KOSMOS: UN SERVER "OODBMS"PER I GEODATABASE

Vincenzo BARRILE* , Francesco COTRONEO*

Università degli Studi Mediterranea di Reggio Calabria – DIMET,
barrile@ing.unirc.it, francesco.cotroneo@unirc.it

Riassunto
Come noto, il paradigma ad oggetti capace di offrire una modellizzazione priva di errori, consistente ed efficiente, viene implementato nei vari software GeoDB solo nello strato di rappresentazione, ovvero di interfaccia utente, mentre nello storage dei dati ci si appoggia a Server DBMS che utilizzano il paradigma relazionale**.** Questa scelta comporta un notevole degrado delle prestazioni del sistema. A seguito di tale considerazione si è deciso di sviluppare un Server OODBMS sperimentale dal nome "Kosmos", che, diversamente dagli altri software DBMS ad oggetti, esso è orientato ai GeoDatabase. La realizzazione di Kosmos, segue una prima fase durante la quale è stata definita una struttura algebrica, un rigoroso modello formale per la rappresentazione degli oggetti GIS e degli operatori funzionali che operano su di essi. Il progetto del software avviene secondo le modalità dell' Open Source; più in particolare Kosmos viene rilasciato sotto licenza GPL ed alla scrittura del suo codice sorgente può partecipare chiunque. Il sito web cui gli autori inviano il loro contributo è ospitato dal portale della Facoltà di Ingegneria - Università degli Studi Mediterranea di Reggio Calabria www.ing.unirc.it. La prima versione sperimentale di Kosmos è stata utilizzata per fare diversi test con dei GIS. I test sono stati condotti utilizzando l'accoppiata GRASS – Kosmos e facendo confronti con diverse soluzioni esistenti come GRASS – mySQL , ArcINFO – Oracle.

Abstract
As known, the object-oriented approach, used to provide an error-free model having consistence and efficiency, is implemented in each common GeoDB software just at a representation level, that's to say at the level of user interface. On the contrary, in data storage, you deals with DBMS servers using a relational approach. That choice implies an important fall in the system performances. In consideration of these facts, we develop an alpha OODBMS server, named "Kosmos", oriented to GeoDatabases, differently from all other DBMS softwares. Kosmos making passes through a preliminary step in which we are going to define an algebraic structure and a rigorous formal model to describe GIS objects and functional operators acting on these objects. Software design follows the open source philosophy; in particular, Kosmos is released under GPL license and anyone can take part in its coding. The web site supposed to collect authors' contributes to the project is hosted by the web-server www.ing.unirc.it, maintained by Faculty of Engineering, University "Mediterranea" of Reggio Calabria, Italy. First Kosmos alpha release has been employed to make some tests over GISs. These were performed matching GRASS - Kosmos and comparing results to different existing solutions like GRASS - mySQL and ArcINFO - Oracle.

Introduzione
L'utilizzo di software GIS legato alla modellizzazione con i soli layer a scarsa tipizzazione e, ancor peggio, a quella CAD generica, può comportare che i risultati delle query siano interpretati erroneamente (la risposta che si ottiene non è relativa alla domanda che si crede sia stata fatta), o che si abbia una certa ridondanza concettuale nella banca dati, ovvero che dati di rilievo o di campionamento siano in realtà ottenibili da query su dati già disponibili. I modelli di costruzione dei GIS cui si accennava prima, danno dei problemi anche nella fase di inserimento e modifica dei dati

poiché è difficile costruire dei vincoli operativi (ad esempio per l'inserimento, la cancellazione, la modifica ecc...) che non siano di tipo strettamente topologico. Per quanto riguarda il progettista del GIS, anche se dotato di ottime doti di building per modelli astratti, non potrà sperare in tempi di debugging ragionevoli qualora utilizzi software che fa uso dei paradigmi tradizionali.
Per superare queste limitazioni con gli anni hanno visto la luce diverse soluzioni software prodotte da grandi multinazionali operanti nel settore, che implementano il paradigma ad oggetti per la modellizzazione degli elementi da rappresentare nella "mappa virtuale" dei GIS. Brevemente, il suddetto paradigma si fonda sull'utilizzo di particolari strutture formali dette Classi, queste sono caratterizzate dal fatto di essere degli insiemi che generalizzano le proprietà definite da Cantor. Le Classi di oggetti possono essere definite sia come dichiarazione dei tipi di elementi contenuti in esse, che come specializzazione di altre Classi aggiungendo a queste la dichiarazione di altri tipi (in tal caso si parla di derivazione); inoltre gli elementi contenuti in esse possono essere delle funzioni che operano su vari domini di Classi, ed ancora tali domini possono variare dinamicamente da una Classe genitrice ad una derivata, cosicché oggetti appartenenti a diverse Classi, ma derivate da una Classe comune si "comportano" in modo diverso nello stesso contesto rappresentativo. Quest' ultima proprietà è probabilmente la più importante in ambito GIS e viene denominata "Polimorfismo", da qui è anche comprensibile perché le funzioni di una Classe si definiscono "metodi" o "comportamenti". La capacità di astrazione cercata in tal modo può essere facilmente ottenuta: si pensi di voler rappresentare in ambito, GIS, dei tubi dislocati sul territorio; se nel modello si specializzano le classi "tubi di amianto cemento" e "tubi di ghisa" e possibile differenziarne i comportamenti, in maniera trasparente all'utilizzatore, a seguito di un tentativo di allaccio, pur essendo entrambi risedenti nello stesso layer.
Da qui in avanti, per motivi che saranno chiari in seguito, chiameremo le Classi di modello degli elementi geografici "GeoClassi".

Kosmos: obiettivi e funzionalità
I più diffusi software GIS presenti oggi sul mercato permettono di costruire il Sistema Informativo utilizzando il paradigma progettuale orientato agli oggetti (ovvero la modellizzazione con le GeoClassi) e quindi permettono di implementare un cosiddetto Geodatabase. Purtroppo lo storage fisico dei dati alfanumerici e spaziali viene fatto con l'ausilio esterno di Server Database di tipo RDBMS e quindi mentre l'utente interagisce con le GeoClassi, queste vengono implementate "fisicamente" dal sistema utilizzando l'algebra relazionale (tabelle indicizzate): di fatto questa scelta restringe di molto le potenzialità del modello ad oggetti oltre che a comportare un notevole decadimento prestazionale.
Kosmos nasce, in un primo momento, dall'esigenza di avere un software per la memorizzazione delle GeoClassi attraverso il paradigma ad oggetti. Di conseguenza esso può essere definito come un Server "OODBMS" (object-oriented database management system), ma più precisamente un "OODBMS" orientato ai GIS poiché implementa delle soluzioni specifiche di tal ambito tralasciando tutte le opzioni necessari in altri contesti.
Subito dopo le prime fasi di sviluppo, l'evidente isomorfismo strutturale delle GeoClassi con i relativi oggetti utilizzati per il loro storage, ci porto' a prendere la decisione di integrare in Kosmos tutte le funzionalità per la Gestione delle GeoClassi: in altri termini Kosmos è allo stato attuale un Gis con modello GeoDatabase ed un Server "OODBMS" per lo storage, tutto in un'unico software.
Mentre per i software cui si faceva rifermento prima si ha un'architettura Client - Server, dove, il Client si occupa delle GeoClassi (interrogazioni, creazione, legende, layer, ecc..., ovvero il software GIS), mentre il Server si occupa dello storage, e della gestione della transazioni multicliente. Nel caso di Kosmos il Client è un qualcosa di appena più complesso di una semplice interfaccia grafica operativa; questa caratteristica ha molti vantaggi pratici come ad esempio il poter pubblicare un GIS su Web attraverso un leggerissimo Client-Applet java, evitando l'aggravio di dover acquistare ulteriori software per lo scopo. Resta comunque la possibilità di integrare nel Client parte della

gestione software del GIS per “alleggerire” Kosmos in quei casi dove il carico computazionale sul server dovesse risultare elevato.
Kosmos è sviluppato seguendo due linee sequenziali di progetto: le nuove versioni e funzionalità vengono implementate prima in java, dopo una fase di test su tale versione per verificare la bontà delle soluzioni adottate, viene fatto il porting in C++ (linguaggio che consegna software molto prestante ma con elevato tempo di debbugging). Questa soluzione permette di eliminare nel primo passo gli errori “logico-implementativi”, supportare velocemente nuove funzionalità, e, dopo un certo lasso di tempo ritrovare le stesse nella versione più performante.
Il modello di sviluppo è cooperativo: Kosmos è rilasciato sotto licenza GPL, dunque chiunque può contribuire alla scrittura del codice, essendo questo di pubblico dominio; inoltre esso può essere utilizzato a scopo di lucro (qualsiasi azienda o privato può vendere un GIS che avrà bisogno di Kosmos per essere utilizzato). Altra caratteristica comune a molti software GIS è quella di lasciare all'utente poche possibilità per la definizioni di GeoClassi Custom: spesso è consentito definirle soltanto a partire da attributi alfanumerici e non anche da metodi, lo stesso per quelle derivate dove è possibile aggiungere la definizione di elementi alfanumerici. Da ciò ne consegue che il polimorfismo tra GeoClassi definiti dall'utente è relativo ad eventi predefiniti, come l'inserimento di un valore in un dato attributo o il controllo della cardinalità delle relazioni. Per superare questi vincoli occorre mettere mano alla programmazione, la cosa può, più o meno, andar bene per il progettista ma non rende felice l' utente inesperto. Kosmos vuole superare questo limite consentendo la definizione di complesse gerarchie di GeoClassi in maniera facile ed intuitiva, e comunque riutilizzabili: si spera possa nascere, affianco alla comunità che prende parte allo sviluppo, l'abitudine di condividere su internet da parte degli utilizzatori di Kosmos la definizione delle proprie GeoClassi per gli ambiti più disparati, dalle reti tecnologiche alle applicazioni in campo forestale e ad altre situazioni più specifiche e di settore.

Architettura e GeoClassi Custom
Come si diceva il linguaggio utilizzato per lo sviluppo di Kosmos è java, linguaggio molto versatile con piena integrazione col Web, ed “orientato ad Oggetti”: ovvero, come già detto, gli attributi e le funzioni vengono contenuti in oggetti definiti a partire da ***“Classi”.*** Occorre a questo punto sottolineare la differenza tra Classi java inerenti allo sviluppo di Kosmos e le **“GeoClassi”**, che sono definite di volta in volta dal progettista o dall' utilizzatore del GIS e servono per modellare in modo astratto ed il più completo possibile gli elementi geografici specifici di una data applicazione (Alberi, Edifici, Tubi).
In Figura 1, è possibile osservare le Classi fondamentali (complessivamente ve ne sono 29), la gestione delle GeoClassi e la loro definizione. Questo gruppo di Classi deriva da kObjGis che contiene alcuni parametri comuni a tutte le Classi derivate. Nel ramo destro della gerarchia di derivazione si ha kObjAlfa ovvero gli oggetti istanziati sul modello di tale Classe contengo un solo elemento privo di contenuto spaziale (in termini di algebra relazionale sarebbe un record); da questa deriva kFeature che è la classe madre di tutte le GeoClassi, e dunque contiene tutti gli elementi utili alla modellazione di qualsivoglia elemento geografico. Continuando su questo percorso le Classi kFClienti_Point, kFClienti_Polyline, kFClienti_Poligon, specializzano la Classe madre specificando la particolare geometria che caratterizza il contenuto spaziale delle Features, ma, cosa più importante, esse implementano il meccanismo di “caricaggio” delle definizioni delle GeoClassi definiti dall'utente, le quali divengono ***“Classi”*** **derivate da una delle prime tre**, ossia come se fossero state implementate direttamente in codice java! Il vantaggio sta nel poter estendere il meccanismo del polimorfismo intrinseco nel linguaggio di programmazione alle GeoClassi. Ovviamente dalla GeoClasse Custom Radice con la quale si ha “l'innesto” si possono derivare quante GeoClassi si vogliono creando gerarchie (ad albero) comunque complesse. Nel ramo sinistro della gerarchia si trova la Classe kLayers che si occupa delle operazioni relative alle Features di cui è composto (linea tratteggiata). Da quest' ultima vengono derivate due Classi che istanziano quei Layers generati dinamicamente nel GIS a valle di un'operazione di Union, Intersect, ecc... Anche in

questo l'utente può specializzare tali Classi per generare dei nuovi Layers attraverso operatori (Metodi) personalizzati.

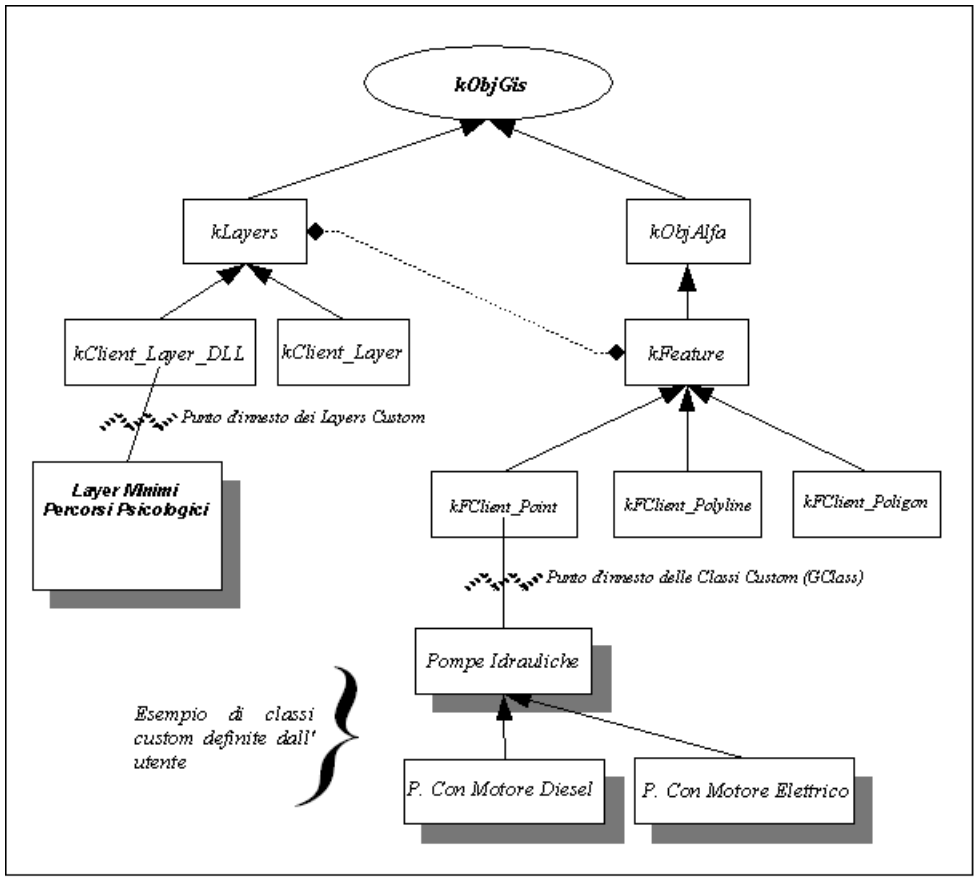

Figura 1- Gerarchia delle Classi e GeoClassi Custom

Il Meccanismo di ancoraggio delle GeoClassi Custom (figura 2) si basa sulla definizione all'interno delle Classi kFClienti_Point, kFClienti_Polyline, kFClienti_Poligon di strutture dati ospitanti i loro attributi alfanumerici, i riferimenti alle GeoClassi con cui si hanno delle relazioni, la definizione dei metodi attraverso stringhe testuali che contengono una combinazione di comandi già predefiniti in Kosmos, la mappatura degli eventi GIS con specifici metodi. Nell'esempio in figura 1, possiamo definire un metodo per il calcolo della potenza dissipata da una pompa con motore diesel e ridefinire lo stesso per quella a motore elettrico: una query per il calcolo della potenza totale dissipata da tutte le pompe di un dato layer contenete "Pompe Idrauliche" riceverà in maniera polimorfica ora un valore ora un' altro a seconda del tipo di pompa. Ritornando alla figura 2 è possibile osservare che le Classi di Innesto per i Layer Custom contengono il metodo CreateLayer che genera un nuovo layer corrente a partire da due layer; anche qui si può costruire tale metodo con combinazioni di istruzioni, oppure attraverso una libreria dinamica costruita con un linguaggio di programmazione come il caso di figura 1, dove tale approccio è obbligatorio in quanto si usa, per esempio, una rete neurale per generare un percorso "Minimo Psicologico" (V. Barrile , F. Cotroneo; **"**Algoritmi avanzati sui geodatabase"; ASITA 2004) a partire da un layer puntuale di origini e destinazioni ed un layer a polyline di percorsi.

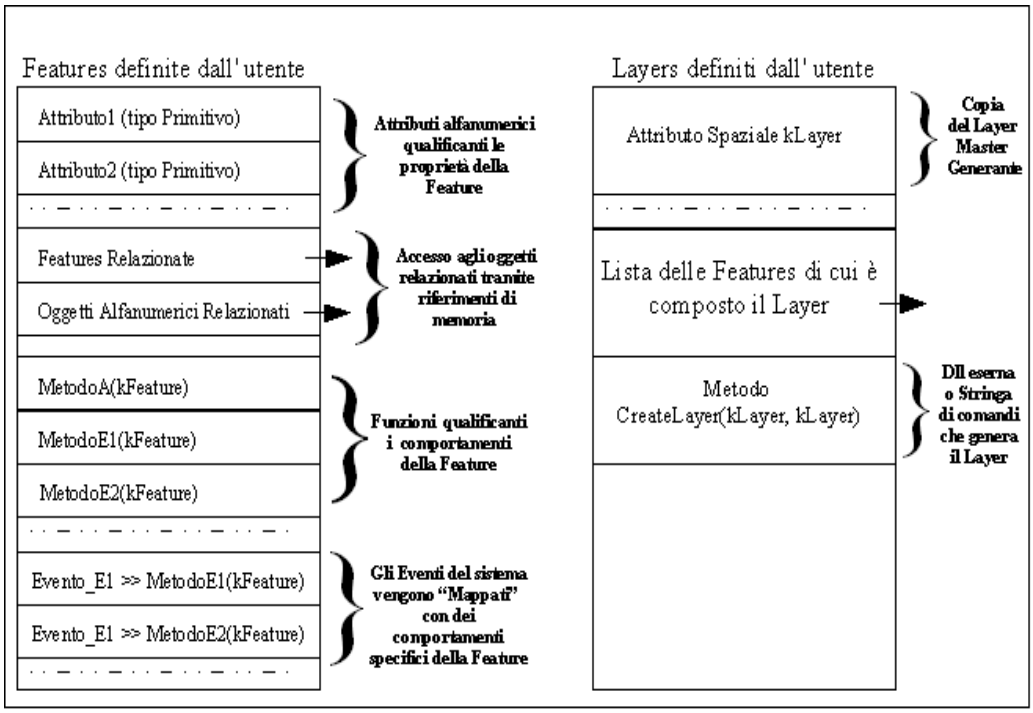

Figura 2 – Strutture dati per l' "Innesto" delle GeoClassi Custom

Analisi Prestazionale

Nella maggior parte delle situazioni in cui operano i GIS, avere un sistema che esegue una data query con una differenza di un decimo di secondo rispetto ad un' altro applicativo non incide sulle scelte aziendali al momento di valutare su quale software GIS basare il proprio business, ma incidono aspetti come il time to market, la facilità di utilizzo, l'alta scalabilità in modo da intervenire con i programmatori il meno possibile. Fare un confronto su questo piano tra Kosmos e altri software non aggiungerebbe molto rispetto a quanto detto, infatti chiunque può fare tale test tra le possibilità offerte dal proprio software GIS e quelle derivanti da approcci affini a quelli adottati da Kosmos. Tuttavia i due aspetti, prestazione e usabilità, sono strettamente correlati, infatti i software che non offrono possibilità di definire complesse gerarchie di GeoClassi Custom con propri metodi particolareggiati, possono permettersi il lusso di usare un RDBMS, mentre per Kosmos, che si trova a gestire metodi polimorfici che possono essere costruiti a partire da operatori

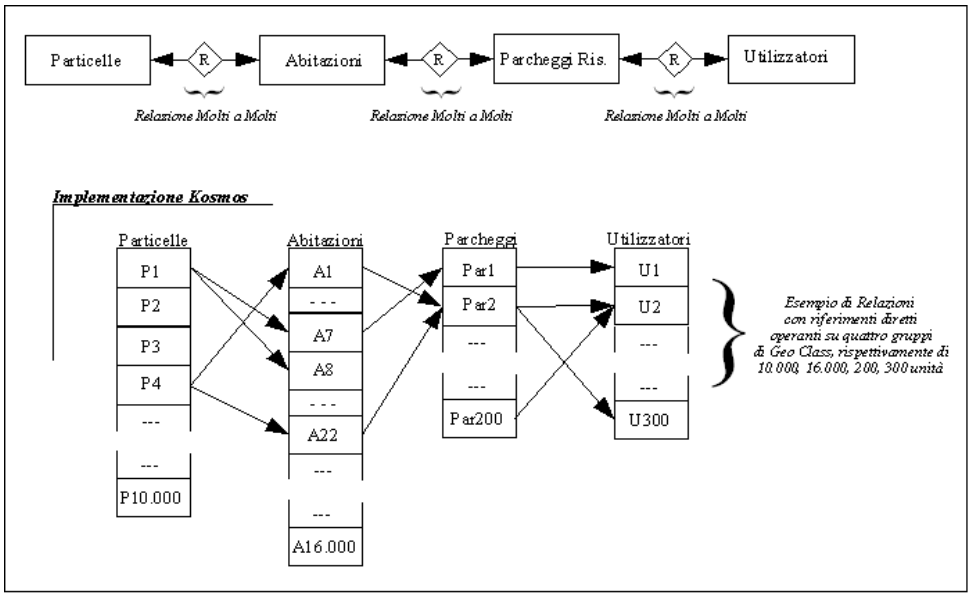

Figura 3 – Esempio di Relazioni Molti a Molti su Kosmos

In figura 3 è schematizzato un' esempio di relazioni collegate a cascata, utile per effettuare un test su query complesse. Si supponga di avere un Gis avente, tra gli altri, tre layers: quello delle particelle cui sono associate delle features del Layer delle Abitazioni, le quali possono aver associati dei parcheggi riservati (terzo Layer). Inoltre ogni parcheggio può essere utilizzato da uno o più utilizzatori autorizzati (lo schema entità relazione è in alto nella figura 3.
Se volessi trovare tutti gli utilizzatori di parcheggi che sono "correlati" a una data particella selezionata dovrei fare una operazione di "Natural Join" (per usare un termine dell'algebra relazionale) su tutte le tabelle/ relazioni. Nel caso di RDBMS le relazioni molti a molti , come quelle in figura, sono rappresentate anch'esse come tabelle. Pertanto la complessità computazionale è data dalla [1]

$$O(n^{2m-1}) \quad [1]$$

$$O(n^{m}) \quad [2]$$

Nel caso di un' approccio OODBMS, ed in particolare per quanto riguarda Kosmos, la complessità computazionale asintotica è data dalla [2] dove *n* è il numero di features (supposto idemtico per ogni Layer) ed *m* il numero di layer o tabelle (nel caso RDBMS). Si ha un guadagno perchè le relazioni sono mappate direttamente con riferimenti di memoria. La query in Kosmos è molto veloce perché sono poche le case associate ad una particella, quindi sono pochi i percorsi da esplorare. Per contro se dato un utilizzatore si vuole trovare la particella associata, ci si avvicina molto al caso (2) che comunque rimane sempre preferibile al caso (1).

Combinazioni	***Q1***	***Q2***
GRASS - mySQL	62 ms	69 ms
GRASS - Kosmos	12 ms (+14)	33 ms (+15)
ArcINFO - Oracle	56 ms	65 ms
Kosmos	11 ms	32 ms

Figura 4 – Confronto tra diverse accoppiate Sever – Client

Conclusioni
Il software GIS denominato Kosmos, implementando il paradigma ad oggetti sia sulla gestione delle GeoClassi sia al momento dello storage, offre risultati apprezzabili sia in fase di modellazione del GIS grazie sulla sua architettura modulare e scalabile, sia dal punto di vista delle prestazioni, Si osservi a tal proposito la tabella di figura 4, dove sulla prima colonna sono riportate varie combinazioni Client/Server utilizzate per i test, sulla seconda e sulla terza i tempi di risposta alla prima e alla seconda query prima esposte. Quando lo Storage è effettuato da un Classico RDBMS i ritardi sono circa identici sia sul primo che sul secondo caso; Kosmos risulta invece due volte più veloce nel secondo caso e addirittura sei volte nel primo; la stessa cosa succede con l'accoppiata GRASS - Kosmos, dove però occorre considerare un certo ritardo di overlay per l'adattamento operato su Grass per garantirne l'interfacciamento.

Bibliografia
V. Barrile , F. Cotroneo; "Storage dei dati nei Geodatabase ", ASITA 2004..
Groff James R., Weinberg Paul N. (2003), La Guida Completa SQL.
Michael Zeiler, (2000) Modeling our World; Esri Press
Drozdek Adam, (2001) Data Structures and Algorithms in Java