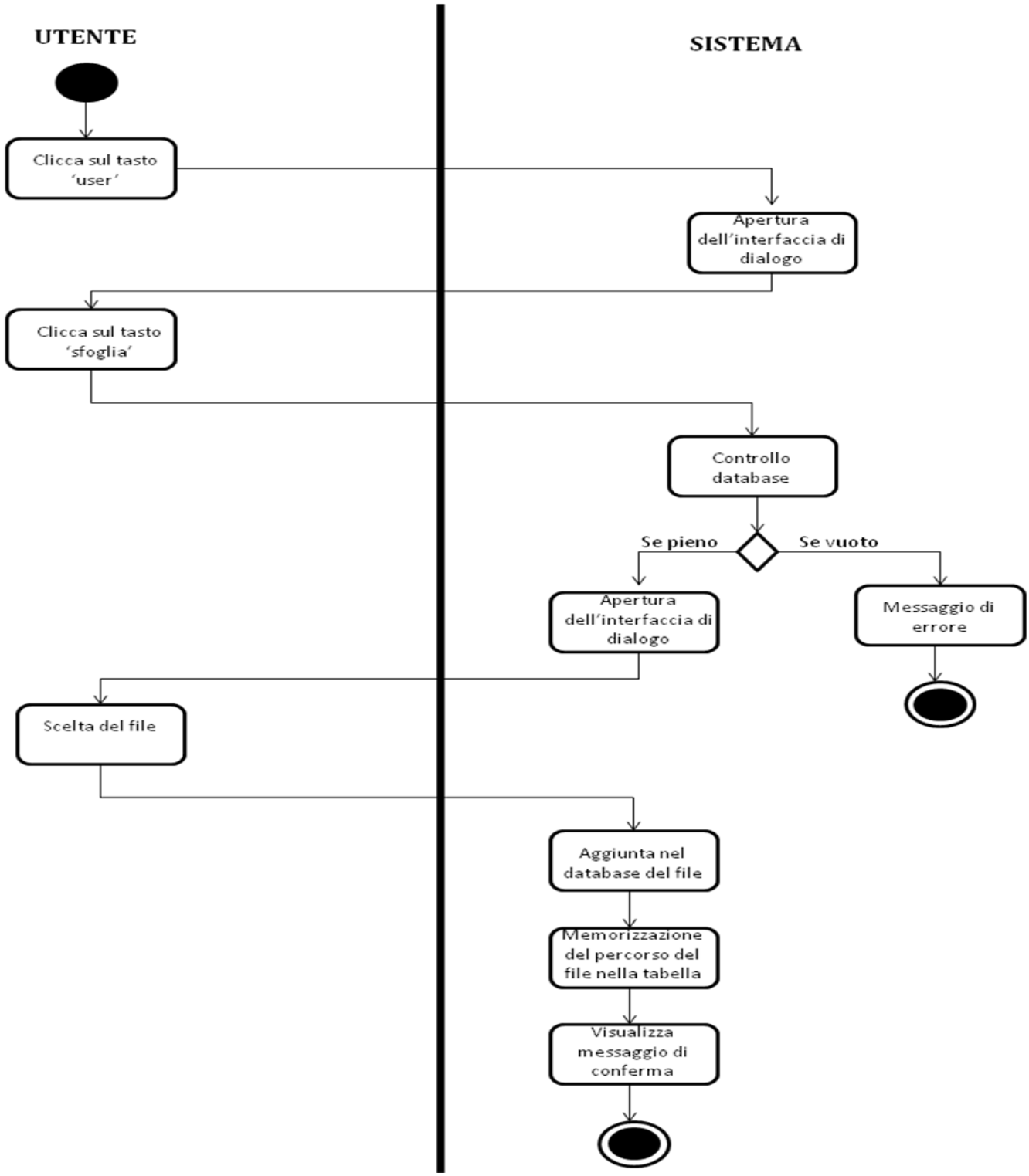
- Visualizzazione dei dati da parte del programmatore:

Obiettivo	Attraverso questa funzione è possibile andare a visualizzare i dati presenti all'interno del database spaziale
Attore primario	Programmatore
Ambito	Software in questione
Precondizioni	Il database deve esistere e contenere dei dati
Evento scatenante	Il programmatore decide di voler visualizzare i dati presenti nel database
Sequenza	1) Il programmatore si identifica attraverso nome utente e password e clicca sul tasto 'ok' 2) Cliccare sul tasto 'Visualizza dati' 3) Il sistema visualizza a video una tabella con i file caricati nel database 4) Dopo che il programmatore clicca sul tasto di chiusura il sistema chiude l'applicazione
Postcondizioni di successo	Le informazioni richieste vengono visualizzate
Postcondizioni di fallimento	Viene visualizzato a video un messaggio di errore

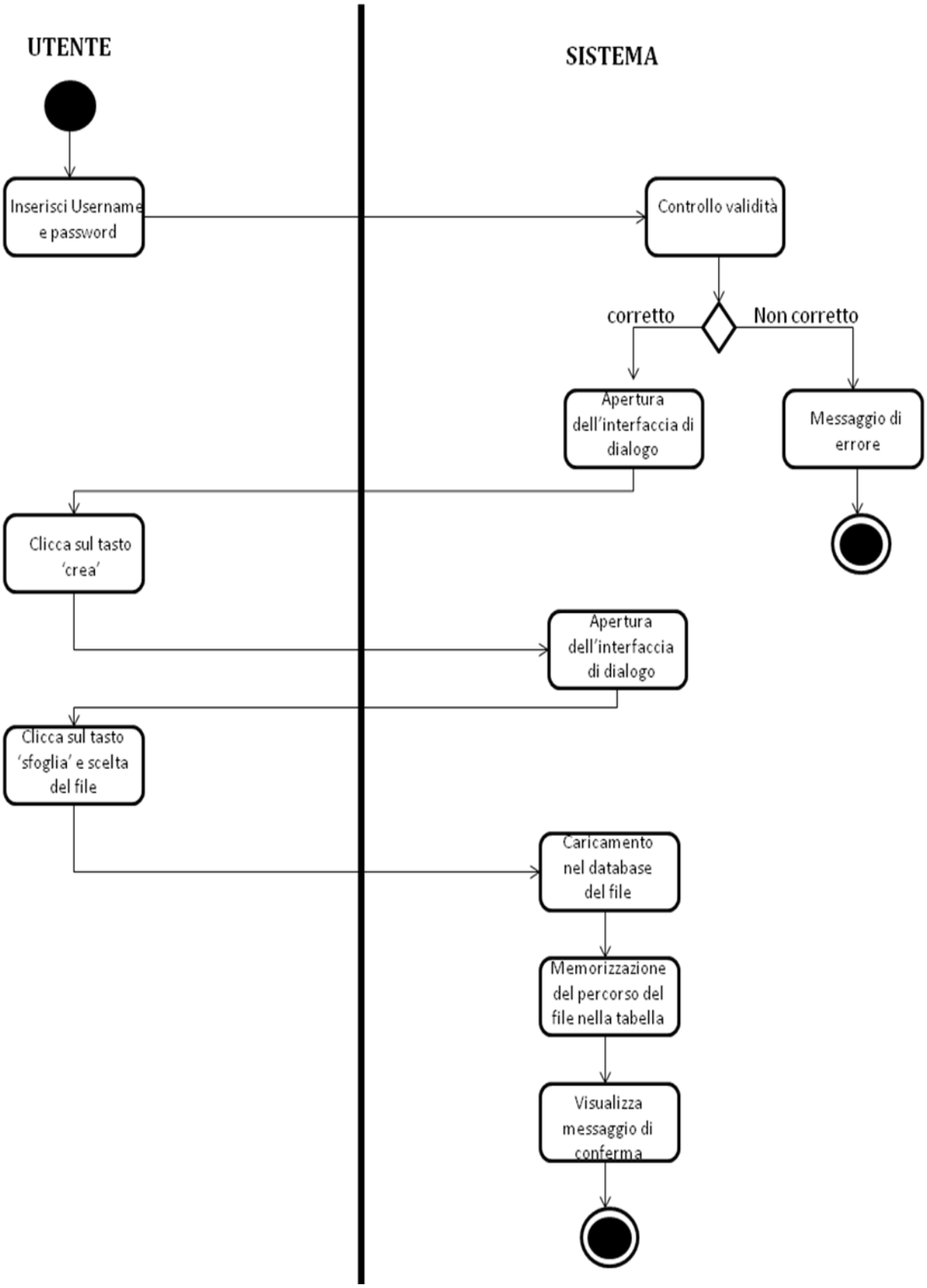
3.3.3 Activity diagram

Riportiamo di seguito gli activity diagram relativi alle tabelle dei casi d'uso sopra riportate:

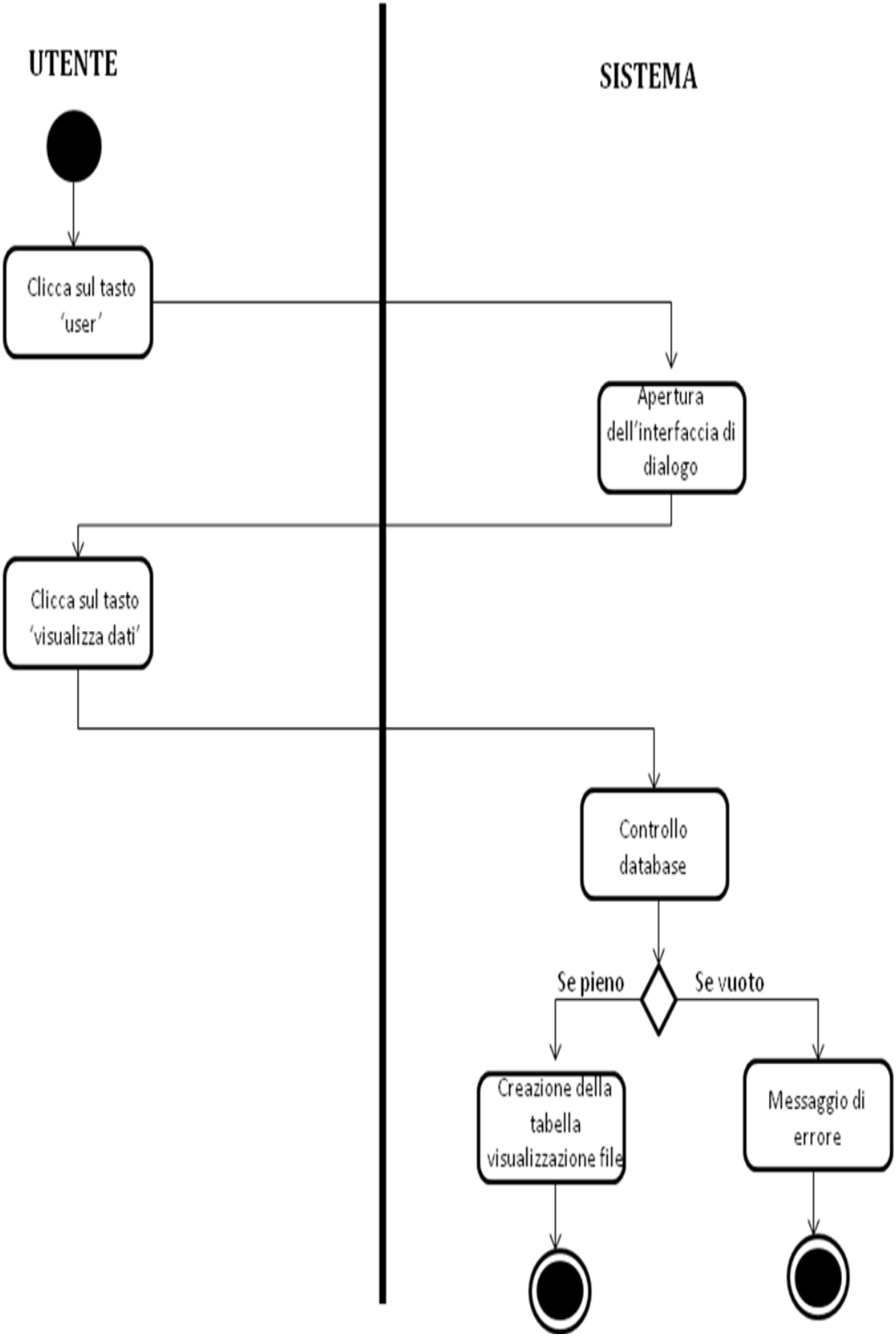
- Inserimento di dati all'interno del database per l'utente



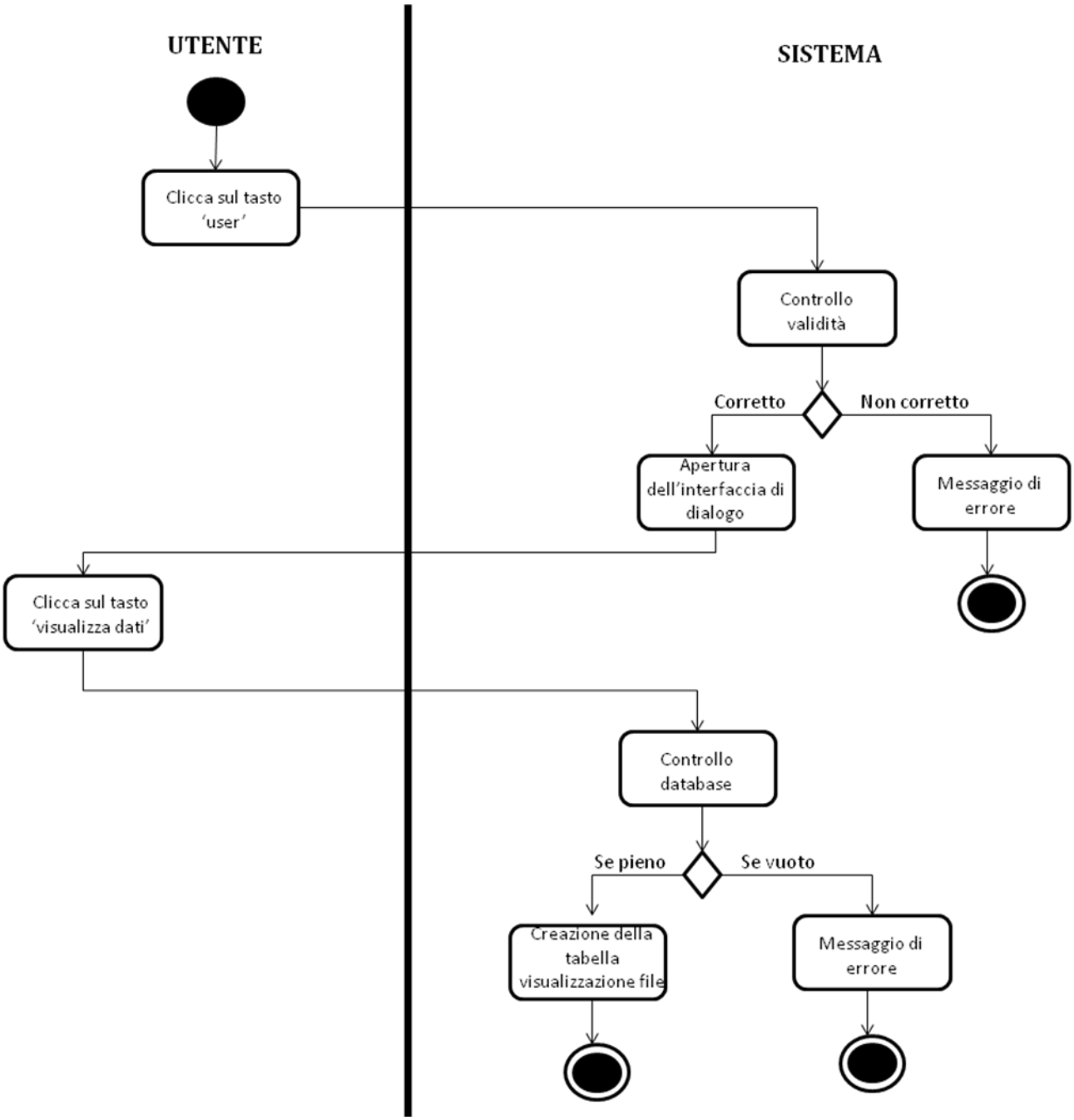
➢ Inserimento di dati all'interno del database per il programmatore



- Visualizzazione dei dati per l'utente



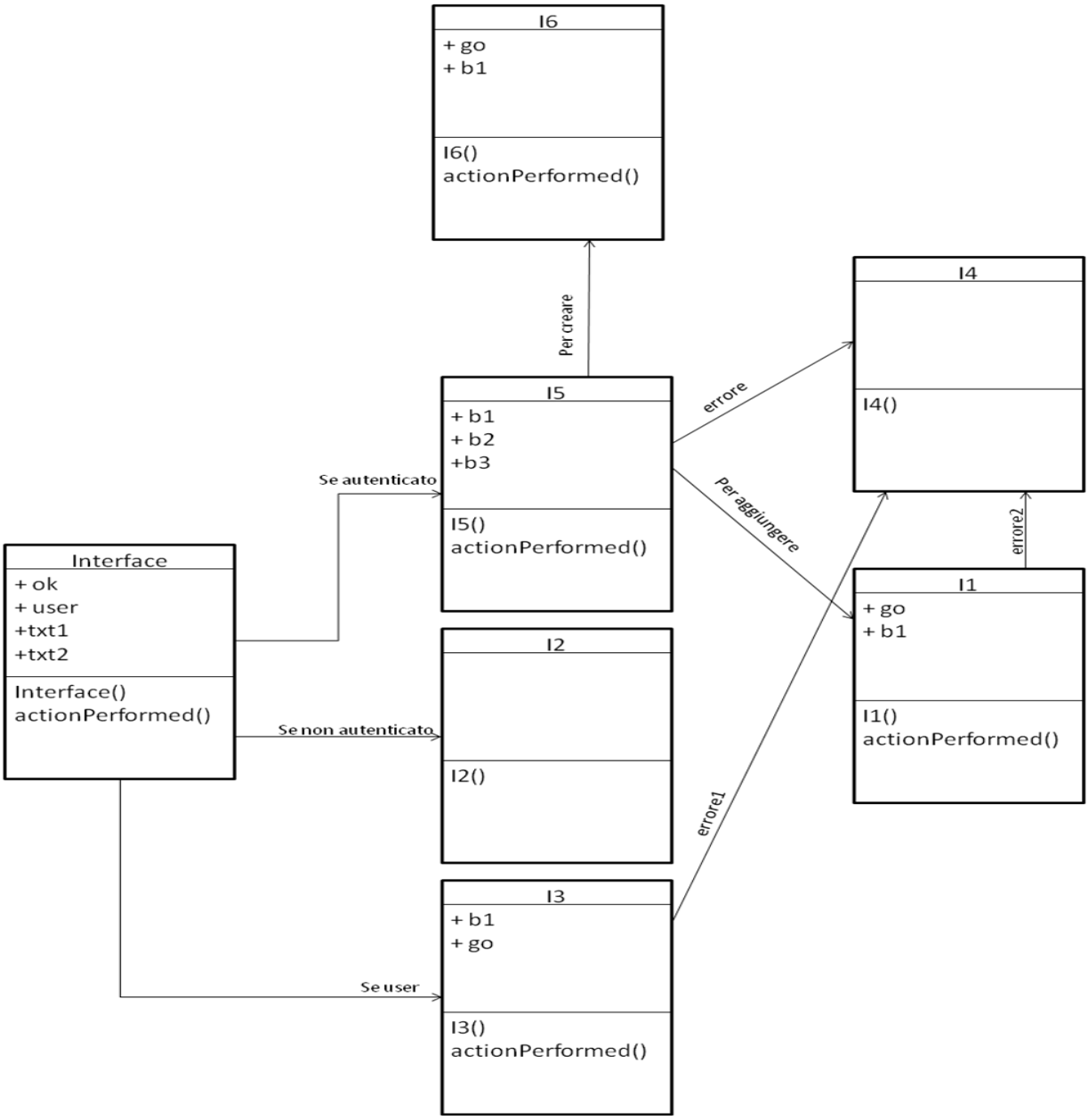
- Visualizzazione dei dati per il programmatore



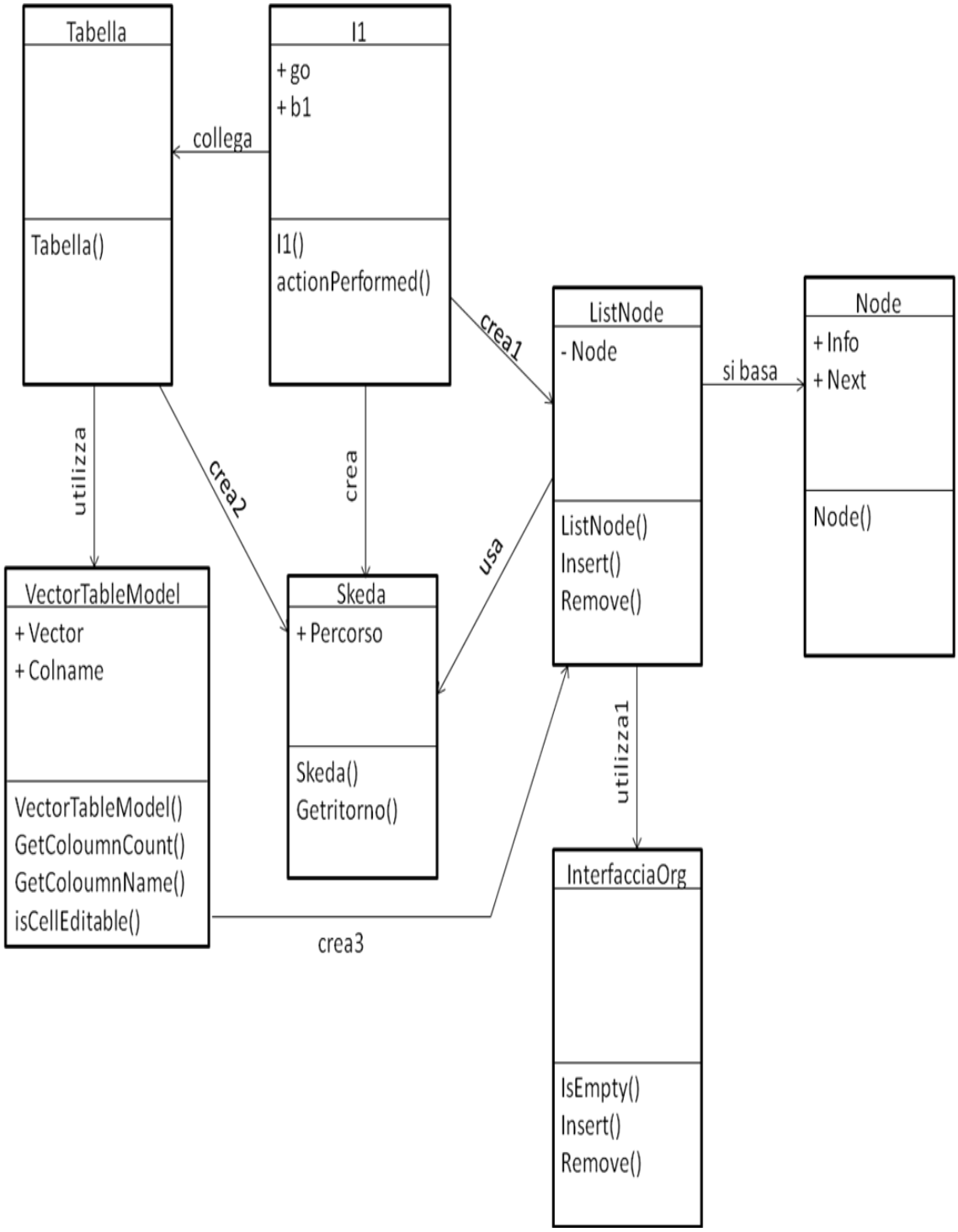
3.4 Class diagram

Andiamo ora a rappresentare un class diagram della nostra applicazione. Attraverso tali diagrammi è possibile andare ad evidenziare tutte le relazioni che ci sono tra le classi (una classe raggruppa e rappresenta più oggetti con le medesime caratteristiche) e capirne quindi l'effettiva funzione.

Per semplicità di notazione andremo a considerare dapprima i rapporti che ci sono tra le classi più importanti e poi scenderemo nel dettaglio ad analizzare i rapporti che alcune di queste classi hanno con quelle meno importanti. Riportiamo ora il class diagram delle classi principali.



Le uniche classi che a questo punto hanno delle relazioni con delle altre classi sono: I1, I3, I5 e I6. Per ognuna di queste classi il class diagram è praticamente identico a quello di seguito riportato nel caso di I1



Andando ad analizzare questo diagramma vediamo che dalla classe I1 si possono fare due possibili operazioni che sono quelle di visualizzazione dei dati (tasto b1) o di caricamento dei dati del database (tasto go).

Se si scegli l'operazione di caricamento dei dati allora il sistema attraverso il metodo I1() carica nel database il file scelto e poi scrive nella tabella il percorso del file scelto. Crea quindi le istanze delle due classi Skeda e ListNode (quest'ultima per funzionare ha bisogno della classe Node): attraverso la prima salva il percorso del file scelto, attraverso la seconda scrive tale percorso in un file ('pro.dat') da cui poi andrà a leggere quando avrà bisogno di creare la tabella.

Se, invece, si sceglie di andare a visualizzare i file caricati allora il sistema fa comparire a video una tabella (che per esistere ha bisogno della classe VectorTableModel) e poi va a creare una classe Skeda e una classe ListNode. Attraverso le istanze delle due classi legge dal file 'pro.dat' i percorsi memorizzati e poi li inserisce nella tabella facendola comparire a video.

Capitolo 4

Implementazione

4.1 Punto della situazione

Nel seguente capitolo andremo ad analizzare tutta la fase implementativa della nostra applicazione che fondamentalmente è stata divisa in due parti.

Nella prima ci si è concentrati sull'utilizzo di postgreSQL per la creazione di un database e per inserire in esso le varie mappe su cui poi saremmo andati a lavorare: quindi dapprima è stato creato un database a cui poi ci si connetteva e in cui si andavano a caricare le varie mappe in formato di shapefile. In seguito questi file venivano visualizzati attraverso QuantumGIS e attraverso i suoi plugin svolte delle operazioni su di essi.

Nella seconda, invece, è stata creata un'applicazione in Java il cui compito era quello di connettersi al database, di caricarci dei file scelti dall'utente o dal programmatore e di visualizzare i dati caricati in esso: perciò quello che prima veniva fatto attraverso l'interfaccia di postgreSQL, di QuantumGIS o attraverso il prompt dei comandi ora, attraverso una semplice interfaccia usata dall'utente, veniva fatto da Java.

Andiamo quindi ad analizzare queste due parti nel dettaglio.

4.2 Creazione di un database in PostgreSQL

Dopo aver scaricato il programma dal sito http://www.postgresql.org/ftp/binary/ ci siamo concentrati sulla creazione del database (attraverso l'interfaccia mostrata in fig. 4.a). Nell'installazione di postgreSQL a cui vogliamo dare delle funzionalità di postGIS bisogna tenere conto del fatto che è fondamentale non installare postGIS nel database template master finché non si è proprio sicuri di volerlo includere in tutti i database che poi andremo a creare. La ragione è molto semplice: dato che l'installazione di postGIS aggiunge molte funzioni al nostro database, l'utente potrebbe non volere utilizzare tutte queste funzioni che pertanto risulterebbero essere inutili e appesantirebbero l'uso dell'applicazione.

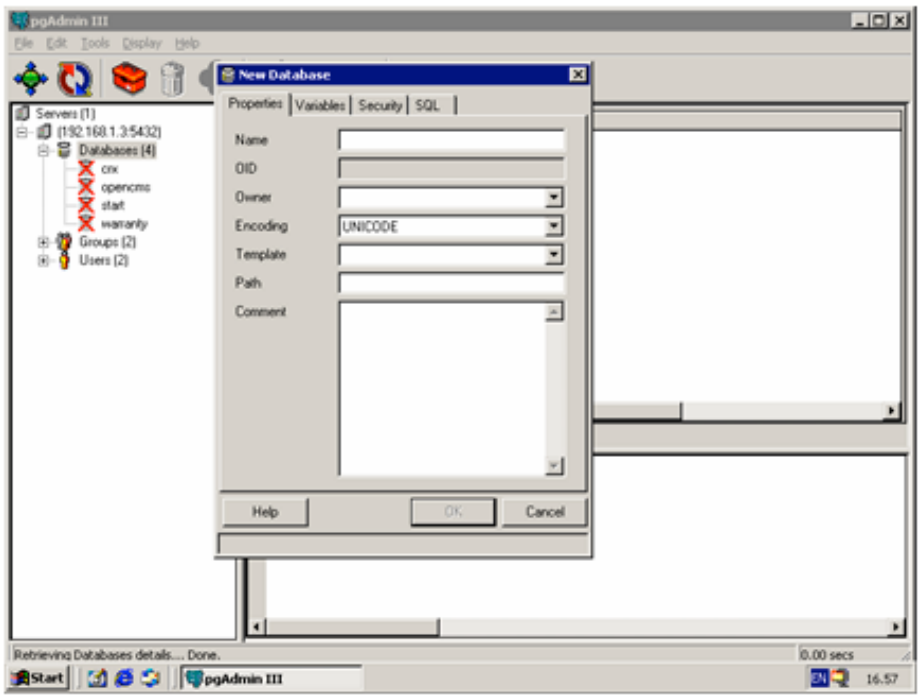


Fig. 4.a: interfaccia per la creazione di un database

Se PosgreSQL viene installato in un sistema operativo Windows bisogna notare che alla creazione del nuovo database automaticamente viene creato un nuovo database chiamato template_postGIS che racchiude al suo interno tutte le funzioni di PostGIS.

4.3 Inserimento di dati nel database

Dopo aver creato il database ci si è concentrati sull'inserimento dei dati all'interno dal database. Ci sono due modi per caricare i dati in PostgreSQL: attraverso il linguaggio SQL e attraverso il comando *shp2pgsql*. Per ognuno di questi due metodi è stato creato un database diverso su cui sono stati caricati i vari dati. La scelta di due database diversi è stata dettata dalla decisione di voler studiare i due database caricati con solo uno dei due metodi. Analizziamo nel dettaglio i due metodi.

4.3.1 Inserimento attraverso query SQL

Se si decide di caricare i dati attraverso il linguaggio SQL allora bisogna scrivere una query attraverso SQLmanager di PosgreSQL (la cui interfaccia è mostrata in fig. 4.b).

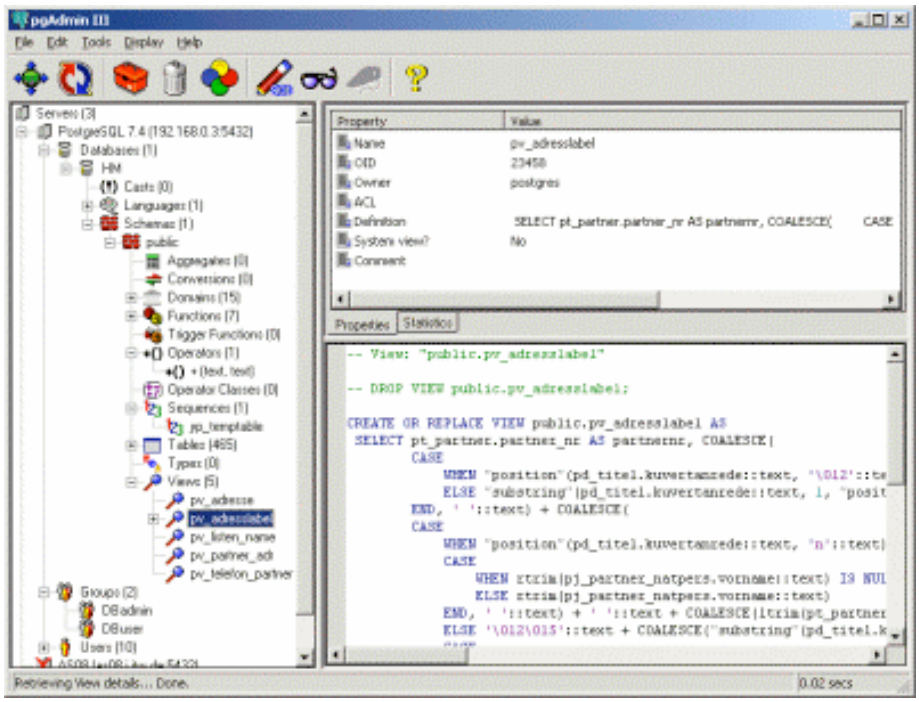


Fig. 4.b: interfaccia di SQLmanager

Un esempio lo possiamo avere con la seguente query

```
CREATE TABLE "test" ("id" integer, "name" varchar(20));
SELECT
AddGeometryColumn('test','the_geom','-1','POINT',2);
```

Attraverso questa query si va a creare la tabella *test* e in cui attraverso il comando INSERT possiamo andare ad inserire i dati geometrici che vogliamo. Un esempio di inserimento di dati lo possiamo avere con la seguente query:

```
INSERT INTO "test" ("id","name","the_geom") VALUES
(1, 'A', GeomFromText('POINT(1 1)')),
(2, 'B', GeomFromText('POINT(1 2)')),
(3, 'C', GeomFromText('POINT(2 3)')),
(4, 'D', GeomFromText('POINT(3 1)')),
(5, 'E', GeomFromText('POINT(4 0)')),
(6, 'F', GeomFromText('POINT(5 4)'));
```

4.3.2 Inserimento attraverso comando shp2pgsql

Se invece si dovesse decidere di usare il comando shp2pgsql (le cui opzioni sono mostrate in fig. 4.c) allora dal prompt dei comandi si dovrà dapprima entrare nella cartella in cui abbiamo il nostro shapefile (il dato che cioè vogliamo inserire nel database) e poi andare a digitare il comando

```
c:\pgutils\shp2pgsql -s 26986 <nome cartella> <nome file>
> <nome file>.sql
```

in cui attraverso la ridirezione del contenuto del file in un altro file .sql abbiamo creato una query che possiamo eseguire all'interno di SQLmanager di PostgreSQL caricando in questo modo il nostro file.

- **-s <srid>** Definisce il campo *SRID*. Se non è specificato, viene automaticamente definito *-1*.
- **-d** Cancella la tabella, quindi la ricrea e la popola con i dati del file shape corrente.
- **-a** Accoda il file shape nella tabella corrente, che deve essere esattamente la stessa tabella dello schema.
- **-c** Crea una nuova tabella e la popola (questa è l'azione predefinita se non viene specificata alcuna opzione).
- **-p** Modalità di preparazione, viene solamente creata la tabella.
- **-g <geometry_column>** Specifica il nome della colonna delle geometrie (usato principalmente in modalità di accodamento).
- **-D** Usa il formato dei dump di postgresql (se non viene specificato nulla vengono impostate le istruzioni di inserimento sql).
- **-k** Keep postgresql identifiers case.
- **-i** Usa il tipo *int4* per tutti i campi del file *dbf* con numeri interi.
- **-I** Create a GiST index on the geometry column.
- **-w** Usa il formato wkt (for postgis-0.x support - drops M - drifts coordinates).
- **-S** (Dalla versione 1.1.5) generare geometrie *S*ingole anziche' multiple (linee e polygons).

Fig. 4.c: opzioni del comando shp2pgsql

È importante notare che in questo caso non abbiamo scritto noi manualmente la nostra query ma abbiamo preso un file (nel nostro caso un .shp) e attraverso uno specifico comando lo abbiamo "trasformato" in una query per il nostro database. Questo metodo è quindi molto più comodo rispetto a quello precedente: infatti usando direttamente delle query SQL dobbiamo poi noi a mano andare ad inserire dei dati nel nostro database, mentre in questo caso siamo andati a caricare un file, magari scaricato da internet o ricavato in qualche altro modo, che, opportunamente modificato, è già la query del nostro database. È questo, quindi, il metodo che abbiamo utilizzato per il proseguo della nostra applicazione in Java.

4.4 Visualizzazione attraverso QuantumGIS

Caricati i file all'interno del nostro database si è passati a visualizzarli attraverso QuantumGIS. Per prima cosa è necessario andare a creare la connessione con postgreSQL attraverso la sua estensione spaziale PostGIS (interfaccia mostrata in fig. 4.d).

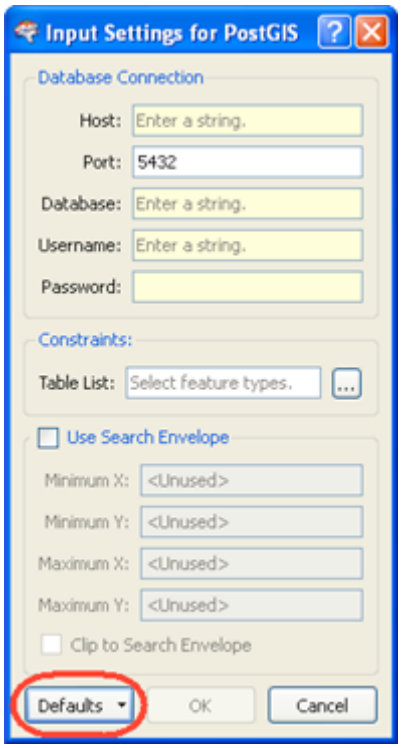


Fig. 4.d: interfaccia per la creazione della connessione a PostgreSQL

Come è già stato anticipato nel paragrafo 2.4 Postgis permette attraverso le sue funzioni di andare a lavorare con i dati geospaziali forniti da PostgreSQL. Tra le funzioni più importanti di PostGIS, vi è senza dubbio quella di inserimento ed estrazione di shapefile. Ogni shapefile è costituito da altri tre file:

- .shp file contenente le geometrie.
- .dbf file contenente gli attributi in formato dBase.
- .shx file di indice.

Ogni volta che andiamo a caricare uno shapefile andiamo a caricare tutti e tre questi altri file. La visualizzazione dei file .shp è quindi molto importante all'interno di

QuantumGIS poiché è proprio su questi file che poi andranno ad essere eseguite le potenzialità dei vari tool (alcuni dai quali mostrati in fig. 4.e).

Le operazioni che possono essere fatte sono molteplici: fondamentalmente nella nostra applicazione ci siamo concentrati sull'inserimento di punti notevoli su una mappa precaricata; le stesse identiche operazioni potevano essere fatte con qualsiasi altra operazione che è possibile fare usando i tool.

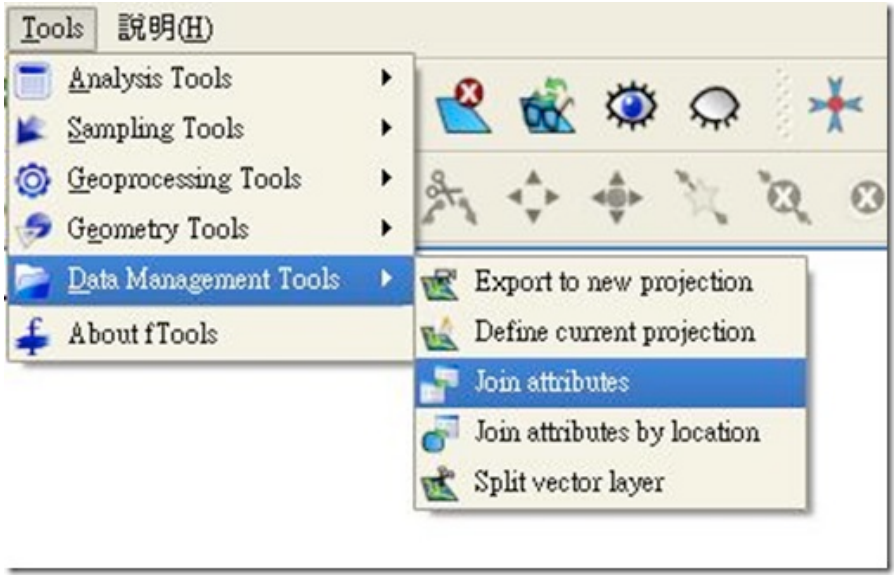


Fig. 4.e: alcuni tool di QuantumGIS

Usando quindi questo programma è stato possibile andare a visualizzare il risultato di tutto il lavori precedentemente fatto usando PosgreSQL. A tutto ciò si era però arrivati per così dire "a mano": attraverso il prompt dei comandi andavamo a caricare gli shapefile nel database, attraverso il mouse andavamo a connetterci ad esso per poi andare a selezionare un tool piuttosto che un altro e vedere a video il risultato. Cosa questa che va bene fin quando viene fatta come studio ma che risulta essere non sufficiente quando si cerca di creare un'applicazione vera e propria. Nel proseguo della nostra trattazione andremo a vedere come è stato possibile andare a superare questo problema attraverso la nostra applicazione in Java.

4.5 Connessione al database

Il primo problema che è stato necessario risolvere per lo sviluppo della nostra applicazione è stato quello di creare un collegamento con il nostro database in PostgreSQL. Java fornisce una libreria di funzioni chiamata `java.sql` che permette di risolvere appunto questo genere di problemi. Usando quindi i comandi

```
Class.forName("org.postgresql.Driver");
Connection conn=
DriverManager.getConnection("jdbc:postgresql://localhost:
<numero porta>/<nome database>”,"<utente>”,
"<password>");
Statement stmt = conn.createStatement();
```

ci siamo connessi al database su cui abbiamo poi caricato, attraverso il comando shp2pgsql digitato dal prompt dei comandi, uno shapefile. In questo modo era, infatti, possibile, attraverso delle generiche query, andare a verificare che il collegamento fosse ben riuscito. Java permette infatti anche di andare a fare delle query sul database a cui ci siamo collegati. Ad esempio attraverso i comandi:

```
String sql = "SELECT * FROM towns WHERE pop1990 > 5000
and pop1990 < 5200 ";
ResultSet rs = stmt.executeQuery(sql);
```

La nostra applicazione andava a selezionare dal nostro database tutte quell città la cui popolazione nel 1990 fosse compresa tra 7000 e 7100 abitanti.
Il risultato, così come speravamo, è stato

```
116 GROVELAND
135 LITTLETON
240 COHASSET
259 SALISBURY
341 NEWBURY
```

Verificato quindi la buona riuscita del collegamento si è passati a cercare di caricare nel database uno shapefile attraverso il comando shp2pgsql.

4.6 Esecuzione del comando shp2pgsql

Analizziamo il comando già introdotto nel paragrafo 4.3.2

```
c:\pgutils\shp2pgsql -s 26986 <nome cartella> <nome file>
> <nome file>.sql
```

Nostro obiettivo è quello di eseguirlo in Java e non più attraverso il prompt dei comandi.

Ora, in java vi è il comando Runtime.getRuntime().exec(<comando>) che permette di eseguire il <comando> che noi gli diamo in input. Quindi teoricamente se al posto di <comando> vado a scrivere la linea di codice sopra indicata Java lo eseguirebbe automaticamente. Il grosso problema, però, è che nel comando sopra indicato vi è da fare una ridirezione (il simbolo >) che Java non riesce a riconoscere. Per risolvere questo problema si è pensato di far precedere al nostro comando il percorso c://windows//system32//cmd.exe . In questo modo Java andava, attraverso questo percorso, a cercare l'eseguibile cmd.exe e appoggiandosi ad esso andava ad eseguire la ridirezione. Alla fine la linea di codice che veniva digitata risultava essere

```
String commands = "c://windows//system32//cmd.exe /c
c:\\pgutils\\shp2pgsql -s 26986 c://"+s1+ "//"+s1+"
"+"c://"+s1+"//"+s1+" >c://"+s1+"//"+s1+ ".sql";
```

Ma che cosa è quell' s1 che vediamo comparire nella nostra linea di commando? Come già detto nel capitolo 4.3.2 questo comando affinché risulti essere eseguibile deve essere eseguito nella cartella all'interno della quale vi sono i tre file .shp .shx e .dbf che compongono uno shapefile. Perciò risulta essere fondamentale portarsi in quella cartella, prendere il nome del file che vogliamo caricare (selezionandolo dal nome assoluto del file che possiamo tranquillamente ricavare) ed in inserirlo come

stringa (che è appunto la nostra s1) all'interno della nostra linea di comando. Il codice Java che deve essere scritto risulta essere

```
String s =chooser.getSelectedFile().getName();
int a = s.length();
String
s1=chooser.getSelectedFile().getName().substring(0, a-4);
```

Attraverso chooser.getSelectedFile().getName() andiamo a prendere il nome del file (di cui calcoliamo la lunghezza attraverso s.lenght()), mentre attraverso il comando chooser.getSelectedFile().getName().substring(0, a-4) andiamo a selezionare il nome del file a cui eliminiamo l'estensione. Ma che cosa è questo metodo chooser?

L'obiettivo della nostra applicazione è quello di permettere all'utente sia di inserire dati all'interno del nostro database e sia di andare a visualizzare i dati caricati. Per permettere all'utente di andare a scegliere quale file caricare nel nostro database usiamo la libreria di Java javax.swing.JFileChooser che permette all'utente, attraverso il comunissimo tasto 'sfoglia' (esempio mostrato in fig. 4.f), di andare a selezionare un file o una cartella.

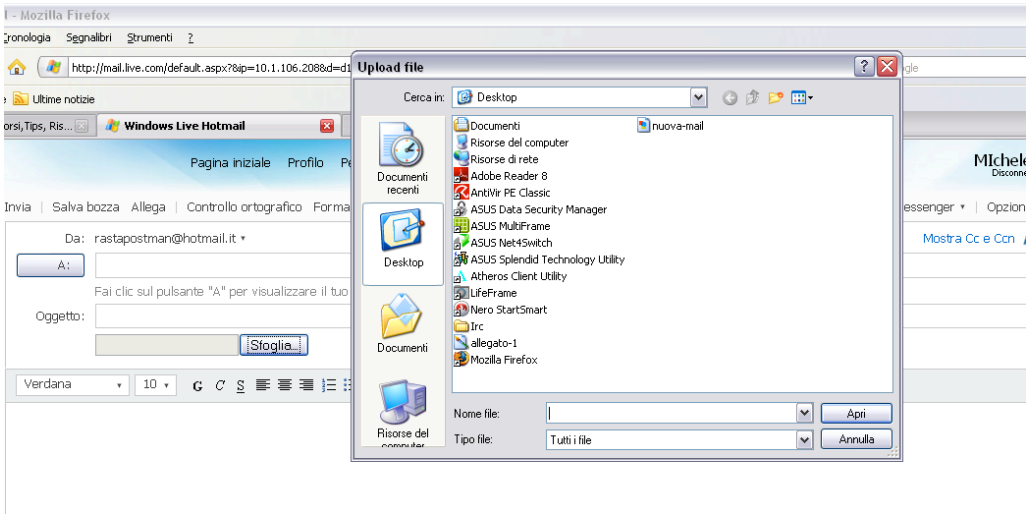


Fig. 4.f: esempio di scelta attraverso tasto 'sfoglia'

4.7 Salvataggio del percorso in 'pro.dat'

Una volta scelto il file da caricare la nostra applicazione considera il percorso di tale file come una stringa e la salva all'interno di un file ('pro.dat') da cui poi si andrà a leggere quando sarà necessario riprendere tali dati. Riportiamo alcune righe di codice di questa operazione

```
try { f = new FileOutputStream("Pro.dat"); }
catch(IOException e1){ System.exit(1); }
ObjectOutputStream os = null;
try {os = new ObjectOutputStream(f);
os.writeObject(l);
os.close();
}
catch (IOException e1)
{ System.out.println("Errore in IO");
System.exit(2);
```

Attraverso `FileOutputStream("Pro.dat")` andiamo ad aprire il file 'pro.dat' in cui poi andiamo a scrivere attraverso `os.writeObject(l).`

Nel caso in cui si voglia andare a leggere dal nostro file scriveremo

```
FileInputStream fin = null;
ObjectInputStream is = null;
try {
fin = new FileInputStream("Pro.dat");
is = new ObjectInputStream(fin);
}
catch(IOException ex){ System.exit(3); }
try { l = (ListNode)(is.readObject());
is.close();
}
catch (IOException ex){ System.exit(4); }
catch (ClassNotFoundException ex){
System.exit(5);
}
```

Questa operazione, resa possibile attraverso la libreria **`java.io`**, risulta essere fondamentale poiché infatti solo in questo modo è possibile andare a conservare i dati quando viene lanciata un'altra applicazione o quando viene spento il computer.

4.8 Creazione della tabella

Per permettere l'inserimento dei dati bisogna usare gli accorgimenti scritti nel paragrafo precedente, mentre per andare a visualizzare i dati sarà necessario andare a creare una tabella, in cui andare a inserire i percorsi dei file, che poi dovremo andare a visualizzare a video.

Per la creazione di una tabella Java ha una libreria (**`javax.swing.table.TableModel`**): una volta importata questa libreria si predispone un vettore all'interno del quale la nostra applicazione andrà ad inserire i percorsi dei file. Poi tutti gli elementi di questo vettore verranno inseriti all'interno della nostra tabella. Per fare quest'ultima operazione dobbiamo creare una nuova classe (VectorTableModel) che definisce il titolo da dare alla tabella, il numero di righe e colonne, cosa deve essere inserito all'interno di ognuna di esse e da dove deve essere preso. Perciò passiamo alla nostra classe VectorTableModel il vettore v prima definito, andiamo a creare la nostra tabella (le linee di codice corrispondenti le verranno inserite nel capitolo 5) che poi verrà visualizzata a video attraverso le seguenti linee di codice

```
TableModel dataModel = new VectorTableModel(v);
dataModel.isCellEditable(1000, 500);
JTable t = new JTable(dataModel);
JScrollPane scrollpane = new JScrollPane(t);
add(scrollpane);
```

4.9 Interfaccia

Vediamo a questo punto come è stato possibile creare un'interfaccia attraverso cui l'utente può interagire con il nostro sistema. Prima di tutto vi è una fase di autenticazione (come quella mostrata in fig. 4.g) in cui deve essere inserito il nome utente e password

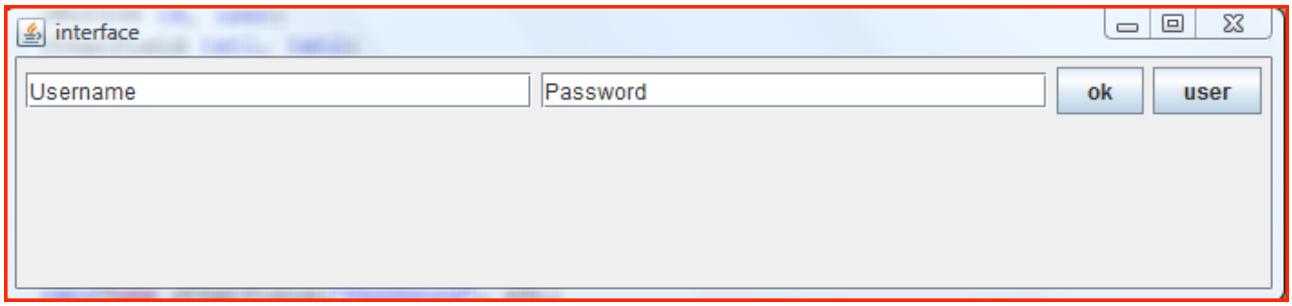


Fig. 4.g: interfaccia di autenticazione

Nel caso in cui il nostro utente sia un programmatore dovrà inserire nome utente e password e cliccare sul tasto 'ok'. Egli può a questo punto decidere se aggiungere, creare un database o visualizzare i dati: in base al tasto premuto verranno aperte le varie interfacce di dialogo.

Se invece è un utilizzatore normale clicca sul tasto 'user' e a quel punto può decidere di visualizzare i dati o di aggiungerne di nuovi in un database già precaricato.

In caso di un qualsiasi errore (come cercare di visualizzare i dati di un database vuoto) verrà comunicato a video con un opportuno messaggio.

Tutto ciò è stato possibile attraverso la libreria **`Java.swing`** che permette di creare pannelli, bottoni con cui l'utente può andare ad interagire e molte altre operazioni (il codice relativo verrà visto nel prossimo capitolo).

Capitolo 5

Codice Java

5.1 Tutte le classi

Riportiamo di seguito il codice di tutte le classi della nostra applicazione

5.1.1 Classe collegamento

Attraverso questa classe viene stabilito il collegamento fra Java e il database da noi scelto per andare a memorizzare i dati

```
package interfaccia;

import java.io.*;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Collegamento {

    public static void main(String[] args) {
        try{//comando di connessione
            Class.forName("org.postgresql.Driver");
            Connection conn=
DriverManager.getConnection("jdbc:postgresql://localhost:
5432/gisdb2", "postgres", "inter" );
            Statement stmt = conn.createStatement();

            //query di prova
            String sql = "SELECT * FROM towns WHERE
pop1990 > 7000 and pop1990 < 7100 ";
            ResultSet rs = stmt.executeQuery(sql);

            while (rs.next())
```

```
                {int num = rs.getInt("gid");
                String nome = rs.getString("town");
                System.out.println(num+" "+nome);

                }

            System.out.println("\n");

            sql = "SELECT * FROM p2 WHERE ID > 3 and ID
< 8 ";
            ResultSet rs1 = stmt.executeQuery(sql);

            while (rs1.next())
            {int num = rs1.getInt("ID");
            System.out.println(num);
            }

            rs.close();
            rs1.close();
            stmt.close();
            conn.close();
            }
            catch (ClassNotFoundException e) {
            e.printStackTrace();
            }
            catch (SQLException e) {
            e.printStackTrace();
            }
    }
}
```

5.1.2 Classe I1

Attraverso questa classe andiamo a creare un'interfaccia di dialogo a cui può accedere solo il programmatore quando vuole caricare o visualizzare dei dati.

```
package interfaccia;

import interfaccia.Skeda;
import interfaccia.I1;
import interfaccia.ListNode;
import interfaccia.Tabella;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.filechooser.FileFilter;


import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

class Terminator implements WindowListener {
    public void windowClosed(WindowEvent e){}
    public void windowClosing(WindowEvent e){
    System.exit(0);
    }
```

```
    public void windowOpened(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}


}

public class I1 extends JPanel implements ActionListener
{
    JButton go, b1;
       JFileChooser chooser;
       String choosertitle;
    public I1() {
        super();
        go = new JButton("Sfoglia");
        go.addActionListener(this);
        add(go);

        b1 = new JButton("Visualizza dati");
        b1.addActionListener(this);
        add(b1);
                }

    public void actionPerformed(ActionEvent e) {
        Object pulsantePremuto = e.getSource();

        if (pulsantePremuto==b1)
        {FileInputStream fin1 = null;
        ObjectInputStream is1 = null;
        ListNode l1 = new ListNode();
        try {
            fin1 = new FileInputStream("Pro.dat");
            is1 = new ObjectInputStream(fin1);
                    }
            catch(IOException ex)
            {
            JFrame f3 = new JFrame("visualizzazione");
            Container c3 = f3.getContentPane();
            I4 t1 = new I4();
            f3.setBounds(300, 170, 700, 480);
            c3.add(t1);
            f3.addWindowListener( new Terminator() );
```

```
            f3.setVisible(true); }

            try { l1 = (ListNode)(is1.readObject());
            is1.close();
            }
            catch (IOException ex){System.exit(4); }
            catch (ClassNotFoundException
ex){System.exit(5);  }
        JFrame f3 = new JFrame("Lista delle Schede");
        Container c3 = f3.getContentPane();
        Tabella t1 = new Tabella();
        f3.setBounds(300, 170, 700, 480);
        c3.add(t1);
        f3.addWindowListener( new Terminator() );
        f3.setVisible(true);

        }

        if (pulsantePremuto==go)
        {FileInputStream fin1 = null;
        ObjectInputStream is1 = null;
        ListNode l1 = new ListNode();
        try {
            fin1 = new FileInputStream("Pro.dat");
            is1 = new ObjectInputStream(fin1);
                    }
            catch(IOException ex)
            {
            JFrame f3 = new JFrame("visualizzazione");
            Container c3 = f3.getContentPane();
            I4 t1 = new I4();
            f3.setBounds(300, 170, 700, 480);
            c3.add(t1);
            f3.addWindowListener( new Terminator() );
            f3.setVisible(true); }

            try { l1 = (ListNode)(is1.readObject());
            is1.close();
            }
            catch (IOException ex){System.exit(4); }
            catch (ClassNotFoundException
ex){System.exit(5);  }
        chooser = new JFileChooser();
```

```
        chooser.addChoosableFileFilter (new FileFilter
() {
            public boolean accept (File f) {
                return f.isDirectory() ||
f.getName().endsWith (".shp");
            }
            public String getDescription () {
                return "File SHP";
            }
        });

        chooser.setCurrentDirectory(new
java.io.File("."));
        chooser.setDialogTitle(choosertitle);
        chooser.setAcceptAllFileFilterUsed(false);

        if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {

             try {
                 System.setProperty("user.dir",
chooser.getCurrentDirectory().toString());
                String path =
System.getProperty("user.dir");
                      System.out.println(path);

                      String s
=chooser.getSelectedFile().getName();
                      String s2
=chooser.getSelectedFile().getAbsolutePath();
                      int a = s.length();
                      String s1
=chooser.getSelectedFile().getName().substring(0, a-4);

                      System.out.println(s1);


                      String commands =
"c://windows//system32//cmd.exe /c c:\\pgutils\\shp2pgsql
-s 26986 c://"+s1+
                     "//"+s1+" "+"c://"+s1+"//"+s1+"
>c://"+s1+"//"+s1+ ".sql";
```

```
                        Process p1 =
Runtime.getRuntime().exec(commands);//cmd.exe
                        BufferedReader in = new
BufferedReader(
                                new
InputStreamReader(p1.getErrorStream()));
        String line = null;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }

        FileInputStream fin = null;
        ObjectInputStream is = null;
        Skeda sc = new Skeda(s2);
        ListNode l = new ListNode();
        try {
            fin = new FileInputStream("Pro.dat");
            is = new ObjectInputStream(fin);
            }
            catch(IOException ex){ System.exit(3); }
            try { l = (ListNode)(is.readObject());
            is.close();
            }
            catch (IOException ex){ System.exit(4); }
            catch (ClassNotFoundException ex){
            System.exit(5);
            }
        l.insert(sc);

        FileOutputStream f = null;
        try { f = new FileOutputStream("Pro.dat"); }
        catch(IOException e1){ System.exit(1); }
        ObjectOutputStream os = null;
        try {os = new ObjectOutputStream(f);
        os.writeObject(l);
        os.close();
        }
        catch (IOException e1)
        { System.out.println("Errore in IO");
        System.exit(2);
       }
                }
            catch (IOException e1) {
```

```
                    System.out.println("exception
happened");
                    e1.printStackTrace();
                    System.exit(-1);
                }

            }
        else {
          System.out.println("No Selection ");
          }
        }
    }

    public Dimension getPreferredSize(){
        return new Dimension(400, 200);
        }

    public static void main(String[] args) {

          JFrame frame = new JFrame("interfaccia1");
          I1 panel = new I1();
             frame.addWindowListener(
               new WindowAdapter() {
                 public void windowClosing(WindowEvent e)
{
                   System.exit(0);
                   }
                 }
               );
             frame.getContentPane().add(panel,"Center");
             frame.setSize(panel.getPreferredSize());
             frame.setVisible(true);
    }
  }
```

5.1.3 Classe I2

Attraverso questa classe viene comunicato all'utente un messaggio di errore quando non vengono correttamente inseriti i dati o quando l'autenticazione non ha avuto successo.

```
package interfaccia;

import java.awt.Container;


import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class I2 extends JPanel  {

    private static final long serialVersionUID = 1L;
        public I2(){
        super();
        JLabel l = new JLabel("Errore. Premere x per
uscire");
        add(l);
        }
        public static void main(String[] v){


            JFrame f5 = new JFrame("Errore");
            f5.setSize(800, 300);
            Container c5 = f5.getContentPane();
            I2 p5 = new I2();
            c5.add(p5);
            f5.setVisible(true);
            }
}
```

5.1.4 Classe I3

Attraverso questa classe l'utente può scegliere se andare a caricare o a visualizzare dei dati.

```
package interfaccia;

import interfaccia.Skeda;
import interfaccia.I1;
import interfaccia.ListNode;
import interfaccia.Tabella;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.filechooser.FileFilter;



import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;


public class I3 extends JPanel implements ActionListener
{
    JButton go, b1;
```

```
        JFileChooser chooser;
        String choosertitle;
    public I3() {
         super();
         go = new JButton("Sfoglia");
         go.addActionListener(this);
         add(go);

         b1 = new JButton("Visualizza dati");
         b1.addActionListener(this);
         add(b1);
                 }

    public void actionPerformed(ActionEvent e) {
         Object pulsantePremuto = e.getSource();

         if (pulsantePremuto==b1)
         {FileInputStream fin1 = null;
         ObjectInputStream is1 = null;
         ListNode l1 = new ListNode();
         try {
              fin1 = new FileInputStream("Pro.dat");
              is1 = new ObjectInputStream(fin1);
                      }
              catch(IOException ex)
              {
              JFrame f3 = new JFrame("User");
              Container c3 = f3.getContentPane();
              I4 t1 = new I4();
              f3.setBounds(300, 170, 700, 480);
              c3.add(t1);
              f3.addWindowListener( new Terminator() );
              f3.setVisible(true); }

              try { l1 = (ListNode)(is1.readObject());
              is1.close();
              }
              catch (IOException ex){System.exit(4); }
              catch (ClassNotFoundException
ex){System.exit(5);   }

         JFrame f3 = new JFrame("Lista delle Schede");
         Container c3 = f3.getContentPane();
         Tabella t1 = new Tabella();
```

```
        f3.setBounds(300, 170, 700, 480);
        c3.add(t1);
        f3.addWindowListener( new Terminator() );
        f3.setVisible(true);

        }


        if (pulsantePremuto==go)
        {FileInputStream fin1 = null;
        ObjectInputStream is1 = null;
        ListNode l1 = new ListNode();
        try {
            fin1 = new FileInputStream("Pro.dat");
            is1 = new ObjectInputStream(fin1);
                    }
            catch(IOException ex)
            {
            JFrame f3 = new JFrame("User");
            Container c3 = f3.getContentPane();
            I4 t1 = new I4();
            f3.setBounds(300, 170, 700, 480);
            c3.add(t1);
            f3.addWindowListener( new Terminator() );
            f3.setVisible(true); }

            try { l1 = (ListNode)(is1.readObject());
            is1.close();
            }
            catch (IOException ex){System.exit(4); }
            catch (ClassNotFoundException
ex){System.exit(5);   }


        chooser = new JFileChooser();

        chooser.addChoosableFileFilter (new FileFilter
() {
            public boolean accept (File f) {
                return f.isDirectory() ||
f.getName().endsWith (".shp");
            }
            public String getDescription () {
                return "File SHP";
```

```
            }
        });

        chooser.setCurrentDirectory(new
java.io.File("."));
        chooser.setDialogTitle(choosertitle);
        chooser.setAcceptAllFileFilterUsed(false);

        if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {

             try {
                  System.setProperty("user.dir",
chooser.getCurrentDirectory().toString());
                 String path =
System.getProperty("user.dir");
                      System.out.println(path);

                      String s
=chooser.getSelectedFile().getName();
                      String s2
=chooser.getSelectedFile().getAbsolutePath();
                      int a = s.length();
                      String s1
=chooser.getSelectedFile().getName().substring(0, a-4);

                      System.out.println(s1);



                      String commands =
"c://windows//system32//cmd.exe /c c:\\pgutils\\shp2pgsql
-s 26986 c://"+s1+
                     "//"+s1+" "+"c://"+s1+"//"+s1+"
>c://"+s1+"//"+s1+ ".sql";

                     Process p1 =
Runtime.getRuntime().exec(commands);//cmd.exe
                     BufferedReader in = new
BufferedReader(
                                 new
InputStreamReader(p1.getErrorStream()));
        String line = null;
        while ((line = in.readLine()) != null) {
```

```
            System.out.println(line);
        }

        FileInputStream fin = null;
        ObjectInputStream is = null;
        Skeda sc = new Skeda(s2);
        ListNode l = new ListNode();
        try {
            fin = new FileInputStream("Pro.dat");
            is = new ObjectInputStream(fin);
            }
            catch(IOException ex){ System.exit(3); }
            try { l = (ListNode)(is.readObject());
            is.close();
            }
            catch (IOException ex){ System.exit(4); }
            catch (ClassNotFoundException ex){
            System.exit(5);
            }
        l.insert(sc);
        FileOutputStream f = null;
        try { f = new FileOutputStream("Pro.dat"); }
        catch(IOException e1){ System.exit(1); }
        ObjectOutputStream os = null;
        try {os = new ObjectOutputStream(f);
        os.writeObject(l);
        os.close();
        }
        catch (IOException e1)
        { System.out.println("Errore in IO");
        System.exit(2);
       }


                }
            catch (IOException e1) {
                    System.out.println("exception
happened");
                    e1.printStackTrace();
                    System.exit(-1);
                }


            }
```

```
        else {
          System.out.println("No Selection ");
          }
        }


    }

    public Dimension getPreferredSize(){
        return new Dimension(400, 200);
        }

    public static void main(String[] args) {

          JFrame frame = new JFrame("interfaccia1");
          I1 panel = new I1();
             frame.addWindowListener(
               new WindowAdapter() {
                 public void windowClosing(WindowEvent e)
{
                   System.exit(0);
                   }
                 }
               );
             frame.getContentPane().add(panel,"Center");
             frame.setSize(panel.getPreferredSize());
             frame.setVisible(true);


    }
  }
```

5.1.5 Classe I4

Attraverso questa classe viene comunicato a video all'utente che non ci sono dati all'interno del database. Questo messaggio compare quindi se l'utente o il programmatore cercano di visualizzare i dati del database che non è stato precedentemente caricato.

```
package interfaccia;

import java.awt.Container;


import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class I4 extends JPanel  {

    private static final long serialVersionUID = 1L;
        public I4(){
        super();
        JLabel l = new JLabel("Errore: non ci sono dati.
Premere x per uscire");
        add(l);
        }
        public static void main(String[] v){


            JFrame f5 = new JFrame("Errore");
            f5.setSize(800, 300);
            Container c5 = f5.getContentPane();
            I2 p5 = new I2();
            c5.add(p5);
            f5.setVisible(true);
            }
```

5.1.6 Classe I5

Attraverso questa classe viene creata l'interfaccia di dialogo attraverso cui il programmatore può decidere se andare a visualizzare, a inserire dati nel database o ad inserirne di nuovi.

```
package interfaccia;

import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class I5 extends JPanel implements ActionListener{

JButton b1, b2, b3;
public I5() {
    super();

    b2 = new JButton("crea");
    b2.addActionListener(this);
    add(b2);

    b1 = new JButton("aggiungi");
    b1.addActionListener(this);
    add(b1);

    b3 = new JButton("visualizza dati");
    b3.addActionListener(this);
    add(b3);
}

public void actionPerformed(ActionEvent e)
    {Object pulsantePremuto = e.getSource();
     if (pulsantePremuto==b1)
        {JFrame f2 = new JFrame("Aggiunta");
```

```
        Container c2= f2.getContentPane();
        I1 p2 = new I1();
        c2.add(p2);
        f2.setBounds(300,0,600,200);
        f2.setVisible(true);
        }
    if (pulsantePremuto==b2)
        {JFrame f1 = new JFrame("creazione");
        Container c1= f1.getContentPane();
        I6 p1 = new I6();
        c1.add(p1);
        f1.setBounds(300,0,600,150);
        f1.setVisible(true);
        }
    if (pulsantePremuto==b3)
        {FileInputStream fin1 = null;
        ObjectInputStream is1 = null;
        ListNode l1 = new ListNode();
        try {
            fin1 = new FileInputStream("Pro.dat");
            is1 = new ObjectInputStream(fin1);
                    }
            catch(IOException ex)
            {
            JFrame f3 = new JFrame("visualizzazione");
            Container c3 = f3.getContentPane();
            I4 t1 = new I4();
            f3.setBounds(300, 170, 700, 480);
            c3.add(t1);
            f3.addWindowListener( new Terminator() );
            f3.setVisible(true); }

            try { l1 = (ListNode)(is1.readObject());
            is1.close();
            }
            catch (IOException ex){System.exit(4); }
            catch (ClassNotFoundException
ex){System.exit(5);   }

        JFrame f3 = new JFrame("Lista delle Schede");
        Container c3 = f3.getContentPane();
        Tabella t1 = new Tabella();
        f3.setBounds(300, 170, 700, 480);
        c3.add(t1);
```

```
        f3.addWindowListener( new Terminator() );
        f3.setVisible(true);
        }
    }

public static void main(String[] args) {

    JFrame sfondo = new JFrame("programmatore");
    sfondo.setSize(1280, 800);
    Container d = sfondo.getContentPane();
    sfondo.setVisible(true);

    JFrame f = new JFrame("programmatore");
    f.setSize(300, 800);
    Container c= f.getContentPane();
    I5 p = new I5();
    c.add(p);
    f.setVisible(true);

    }
}
```

5.1.7 Classe I6

Attraverso questa classe viene creata un'interfaccia di dialogo attraverso cui l'utente può decidere di andare a visualizzare o ad inserire i dati nel database. Non gli viene permesso quindi di inserire (come in I5) dei dati nuovi nel database.

```
package interfaccia;
import interfaccia.Skeda;
import interfaccia.I1;
import interfaccia.ListNode;
import interfaccia.Tabella;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.filechooser.FileFilter;


import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;



public class I6 extends JPanel implements ActionListener
{
```

```
    JButton go, b1;
       JFileChooser chooser;
       String choosertitle;
    public I6() {
         super();
         go = new JButton("Sfoglia");
         go.addActionListener(this);
         add(go);

         b1 = new JButton("Visualizza dati");
         b1.addActionListener(this);
         add(b1);
                 }

    public void actionPerformed(ActionEvent e) {
         Object pulsantePremuto = e.getSource();

         if (pulsantePremuto==b1)
         {
         JFrame f3 = new JFrame("Lista delle Schede");
         Container c3 = f3.getContentPane();
         Tabella t1 = new Tabella();
         f3.setBounds(300, 170, 700, 480);
         c3.add(t1);
         f3.addWindowListener( new Terminator() );
         f3.setVisible(true);

         }


         if (pulsantePremuto==go)
         {chooser = new JFileChooser();

         chooser.addChoosableFileFilter (new FileFilter
() {
             public boolean accept (File f) {
                 return f.isDirectory() ||
f.getName().endsWith (".shp");
             }
             public String getDescription () {
                 return "File SHP";
             }
         });
```

```
        chooser.setCurrentDirectory(new
java.io.File("."));
        chooser.setDialogTitle(choosertitle);
        chooser.setAcceptAllFileFilterUsed(false);

        if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {

             try {
                 System.setProperty("user.dir",
chooser.getCurrentDirectory().toString());
                 String path =
System.getProperty("user.dir");
                     System.out.println(path);

                     String s
=chooser.getSelectedFile().getName();
                     String s2
=chooser.getSelectedFile().getAbsolutePath();
                     int a = s.length();
                     String s1
=chooser.getSelectedFile().getName().substring(0, a-4);

                     System.out.println(s1);



                     String commands =
"c://windows//system32//cmd.exe /c c:\\pgutils\\shp2pgsql
-s 26986 c://"+s1+
                    "//"+s1+" "+"c://"+s1+"//"+s1+"
>c://"+s1+"//"+s1+ ".sql";

                    Process p1 =
Runtime.getRuntime().exec(commands);//cmd.exe
                    BufferedReader in = new
BufferedReader(
                              new
InputStreamReader(p1.getErrorStream()));
        String line = null;
        while ((line = in.readLine()) != null) {
            System.out.println(line);
        }
```

```
        Skeda sc = new Skeda(s2);
        ListNode l = new ListNode();

        l.insert(sc);

        FileOutputStream f = null;
        try { f = new FileOutputStream("Pro.dat"); }
        catch(IOException e1){ System.exit(1); }
        ObjectOutputStream os = null;
        try {os = new ObjectOutputStream(f);
        os.writeObject(l);
        os.close();
        }
        catch (IOException e1)
        { System.out.println("Errore in IO");
        System.exit(2);
       }


                }
            catch (IOException e1) {
                    System.out.println("exception
happened");
                    e1.printStackTrace();
                    System.exit(-1);
                }

            }
       else {
          System.out.println("No Selection ");
          }
        }




    }

    public Dimension getPreferredSize(){
        return new Dimension(400, 200);
        }
```

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

          JFrame frame = new JFrame("interfaccia1");
          I1 panel = new I1();
             frame.addWindowListener(
               new WindowAdapter() {
                 public void windowClosing(WindowEvent e)
{
                   System.exit(0);
                   }
               }
               );
             frame.getContentPane().add(panel,"Center");
             frame.setSize(panel.getPreferredSize());
             frame.setVisible(true);

    }
}
```

5.1.8 Classe InterfacciaOrg

Questa classe viene utilizzata dalla classe ListNode per andare a creare la lista dei dati.

```
package interfaccia;


import interfaccia.Skeda;


public interface InterfacciaOrg
{
public boolean isEmpty();
public void insert(Skeda sc);
public void revInsert(InterfacciaOrg h);
public void remove(Skeda sc);
public Skeda head();
public InterfacciaOrg tail();
}
```

5.1.9 Classe Interface

Questa è la prima classe ad essere utilizzata poiché va a creare quella interfaccia che permette all'utente di accedere al database o al programmatore di accerdervi con le sue credenziali (username e password).

```
package interfaccia;



import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Interface extends JPanel implements
ActionListener {
    JButton ok, user;
    JTextField txt1, txt2;
    public Interface(){
    super();
    ok = new JButton("ok");
    ok.addActionListener(this);
    user = new JButton("user");
    user.addActionListener(this);
    txt1=new JTextField("Username", 25);
    txt2=new JTextField("Password", 25);
    add(txt1);
    add(txt2);
    add(ok);
    add(user);
```

```
    }
public void actionPerformed(ActionEvent e)
    {
    Object pulsantePremuto = e.getSource();
     if (pulsantePremuto==ok)
     {String a= txt1.getText();
     String b="prog";
     String d= txt2.getText();
     String f="prog";
    int c = b.compareTo(a);
     int g = f.compareTo(d);
     if ((c==0)&&(g==0))
         {JFrame f3 = new JFrame("Interface");
         Container c3 = f3.getContentPane();
         I5 t1 = new I5();
         f3.setBounds(300, 170, 700, 480);
         c3.add(t1);
         f3.addWindowListener( new Terminator() );
         f3.setVisible(true);
         }
     else
         {JFrame f3 = new JFrame("Errore");
         Container c3 = f3.getContentPane();
         I2 t1 = new I2();
         f3.setBounds(300, 170, 700, 480);
         c3.add(t1);
         f3.addWindowListener( new Terminator() );
         f3.setVisible(true);
         }
     }
     if (pulsantePremuto==user)
         {JFrame f3 = new JFrame("User");
         Container c3 = f3.getContentPane();
         I3 t1 = new I3();
         f3.setBounds(300, 170, 700, 480);
         c3.add(t1);
         f3.addWindowListener( new Terminator() );
         f3.setVisible(true);

         }



    }
```

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

      JFrame frame = new JFrame("interface");
      Interface panel = new Interface();
      frame.setSize(700, 200);
         frame.addWindowListener(
           new WindowAdapter() {
             public void windowClosing(WindowEvent e) {
               System.exit(0);
               }
             }
           );
         frame.getContentPane().add(panel,"Center");
         frame.setVisible(true);


     }
}
```

5.1.10 Classe ListNode

Attraverso questa classe è possibile andare a creare la lista di dati che poi verranno usta dal programmatore o dall'utente.

```
package interfaccia;


import interfaccia.InterfacciaOrg;
import interfaccia.ListNode;
import interfaccia.Node;

import java.io.*;

public class ListNode implements InterfacciaOrg,
Serializable
{
    private Node node;


    public ListNode()
    {
    node = null;
    }
    //se vuoto
    public boolean isEmpty()
    {
        return (node == null);
    }

    //costruisce la lista vuota
    public ListNode(Skeda sc)
    {
    node = new Node(sc);
    }

    //inserisce tutti gli elementi della lista h
mantenendo l'ordine
    public ListNode(Skeda sc, InterfacciaOrg h)
    {
        if (h != null)
        revInsert(h);
        insert(sc);
```

```
    }
    //legge gli elementi della lista l e li inserisce in
modo da mantenere l'ordine.
    public void revInsert(InterfacciaOrg h)
    {
    if (!h.isEmpty())
    {
    revInsert(h.tail());
    insert(h.head());
    }
    }
    //insert
    public void insert(Skeda sc)
    {
    // costruisce un nuovo nodo con o all'inizio
    // e il vecchio nodo come successivo
    node = new Node(sc, node);
    }

    //il primo elemento
    public Skeda head()
    { if (isEmpty())
    return null;
    else
    return node.info;
    }


    //coda della lista

    public ListNode tail()
    {
    if (isEmpty())
    return null;
    else
    {
        ListNode tmp = new ListNode();
        tmp.node = node.next;
        return tmp;
    }
    }
    //cancella un oggetto
    public void remove(Skeda sc)
    {
```

```
if (!isEmpty())
{ Node pred = node;
Node tmp = node.next;
if (node.info.equals(sc))
node = node.next;
else
     while (tmp != null)
          if (tmp.info.equals(sc))
          { pred.next = tmp.next;
          return;
          }
else
     { pred = pred.next;
     tmp = tmp.next;
     }
}
}
}
```

5.1.11 Classe Node

Su questa classe si basa la classe ListNode per la creazione delle lista di dati. Infatti attraverso questa classe vengono messi uno di fila all'altra i vari dati che poi saranno utilizzati dalla classe ListNode.

```
package interfaccia;

import interfaccia.Node;
import interfaccia.Skeda;

import java.io.Serializable;

class Node implements Serializable
{
Skeda info;
Node next;

public Node(Skeda sc)
{
this(sc, null);
}

public Node(Skeda sc, Node n)
{
info = sc;
next = n;
}
}
```

5.1.12 Classe Skeda

Attraverso questa classe viene preparato una sorta di “recipiente” all’interno del quale vengono memorizzati i percorsi dei file caricati nel database.

```
package interfaccia;

import java.io.Serializable;

class Skeda implements Serializable
{
 String percorso;
 //costruttore
 public Skeda(String a)
 {percorso= a;

 }

 public String getritorno()
    {return percorso+"\n"; }
}
```

5.1.13 Classe Tabella

Attraverso questa classe viene creata la tabella in cui viene inserita la lista dei file caricati.

```
package interfaccia;

import java.awt.Component;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.Vector;
import javax.swing.*;
import javax.swing.table.TableModel;


public class Tabella extends JPanel {
    public Tabella() {

    // predispone il vettore
    Vector v = new Vector(200);

    FileInputStream fin = null;
    ObjectInputStream is = null;
    try {
        fin = new FileInputStream("Pro.dat");
        is = new ObjectInputStream(fin);
        }
    catch(IOException ex){ System.exit(3); }
    ListNode l=null;
    try { l = (ListNode)(is.readObject());
        is.close();
        }
    catch (IOException ex){ System.exit(4); }
    catch (ClassNotFoundException ex){
            System.exit(5);
        }

    ListNode ll=l;

    while (ll.head()!=null)
    {   Skeda sc = ll.head();
        v.add(sc);
```

```
        ll=ll.tail();
    }

    // crea il modello di dati
    // a partire dal vettore
    TableModel dataModel = new VectorTableModel(v);
    dataModel.isCellEditable(1000, 500);
    // crea la tabella
    JTable t = new JTable(dataModel);
    // aggiunge la tabella ad uno ScrollPane
    JScrollPane scrollpane = new JScrollPane(t);
    // aggiunge lo ScrollPane al pannello
    add(scrollpane);
    }

}
```

5.1.14 Classe VectorTableModel

Attraverso questa classe viene formalmente creata la tabella definendo il numero di righe e di colonne e cosa verrà inserito al loro interno.

```
package interfaccia;

import java.util.Vector;
import javax.swing.table.AbstractTableModel;
public class VectorTableModel extends AbstractTableModel
{
    Vector v = null;
    // intestazioni delle colonne
    String[] ColName = {"Percorso"};
    public VectorTableModel(Vector v) {
        this.v = v;// inizializzato con il vettore
        }
    // il numero di colonne
    public int getColumnCount()
    { return ColName.length; }
    // numero righe = dimensione del vettore
    public int getRowCount() { return v.size();}
    // ritorna il contenuto di una cella
    public Object getValueAt(int row, int col) {
        // seleziona il film
        Skeda fil = (Skeda)v.elementAt(row);
        String val = null;
        // la stringa corrispondente alla colonna
        switch (col){
            case 0: val = fil.percorso; break;
            default: val = "";
        }
        return val;
    }
    // ritorna il nome della colonna
    public String getColumnName(int col) {
        return ColName[col];
    }
    // specifica se le celle sono editabili
    public boolean isCellEditable(int row, int col)
        {// nessuna cella editabile
        return false;
    }
```

Conclusioni e sviluppi futuri

Partendo dal caso di studio, passando per l'utilizzo delle varie applicazioni e finendo con la scrittura del codice Java siamo arrivati a questo punto dell'elaborato in cui possiamo andare a fare un bilancio dei risultati ottenuti:

- ✓ Sono state studiate ed analizzate tutte le immense capacità degli SDBMS focalizzandoci sulle loro caratteristiche e sulla loro utilità nello sviluppo della nostra applicazione in Java.
- ✓ Ci si è concentrati sullo studio di tre importantissimi programmi, PosgreSQL, PostGIS e QuantumGIS, utilissimi in tutti i campi in cui vengono usati gli SDBMS.
- ✓ Ci si è in seguito concentrati sull'utilizzo di tali programmi, attraverso la creazione di diverse query, e sull'analisi dei risultati ottenuti.
- ✓ Infine si è tradotto tutto quello che in precedenza era stato fatto in linguaggio Java attraverso la creazione di un'applicazione che oltre a caricare dati in un database permette di visualizzare tutti quelli in precedenza caricati.
- ✓ È stata poi sviluppata ed implementata la portabilità su altri sistemi operativi diversi da Windows come Linux o Mac.
- ✓ È stata sviluppata ed implementata un'interfaccia semplice per la nostra applicazione che potesse essere usata senza problemi da tutte le tipologie di utenti. Tale risultato era fondamentale soprattutto in campo militare dove si

deve sempre ricercare la creazione di programmi semplici da usare in campo operativo anche nelle più disparate situazioni tattiche.

Poiché quest'applicazione è nata sia per la ricerca scientifica che per il campo professionale, numerose sono le migliorie che possono essere fatte in futuro tra cui:

- ✓ Cercare di migliorare ulteriormente la portabilità della nostra applicazione anche su altri sistemi operativi cercando di andare a ottenere la ridirezione del comando shp2pgsql in maniera diversa da come è stato fatto in questa tesi (usando l'eseguibile di Windows cmd.exe).
- ✓ Cercare di migliorare ulteriormente la nostra applicazione inserendo degli altri comandi quali quello di cancellazione di uno specifico file dal database, o la modifica di una voce nella tabella di "Visualizzazione percorsi".
- ✓ Migliorare ulteriormente l'interfaccia grafica rendendola ancora più intuitiva e semplice da usare senza appesantirla troppo con l'inserimento di troppi pulsanti.
- ✓ Ottimizzare le prestazioni della nostra applicazione velocizzando di almeno 30-40ms i tempi di caricamento dei dati all'interno del database (che attualmente si aggira intorno ai 330-340 ms).

Concludendo possiamo dire che il lavoro svolto in questa tesi ha portato all'incontro e all'approfondimento con tante materie già studiate durante il corso di laurea. Lo studio e l'analisi fatta sui database spaziali, la formulazione di query e l'utilizzo del linguaggio SQL sicuramente sarebbe stato molto più complicato senza le conoscenze acquisite durante il corso di DataBase. Nello sviluppo di questa tesi (soprattutto per il Capitolo 3) molto importanti sono state le conoscenze apprese dal corso di Ingegneria

del Software. La possibilità di utilizzare anche su altre piattaforme la nostra applicazione è dovuta al Corso di Sistemi Operativi, tenuto dalla prof.sa Leonardi relatrice di questa tesi, oltre che all'intrinseca portabilità di Java, le cui caratteristiche sono state analizzate durante il corso di Informatica C tenuto dalla prof.sa Mandreoli, altra relatrice di questa tesi.

BIBLIOGRAFIA

[1] Domenico Beneventano, Sonia Bergamaschi, Francesco Guerra, Maurizio Vincini
Progetto di Basi di Dati Relazionali
Pitagora Editrice Bologna

[2] John R. Hubbard
Programmare in Java
McGraw-Hill

[3] Pressman
Principi di Ingegneria del Software
McGraw-Hill

[4] Ralf Hartmut Güting
An Introduction to Spatial Database Systems

[5] PostGis tutorial: www.isgroup.unimo.it/corsi/TEvGD/lucidi/esercitazioni/DL1-HandsOn_PostGIS.pdf

[6] PostgreSQL tutorial: www.postgresql.org/files/documentation/pdf/7.3/tutorial-7.3.2-A4.pdf

[7] QuantumGIS tutorial:
http://gis.coaps.fsu.edu/FOSS_GIS/Introduction_to_Quantum_GIS_0_8_0.pdf

[8] Manuale di QuantumGIS: www.download.osgeo.org/qgis/doc/manual/qgis-0.9.1_user_guide_it.pdf

[9] Manuale di PosgreSQL: www.postgresql.org/docs/8.1/static/

[10] Dispense varie su database spaziali:www.isgroup.unimo.it/