UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

ANALISI DELLE PERFORMANCE DEI DATABASE NON RELAZIONALI IL CASO DI STUDIO DI MONGODB

RELATORE: Ch.mo Prof. Giorgio Maria Di Nunzio

LAUREANDO: Luca Merelli

Padova, 21 Settembre 2012

Ringraziamenti

Desidero innanzitutto ringraziare il Professor Giorgio Maria Di Nunzio per aver accettato l'incarico di relatore per la mia tesi.

Inoltre, ringrazio con affetto i miei genitori per il sostegno dato e per essermi stati vicino sia nei momenti felici sia in quelli difficili durante tutto il percorso universitario.

Infine ringrazio i miei amici con cui ho affrontato e condiviso il percorso di studi.

Indice

Elenco delle tabelle

Elenco delle figure

Sommario

Questa tesi ha come obiettivo l'analisi delle prestazioni dei database non relazionali prendendo come caso di studio MongoDB.

I database non-relazionali o NoSQL sono database di nuova generazione nati come alternativa ai database logico-relazionali, sono caratterizzati generalmente da modelli di dati senza uno schema fisso, dotati di nuovi metodi di interrogazione con lo scopo di gestire enormi quantità di dati, rendere agevole la replica e la distribuzione del database su più macchine e per ottenere una maggiore velocità in scrittura.

MongoDB è un database NOSQL, scritto in C++ e sviluppato da 10gen[1], facilmente scalabile, con elevate prestazioni e open source. Il suo modello dei dati è orientato ai documenti raggruppati in collezioni che non impongono uno schema specifico e per questo conserva la stessa potenza di interrogazione dei database relazionali.

Questa tesi si pone l'obbiettivo di fornire una descrizione del funzionamento e delle caratteristiche di MongoDB effettuando un confronto con il database relazionale MySQL. Attraverso lo sviluppo di un database, vengono analizzate le prestazioni di inserimento e interrogazione dei due database per fornire una serie di risultati che andranno a svelare eventuali punti di forza e debolezza di entrambi.

[1] http://www.10gen.com/

Capitolo 1

Introduzione

La tesi ha lo scopo di entrare in contatto con il mondo dei database non relazionali, che si sono sviluppati negli ultimi anni per risolvere il nascere di nuove problematiche dovute all'aumento esponenziale del volume dei dati da gestire in modo veloce dai sistemi di basi di dati. A tale scopo si è preso come caso di studio il database NoSQL MongoDB. Se ne sono studiate le caratteristiche, le funzionalità, il linguaggio di interrogazione e la struttura basandosi su KYLE BANKER, 2012, *MongoDB In Action*, New York, Manning.

Avendo una buona conoscenza dei database relazionali affrontati durante il corso di studi nell'insegnamento di Basi di Dati si è voluto effettuare un confronto tra i database relazionali prendendo come esempio MySQL e il database non relazionale MongoDB.

Prendendo spunto dall'applicazione ecommerce per la vendita di prodotti di giardinaggio, presente nel libro KYLE BANKER, 2012, *MongoDB In Action*, New York, Manning, si è implementato un database con MongoDB e attraverso un processo di reverse engineering dal modello dei dati di MongoDB si è sviluppato lo stesso database implementato con MySQL.

Una volta realizzati i due database si è passati all'analisi delle performance. Testando prima di tutto le prestazioni di inserimento attraverso di diversi metodi di inserimento ad esempio utilizzando i prepared statement per MySQL o i bulk insert per MongoDB. In secondo luogo si sono testate le prestazioni dei due metodi interrogazione. In particolare si sono testate le prestazioni del map-reduce di MongoDB e poi è stata scritta l'interrogazione speculare in SQL da essere applicata a MySQL.

1.1 Strumenti utilizzati

Java

Java[1] è un linguaggio di programmazione orientato agli oggetti utilizzabile gratuitamente su licenza GNU General Public License, creato da James Gosling, Patrick Naughton e altri ingegneri di Sun Microsystems. Java è un marchio registrato di Oracle. Per lo sviluppo dei test è stata usata versione Java SE 7 (1.7.0) rilasciata il 28 luglio 2011 e gli eseguibili sono stati creati su piattaforma Windows Vista.

MySQL

MySQL[2] è un Database Management System, ovvero un sistema per la gestione dei dati. È un progetto Open Source sotto la licenza GPL, GNU General Public License, disponibile gratuitamente per molte piattaforme e permette il salvataggio, la manipolazione e l'estrazione di dati tramite il linguaggio SQL. La versione utilizzata è la 5.5.25 per piattaforme a 32 e 64 bit.

MongoDB

MongoDB[3] è un database NOSQL orientato ai documenti, schema-free, ad alte prestazioni. MongoDB è scritto in C++ e sviluppato da 10gen. Il progetto compila sulla maggior parte dei sistemi operativi: Mac OS X, Windows e Linux. MongoDB è open source e sotto licenza di GNU-AGPL. La versione utilizzata è la 2.0.4 per piattaforme a 32 e 64 bit.

JDBC

L'API Java Database Connectivity (JDBC) è lo standard industriale per la connettività tra il linguaggio di programmazione Java e una vasta gamma di database SQL. La connessione con MySQL è stata ottenuta utilizzando MySQL Connector/J, che è il driver JDBC ufficilie per MySQL. La versione utilizzata è Connector/J 5.1.22[4].

Java Driver per MongoDB

Per ottenere la connessione tra Java e MongoDB è stato utilizzato il seguente driver: Java Driver 2.8.0[5] rilasciato il 18 Giugno 2012.

[1] http://www.java.com/it/
[2] http://www.mysql.it/
[3] http://www.mongodb.org/
[4] http://dev.mysql.com/downloads/connector/j/
[5] https://github.com/mongodb/mongo-java-driver/downloads

1.2 Panoramica dei database NoSQL

I sistemi di gestione di basi di dati relazionali oggi sono la tecnologia predominante per la memorizzare di dati strutturati per applicazioni web e aziendali. Da quando Codds pubblicò l'articolo *A relational model of data for large shared data banks* del 1970, questo metodo di archiviazione di dati è stato ampiamente adottato ed è stato spesso considerato come la migliore alternativa per la memorizzazione di dati.

Negli ultimi anni una serie di nuovi sistemi sono stati progettati per fornire una buona scalabilità orizzontale al fine di ottenere elevate prestazioni nelle operazioni di scrittura/lettura su database distribuiti su più server, in contrasto con i database tradizionali che hanno poche possibilità di essere scalabili orizzontalmente. Con il termine scalabilità orizzontale si intende l'abilità di distribuire i dati e il carico di queste semplici operazioni su molti server, senza RAM o dischi di archiviazione condivisi.

Molti di questi nuovi sistemi sono chiamati NoSQL data stores. La definizione di NoSQL, che sta per "Not Only SQL" o "Not Relational", fu usata per la prima volta nel 1998 per una base di dati relazionale open source che non usava un'interfaccia SQL. Il termine fu reintrodotto nel 2009 dall'impiegato di Rackspace, Eric Evans, quando Johan Oskarsson di Last.fm volle organizzare un evento per discutere di basi di dati distribuite open source. Il nome fu un tentativo per descrivere l'emergere di un numero consistente di sistemi di archiviazione dati distribuiti non relazionali che spesso non tentano di fornire le classiche garanzie ACID riferendosi a sistemi come MySQL, MS SQL e PostgreSQL.

Generalmente i sistemi NoSQL hanno delle caratteristiche in comune che possono essere riassunte nei seguenti punti:

- l'abilità di scalare orizzontalmente semplici operazioni su più server, con il vantaggio di supportare un gran numero di semplici operazioni in lettura/scrittura, questo metodo di elaborazione dati è chiamato OLTP, Online Transaction Processing, che ha lo scopo di supportare un gran numero di piccole transazioni on-line, con un'elevata velocità nell'elaborazione delle interrogazioni, mantenendo l'integrità dei dati in ambienti multi-accesso e l'efficacia di esecuzione di un gran numero di transazioni al secondo.

- l'abilità di replicare e distribuire i dati su più server.

- un modello di concorrenza delle transazioni non basato sulle proprietà ACID della maggior parte dei database relazionali. Alcuni suggeriscono l'acronimo BASE, Basically Available, Soft state, Enetually consistent. L'idea è che rinunciando ai vincoli ACID si possano ottenere prestazioni elevate e maggiore scalabilità.

- un uso efficiente di indici distribuiti e RAM per la memorizzazione dei dati.

- l'abilità di aggiungere dinamicamente nuovi attributi ai record di dati.

1.3 Motivazioni dello sviluppo del movimento NoSQL

Negli ultimi due anni il movimento NoSQL ha attirato un gran un numero di imprese e aziende, che sono passate dai database relazionali agli archivi di dati non relazionali. Alcuni esempi sono Cassandra originariamente sviluppato all'interno di Facebook[6] e oggi usato anche da Twitter[7] e Digg[8], Project Voldemort[9] sviluppato e usato da Linkedln[10], servizi di cloud come il database NoSQL di Amazon SimpleDB[11], il servizio di memorizzazione e sincronizzazione cloud Ubuntu One[12] basato sul database non relazione CouchDB.

La maggior parte dei database NoSQL hanno adottato le idee di Bigtable di Google[13] e Dynamo[14] di Amazon[15], che hanno fornito una serie di concetti che hanno ispirato molti degli attuali data stores.

Come si può vedere i pionieri del movimento NoSQL sono grandi compagnie web o imprese che gestiscono importanti siti web come Google, Facebook e Amazon e altri che in questo campo hanno adottato queste idee per venire incontro alle loro esigenze e requisiti.

Perciò una delle principali motivazioni per l'uso di tali database è la necessità di sviluppare nuove applicazioni che non possono essere affrontate con il tradizionale approccio relazionale.

La ricerca di alternative ai database relazionali può essere spiegata da due principali esigenze:

- la continua crescita del volume di dati da memorizzare
- la necessità di elaborare grandi quantità di dati in poco tempo

Queste nuove necessità sono dovute a molti fattori come l'aumento esponenziale del numero di utenti della rete, la diffusione sempre maggiore dell'OpenID, lo sviluppo di sinergie tra le varie community e i fornitori di servizi, ma anche la crescente disponibilità di dispositivi con accesso ad Internet come smartphone, tablet e altri dispositivi portatili.

[6] http://it-it.facebook.com/
[7] https://twitter.com/
[8] http://www.digg.com/
[9] http://www.project-voldemort.com/voldemort/
[10] http://it.linkedin.com/
[11] http://aws.amazon.com/simpledb/
[12] https://one.ubuntu.com/
[13] https://www.google.it/
[14] http://aws.amazon.com/dynamodb/
[15] http://www.amazon.com/

Per far fronte alla necessità di gestire quantità di dati sempre maggiori e a velocità sempre più elevate i nuovi sistemi di memorizzazione si sono evoluti per essere più flessibili, utilizzando modelli di dati meno complessi, cercando di aggirare il rigido modello relazionale e per aumentare il livello delle prestazioni nella gestione e interrogazione dei dati.

Queste nuove esigenze si connettono al mondo dei sistemi distribuiti. Più precisamente verso la metà degli anni 90, con l'avvento dei grandi sistemi basati su Internet, si iniziò a considerare la disponibilità di servizio da parte di un sistema come un requisito importante. Fino a quel momento infatti, la cosa che più contava era la coerenza e la sicurezza dei dati.

Eric Brewer in un KeyNote alla conferenza "Principle of Distributed Computing" del 2000 presentò il Teorema del CAP. Il teorema, si basa sulle tre lettere che formano la parola CAP, ed ogni lettera è una caratteristica del sistema distribuito su cui si sta lavorando:

- Consistency (Coerenza): dopo una modifica tutti i nodi del sistema distribuito riflettono la modifica.
- Availability (Disponibilità): ad una richiesta, il sistema è sempre in grado di dare una risposta, in altre parole, il sistema è sempre disponibile.
- Partition Tollerance (Tolleranza al Partizionamento): se le comunicazioni si interrompono tra due punti del sistema, il sistema non fallisce ma continua ad essere disponibile.

Secondo il teorema, avere tutte e tre queste caratteristiche in un sistema distribuito in un determinato momento è impossibile, ma si può scegliere quale delle due avere a discapito della terza.

Con il passare del tempo, oltre alla disponibilità di un sistema un'altra caratteristica divenne fondamentale, ovvero la capacità di crescere e decrescere in base alle necessità richieste dagli utenti del sistema, più propriamente chiamata scalabilità orizzontale.

La scalabilità orizzontale si ha se l'aumento delle risorse si riferisce all'aumento dei nodi nel sistema, cioè il sistema riesce a parallelizzare il carico di lavoro. Il vantaggio più importante dato da questo tipo di scalabilità è il costo: infatti, con un'applicazione perfettamente scalabile, si potrebbero impiegare molti nodi a basso costo, ottenendo anche una maggiore prevedibilità del costo marginale. Un altro vantaggio è la maggior tolleranza ai guasti: l'errore in un nodo non pregiudica totalmente il funzionamento dell'applicazione. Gli svantaggi risiedono nei maggiori sforzi in fase di progettazione perché l'applicazione deve innanzitutto supportare questo modello di scalabilità; inoltre l'applicazione deve essere facile da amministrare, per ridurre i costi d'installazione.

Importante è il caso di Amazon, che nel 2007 pubblicò un WhitePaper, nel quale si spiegava il motivo della nascita ed il funzionamento di Dynamo, un Key/Value Store ad alta disponibilità.

Leggendo il WhitePaper, e soprattutto il Blog di Werner Vogels si capisce come il modello relazionale non soddisfaceva le sfide che stava affrontando Amazon, o meglio, si sarebbe potuto adattare il modello relazionale ad Amazon, ma questo avrebbe significato creare una soluzione inefficiente soprattutto dal punto di vista economico.

Tradizionalmente infatti, i sistemi di produzione immagazzinano il loro stato in database relazionali. Ma la maggior parte di questi sistemi immagazzina e legge dati partendo da un'unica chiave primaria e di conseguenza non richiedono tutta la complessità di interrogazione che un database relazionale può essere in grado di fornire, producendo uno spreco di risorse.

La soluzione relazionale è quindi in questo caso sconveniente, ma lo diventa ancora di più quando una compagnia come Amazon deve scalare il proprio sistema, infatti, nonostante i passi in avanti fatti negli ultimi anni, resta complicato aggiungere o togliere un nodo ad un database relazionale distribuito, e questa complessità difficile da abbattere è figlia della potenza che il database relazionale garantisce nella gestione dei dati, dalla sincronizzazione spesso non implementata in modo efficiente che richiede protocolli costosi come il commit a due o tre fasi e cosa più importante dal fatto che i database relazioni inizialmente furono realizzati per applicazioni centralizzate

Amazon affrontò il problema creando una tecnologia apposita per risolvere un determinato tipo di problema, evitando quindi di snaturare un'altra tecnologia, che non era assolutamente nata pensando a questo tipo di problemi.

Amazon, aveva necessariamente a che fare con la tolleranza al partizionamento, e per garantire una migliore esperienza all'utente, scelse di privilegiare le caratteristiche AP del CAP, rinunciando ad avere un sistema sempre consistente. Venne quindi introdotto il concetto di eventualmente consistente, ovvero, in un sistema altamente disponibile e tollerante al partizionamento dei dati, un aggiornamento arriverà eventualmente a tutti i nodi in un determinato lasso di tempo, in modo da non bloccare il sistema nel caso in cui tutti i nodi non abbiano la stessa versione dei dati. Per far questo ovviamente, la soluzione creata da Amazon è dotata di un efficiente sistema di gestione dei conflitti, dato che due nodi in un determinato momento, possono aver ricevuto aggiornamenti diversi tra loro.

Oltre ad Amazon, Google rese pubblico il WhitePaper che spiega il funzionamento di BigTable che fu creato per soddisfare i bisogni specifici di Google sviluppando un sistema diverso dal modello relazionale. BigTable infatti è stato progettato per poter supportare carichi a livello del petabyte attraverso centinaia o migliaia di macchine e di rendere facile l'aggiunta di nuove macchine al sistema, utilizzando le nuove risorse senza che sia necessaria alcuna riconfigurazione.

Dopo gli esempi di Google ed Amazon sono nati altri importanti progetti, quasi tutti open source con lo scopo di creare nuovi strumenti di gestione dei dati che fossero a disposizione di tutti gli sviluppatori. In pochi anni si è aperto un nuovo mercato nel campo dei database nato per affrontare e trovare soluzioni a problemi introdotti dallo sviluppo di nuove applicazioni, progetti e tecnologie che non potevano più essere affrontate utilizzando un singolo strumento come i database relazionali.

1.4 Classificazione e confronto dei database NoSQL

I database NoSQL sono classificati in base al tipo di modello che utilizzano per la memorizzazione dei dati, in particolare possono essere individuate quattro grandi famiglie:

1. Key-Values stores.
2. Column-oriented database
3. Document database
4. Graph database

1.4.1 Key-Values stores

Il modello a chiave-valore si basa su una API analoga ad una mappa dove il valore è un oggetto del tutto trasparente per il sistema, cioè su cui non è possibile fare query sui valori, ma solo sulle chiavi.

Alcune implementazioni del modello chiave-valore permettono tipi di valore più complessi come tabelle hash o liste; altri forniscono mezzi per scorrere le chiavi. Nonostante questi database siano molto performanti per certe applicazioni, generalmente non sono d'aiuto se si necessitano operazioni complesse di interrogazione e aggregazione. Sebbene sia possibile estendere una tale API per permettere transazioni che coinvolgono più di una chiave, ciò sarebbe controproducente in un ambiente distribuito, dove l'utilità di avere coppie chiavi-valore non legate fra loro permette di scalare orizzontalmente in modo molto semplice.

Ci sono molte opzioni disponibili open source, tra le più popolari ci sono Redis and Riak.

Redis

Redis[16] è un database basato sul modello chiave-valore, open source e scritto in C. Redis implementa inserimenti, cancellazioni e operazioni di ricerca, inoltre permette

[16] http://redis.io/

di associare alle chiavi liste e insiemi permettendo operazioni su di essi. Supporta aggiornamenti atomici tramite l'utilizzo di lock e la replicazione asincrona.

Riak

Riak[17] è scritto in Erlang, open source ed è considerato come un database basato su un modello chiave-valore avanzato dotato di maggiori funzionalità rispetto ai classici database chiave-valore, ma con delle carenze rispetto ai database orientati ai documenti. Gli oggetti di Riak vengono recuperati e memorizzati nel formato JSON, quindi possono avere campi multipli come i documenti e gli oggetti sono raggruppati in buckets. Riak non supporta gli indici su qualsiasi campo ma solo sulla chiave primaria, l'unica azione possibile con i campi che non sono chiavi primarie sono la ricerca e la memorizzazione come parte dell'oggetto JSON.

1.4.2 Column-oriented database

I database column-oriented organizzano i dati per colonne in contrapposizione con i database row-oriented, che memorizzano i dati per righe successive. Memorizzare consecutivamente i valori contenuti in ogni colonna consente, per query che coinvolgono pochi attributi selezionati su un gran numero di record, di ottenere tempi di risposta migliori, in quanto si riduce anche di diversi ordini di grandezza il numero di accessi necessari alle memorie di massa; è inoltre possibile applicare efficienti tecniche di compressione dei dati, in quanto ogni colonna è costituita da un unico tipo di dati riducendo lo spazio occupato, aggiungere una colonna è poco costoso, inoltre ogni riga può avere un diverso insieme di colonne permettendo alle tabelle di rimanere piccole senza preoccuparsi di gestire valori nulli.

Cassandra

Cassandra[18] è un DBMS distribuito e open source, sviluppato da Apache per gestire grandi quantità di dati dislocati in diversi server, fornendo un servizio orientato alla disponibilità. Cassandra fornisce una struttura di memorizzazione chiave-valore, con consistenza eventuale. Alle chiavi corrispondono dei valori, raggruppati in famiglie di colonne: una famiglia di colonne è definita quando il database viene creato. Tuttavia le colonne possono essere aggiunte ad una famiglia in qualsiasi momento. Inoltre, le colonne sono aggiunte solo specificando le chiavi, così differenti chiavi possono avere differenti numeri di colonne in una data famiglia.

[17] http://wiki.basho.com/
[18] http://cassandra.apache.org/

1.4.3 Document database

I database orientati ai documenti sono simili a delle tabelle hash, con un unico campo di identificazione e valori che possono essere di qualsiasi tipo. I documenti possono contenere strutture nidificate come ad esempio documenti, liste o liste di documenti. A differenza dei database basati su chiave-valore, questi sistemi supportano indici secondari, replicazione e interrogazioni ad hoc. Ma come gli altri database NoSQL non sono supportate le proprietà ACID per le transazioni.

CouchDB

CouchDB[19] è un progetto dell'Apache del 2008, scritto in Erlang. CouchDB supporta collezioni di documenti con schema libero. Su questi documenti delle funzioni Javascript effettuano operazioni di selezione e aggregazione e forniscono delle rappresentazioni del database chiamate viste che possono essere indicizzate. CouchDB è distribuito e può essere replicato tra più nodi server e sono permesse versioni concorrenti di alcuni documenti e il database è in grado di rilevare conflitti ed è in grado di gestire la loro risoluzione al lato client.

MongoDB

MongoDB è un database open source, sviluppato da 10gen e scritto in C++. MongoDB memorizza documenti in collezioni, fornisce l'uso di indici sulle collezioni e un linguaggio di interrogazione sui documenti dinamica. MongoDB supporta lo sharding automatico, distribuendo documenti su più server, ma non supporta la consistenza globale dei dati come i sistemi relazionali, ma viene garantita la consistenza sulla prima copia aggiornata del documento. MongoDB supporta il comando findAndModify per eseguire aggiornamento atomici e ritornare immediatamente il documento aggiornato. Utilizza il map-reduce, che permette di effettuare aggregazioni complesse su più documenti. La replicazione è effettuata a livello degli shards ed asincrona per aumentare le prestazioni.

1.4.4 Graph database

Un database a grafo è costituito da nodi e relazioni tra nodi, questo tipo di database può essere visto come un caso particolare di un database orientato ai documenti in cui i documenti rappresentano sia i nodi che che le relazioni che interconnettono i nodi. Pertanto nodi e relazioni hanno proprietà per la memorizzazione dei dati. La forza di questo tipo di database è di gestire dati fortemente interconnessi, proprietà che permette un'operazione molto interessante: l'attraversamento, graph traversal,

[19] http://couchdb.apache.org/

che rispetto a una normale query su database chiave-valore, stabilisce come passare da un nodo all'altro utilizzando le relazioni tra nodi.

Neo4j

La forza di Neo4j[20] è quella di essere in grado di gestire dati auto referenziati o comunque strettamente interconnessi, ambito in cui spesso altri database falliscono. Il beneficio di usare un database a grafo è la velocità di attraversare nodi e relazioni per trovare dati rilevanti. Spesso questi database si trovano come implementazione di social network, utilizzati soprattutto per la loro flessibilità.

[20] http://neo4j.org/

Capitolo 2

MongoDB

2.1 Introduzione

MongoDB è un sistema di gestione di basi di dati per applicazioni e infrastrutture internet, scritto in C++ e sviluppato da 10gen. Progettato per essere potente flessibile e scalabile; per combinare le migliori caratteristiche dei database key-value per la loro semplicità, velocità e facilità di essere scalabili e dei database relazionali per il ricco modello di dati e per il potente linguaggio di interrogazione.

MongoDB è un database dotato di un modello di dati orientato ai documenti, sviluppato per ottenere elevate performance in lettura e scrittura e l'abilità di essere facilmente scalabile con failover automatico. L'approccio ai documenti rende possibile la rappresentazione di complesse relazioni gerarchiche tramite l'uso documenti nidificati e array.

Le principali proprietà che fanno di MongoDB una valida alternativa ai database relazionali sono:

- Modello di dati orientato ai documenti:
  - i documenti si associano facilmente ai tipi di dati dei linguaggi di programmazione
  - documenti nidificati e array riducono il bisogno di utilizzare join
  - i documenti sono senza schema per una facile evoluzione
  - non vengono supportati join e transazioni su più documenti per scalare più facilmente
- Elevate prestazioni
  - l'assenza di join consente operazioni di lettura e scrittura più veloci

  - indici con la possibilità di indici sulle chiavi di documenti nidificati
  - scritture del tipo *fire and forget*

- Disponibilità elevata
  - i server replicati con failover e distribuzione dei dati sui nodi automatiche

- Facilmente scalabile
  - sharding automatico
  - letture e scritture sono distribuite sugli shard
  - le interrogazioni sono rese più semplici e veloci dall'assenza di join e transazioni su multi-document
  - letture eventualmente consistenti possono essere distribuite sui server replicati

- Ricco linguaggio di interrogazione

2.2 Filosofia di progettazione

MongoDB è stato progettato con un approccio non relazionale per essere scalabile orizzontalmente su più macchine e per memorizzare ordini di grandezza di dati maggiori rispetto al passato. Il modello dei dati basato su documenti memorizzati in formato JSON/BSON sono facili da codificare, facili da gestire grazie anche all'assenza di uno schema fisso e consentono di ottenere eccellenti performance nelle interrogazioni di raggruppamento di dati.

MongoDB si concentra su quattro aspetti importanti: flessibilità, potenza, velocità e facilità d'uso. Per questo motivo sacrifica certi aspetti come controllo e messa a punto, funzionalità troppo potenti come MVCC, Multiversion Concurrency Control, che richiedono scrittura di codice complicato e logica nello strato applicazione e certe proprietà ACID come le transazioni su documenti multipli.

Flessibilità

MongoDB memorizza i dati in documento di formato JSON. JSON fornisce un ricco modello di dati che si adatta perfettamente ai vari tipi di linguaggi di programmazione e dato che è privo di uno schema fisso, è molto più facile sviluppare e ampliare il modello di dati rispetto a quelli che forzano l'uso di uno schema come i database relazionali.

Potenza

MongoDB fornisce molte proprietà dei database relazionali come gli indici secondari, interrogazioni dinamiche, sorting, ricchi aggiornamenti, upserts, cioè aggiornamenti che avvengono solo se il documento esiste e viene inserito se non esiste e aggregazioni. Questo fornisce un'ampia gamma di funzionalità dei database relazionali assieme alla flessibilità e la capacità di scalare propria di un modello non relazionale.

Velocità e scalabilità

Tenendo dati riferiti assieme in un documento, le interrogazioni sono più veloci rispetto a quelle dei database relazionali dove i dati riferiti sono separati in tabelle multiple che necessitano di essere unite tramite join. MongoDB inoltre permette di scalare i dati del database. L'auto sharding permette di scalare i cluster linearmente aggiungendo più macchine. È possibile aumentarne il numero e la capacità senza tempi di inattività, che è molto importante nel web quando il carico di dati può aumentare improvvisamente e disattivare il sito web per lunghi periodi di manutenzione.

Facilità d'uso

MongoDB è stato creato per essere facile da installare, configurare, matenere e usare. A tal fine, MongoDB fornisce poche opzioni di configurazione e cerca automaticamente, quando possibile, di fare la "cosa giusta" permettendo agli sviluppatori di concentrarsi sulla creazione dell'applicazione piuttosto che perdere tempo in oscure configurazioni di sistema.

2.2.1 Ambiti di utilizzo

MongoDB è adatto in svariati ambiti di utilizzo alcuni esempi sono:

- archiviazione e registrazione di eventi
- sistemi di gestione di documenti e contenuti
- E-commerce. Molti siti usano MongoDB come nucleo per l'infrastruttura e-commerce spesso affiancato con un RDBMS per la fase finale di eleborazione e contabilità degli ordini.
- Gaming
- Le infrastrutture del lato server di sistemi mobili

2.3 Principali caratteristiche e strumenti

2.3.1 Il modello dei dati

Il modello dei dati di MongoDB è orientato ai documenti. Un documento è un insieme di chiavi alle quali sono associati dei valori. I valori possono essere semplici tipi di dati come ad esempio stringhe, numeri o date, ma possono essere anche valori complessi come altri documenti, array o perfino array di documenti.

Un esempio di documento è il seguente:

```
{ _id: ObjectID('4bd9e8e17cefd644108961bb'),
  title: 'Adventures in Databases',
  url:'http://example.com/databases.txt',
  author: 'msmith',
  vote_count: 20,
  tags: ['databases','mongodb','indexing'],
  image: {
    url: 'http://example.com/db.jpg',
    caption: '',
    type: 'jpg',
    size: 75381,
    data: "Binary"
  },
 comments: [
    { user: 'bjones',
      text: 'Interesting article!',
    },
    { user: 'blogger',
      text: 'Another related article is at http://example.com/db/db.txt'
    }
  ]
}
```

Il documento precedente rappresenta tutte le informazioni relative a un articolo su un sito di notizie.

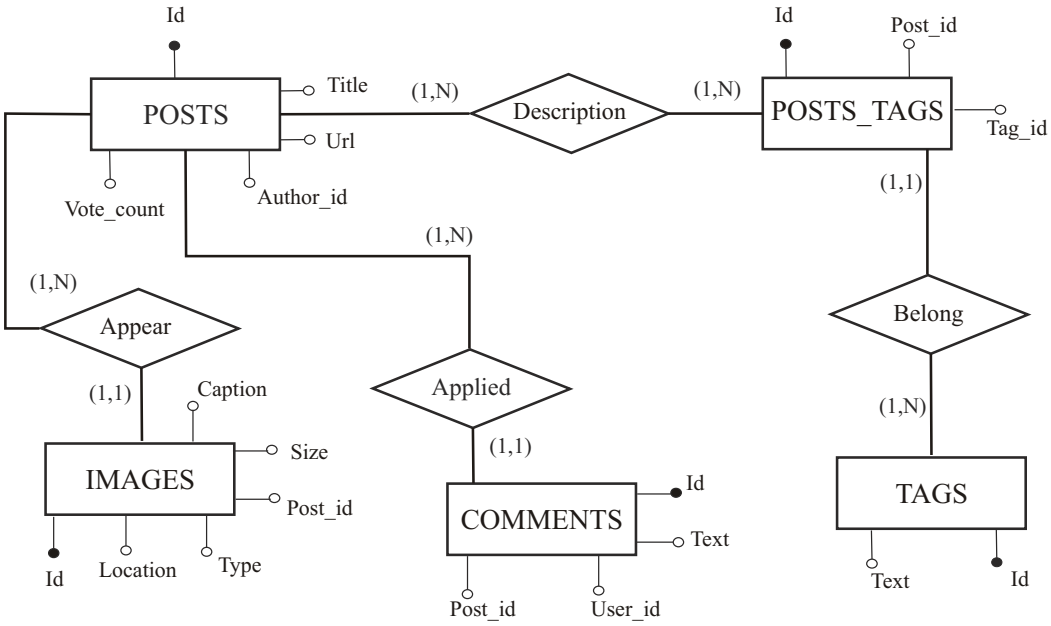

Figura 2.1: Schema ER per rappresentare informazioni di un sito di notizie

Se prendiamo in considerazione il modello logico relazionale un documento del genere sarebbe rappresentato da più tabelle con relazioni tra di esse come nella Figura 2.1; si sarebbe dovuto creare una tabella contenente le informazioni principali del post, poi si sarebbero create le altre tabelle che includevano gli altri campi del post con dei riferimenti al post originale. La tecnica di separare i dati di un oggetto in più tabelle è chiamata normalizzazione. Un insieme di dati normalizzati assicura che ogni unità di dati sia memorizzata in un solo posto, ma per mostrare l'intero post è necessario effettuare dei join tra il post e le tabelle che lo riferiscono, che ovviamente ha un suo costo per quanto riguarda il tempo di esecuzione.

Pertanto una delle differenze sostanziali con il modello logico relazione sta nel fatto che un database orientato ai documenti rappresenta i dati in modo aggregato, permettendo di lavorare con l'oggetto nella sua interezza: tutti i dati che rappresentano un post sono inseriti in un singolo oggetto del database.

Altro aspetto importante da notare è che i documenti non devo essere creati per rispettare uno schema prespecificato. Nei database relazionali si memorizzano righe in tabelle che hanno uno schema ben definito che specifica quali colonne e tipi di

dati sono permessi. Al contrario MongoDB memorizza i documenti in collezioni, che sono semplicemente dei contenitori che non impongono alcuna sorta di schema, i documenti al suo interno possono avere qualsiasi tipo di struttura.

La mancanza di uno schema specifico ha dei vantaggi. Per prima cosa, è l'applicazione che impone la struttra dei dati e i dati stessi, questo può velocizzare la fase iniziale di sviluppo di una applicazione quando lo schema cambia di frequente. Secondo, un modello di dati senza schema permette di rappresentare i dati con varie proprietà in quanto gli attributi di un documento possono essere aggiunti dinamicamente senza dover alterare nessuno schema come invece accade con i database relazionali.

Pertanto riassumendo MongoDB gestisce un insieme di database, ognuno di essi contiene un insieme di collezioni. Una collezione è un insieme di documenti costituiti da un certo numero di campi che sono una coppia chiave/valore. Una chiave è un nome che identifica il campo e il valore può essere un dato qualsiasi.

2.3.2 Linguaggio di interrogazione

MongoDB supporta interrogazioni dinamiche o ad hoc, ovvero un sistema che supporta questo tipo di interrogazioni non necessita di definire in anticipo quali interrogazioni il sistema dovrà accettare, in quanto gli utenti possono trovare i dati usando qualsiasi condizione. In pratica un utente è in grado di scrivere un'interrogazione sempre diversa a seconda dei valori che si cercano, invece in un sistema che non supporta query ad hoc viene creata una query per un'azione generale sul database e vengono passati i valori come delle variabili. Anche i database relazionali supportano questa proprietà, in quanto tramite il linguaggio SQL si possono effettuare interrogazioni con qualsiasi condizione. Uno dei principali obiettivi raggiunto dal progetto MongoDB è stato quello di mantenere la maggior parte della potenza di interrogazione che dei database relazionali.

Per vedere le differenze tra i metodi di interrogazione dei database relazionali e MongoDB si può considerare lo schema ER del sito di notizie della sezione precedente e il relativo documento di MongoDB e si voglia scrivere l'interrogazione che ritorni tutti i posti taggati con il termine 'politics' che hanno più di 10 voti.

L'interrogazione SQL è la seguente:

```
SELECT * FROM posts
INNER JOIN posts_tags ON posts.id = posts_tags.post_id
INNER JOIN tags ON posts_tags.tag_id == tags.id
WHERE tags.text = 'politics' AND posts.vote_count > 10;
```

L'interrogazione equivalente in MongoDB è:

```
db.posts.find({'tags':'politics', 'vote_count':{'$gt':10}});
```

Le due interrogazioni rispecchiano il modello di dati utilizzato. L'interrogazione SQL si affida a un modello normalizzato, dove post e tag sono memorizzati in tabelle differenti, mentre l'interrogazione di MongoDB assume che tag e post siano memorizzati all'interno dello stesso documento. Ma entrambe le interrogazioni hanno la possibilità di effettuare l'interrogazione su una combinazione di attributi arbitraria che è l'essenza di un linguaggio di interrogazione ad hoc.

2.3.3 Indici secondari

Un indice è una struttura dati realizzata per migliorare i tempi di ricerca dei dati. Se non sono utilizzati gli indici ogni ricerca obbliga il sistema a leggere tutti i dati presenti in essa. L'indice consente invece di ridurre l'insieme dei dati da leggere per completare la ricerca. Ad esempio, se si ha un insieme di dati disordinato, è possibile creare un indice in ordine alfabetico, e sfruttare le proprietà dell'ordine alfabetico per arrivare prima al dato o ai dati cercati.

MongoDB fornisce delle veloci performance di interrogazione per mezzo dell'uso degli indici, con una struttura dati che raccoglie informazioni sui valori dei campi indice specificati nei documenti di una collezione. Questa struttura dati è usata dal query optimizer di MongoDB per attraversare velocemente e ordinare i documenti in una collezione. Descrivendoli in maniera formale, questi indici sono implementati come indici B-Tree, concettualmente uguali agli indici utilizzati nei database relazionali.

Si possono creare più di 64 indici per ogni collezione e vengo supportati tutti i tipi di indici utilizzati anche dai database relazionali come gli indici ascendenti e discendenti, gli indici su chiave singola e indici composti da più chiavi, e gli indici geospaziali. Proprietà importante di MongoDB è che si può indicizzare qualsiasi tipo di chiave anche i documenti.

2.3.4 Replicazione

La replicazione è la distribuzione e la manutenzione del database su più macchine con lo scopo di prevenire guasti nell'ambiente in cui il database viene utilizzato, pertanto la replicazione del database è uno strumento che fornisce un'assicurazione contro guasti sulla rete e sul server. Oltre a prevenire i guasti la replicazione in MongoDB è importante per l'aspetto della durevolezza dei dati, in quanto assieme all'utilizzo del journaling la replicazione garantisce di avere una copia dei dati nel caso in cui un nodo sia affetto da un guasto.

MongoDB fornisce due tipi di replica: la replica master-slave e il set di repliche. Per entrambi, un singolo nodo primario riceve tutte le scritture, e poi tutti i nodi secondari leggono e applicano su di essi quelle operazioni in modo asincrono. I due

metodi di replicazione utilizzano lo stesso meccanismo di replica ma hanno rilevanti differenze.

Il metodo di replica master-slave è facile da configurare e ha il vantaggio di supportare qualsiasi numero di nodi secondari, ma ha lo svantaggio di avere un meccanismo di failover completamente manuale, infatti se il nodo master fallisce deve essere un amministratore di sistema a dover spegnere un nodo secondario e riattivarlo come nodo master. Inoltre anche il ripristino è complicato in quanto l'oplog, ovvero la collezione che memorizza la storia di tutte le operazioni sul database, esiste solo sul nodo master e nel caso in cui debba fallire bisogna creare un nuovo oplog sul nuovo master e sincronizzare tutti i nodi figli con esso.

Il set di repliche invece è costituito da un nodo primario e uno o più nodi secondari e garantisce il failover automatico: se per un qualsiasi motivo il nodo principale non è più in linea, allora uno dei nodi secondari sarà automaticamente promosso come nodo primario; quando il precedente nodo primario ritorna in linea viene inserito nel sistema come secondario. Questo metodo fornisce altri miglioramenti, come ad esempio un ripristino più semplice e una distribuzione dei dati più sofisticata.

2.3.5 Velocità e persistenza

MongoDB è stato sviluppato per ottenere un compromesso e trovare un bilanciamento tra velocità in scrittura e persistenza. Con velocità in scrittura si intende il volume di operazioni di inserimento, aggiornamento e cancellazione che un database può elaborare in un determinato periodo di tempo. Persistenza si riferisce alla garanzia che queste operazioni di scrittura siano state effettuate in modo permanente.

In MongoDB gli utenti controllano la velocità e la persistenza delle operazioni scegliendo opportune semantiche di scrittura e decidendo se abilitare il journaling. Tutte le scritture, di default, sono *fire-and-forget*, il che significa che queste operazioni di scrittura sono inviate attraverso un socket TCP senza richiedere una risposta dal database. Se gli utenti richiedono una risposta, possono effettuare le scritture usando la modalità *safe mode*, che forza una risposta e assicura che la scrittura è stata ricevuta dal server senza errori. Il journaling è attivato di default e assicura che di ogni scrittura venga fatto il commit. Se il server viene spento in modo errato, il journaling viene usato per assicurare che i file di dati di MongoDB siano ripristinati in uno stato consistente quando si riattiva il server.

2.3.6 Scalabilità orizzontale

MongoDB è stato progettato per essere scalabile orizzontalmente. In MongoDB lo sharding è l'approccio per scalare orizzontalmente, che permette di dividere i dati e memorizzare partizioni di essi su macchine differenti, questo consente di memorizzare

più dati e gestire più carico di lavoro senza richiedere l'utilizzo di macchine più grandi e potenti.

In particolare MongoDB supporta l'auto-sharding, che elimina alcuni dei problemi amministrativi del carico di lavoro dovuto allo sharding manuale. Infatti i dati vengono distribuiti e bilanciati in modo automatico sui diversi shard, che sono dei contenitori che gestiscono un sottoinsieme di una collezione di dati, uno shard è visto anche come un singolo server o un insieme di repliche del database.

L'auto-sharding divide le collezioni in partizioni più piccole e le distribuisce tra i diversi shard, così che ogni shard diventa responsabile di un sottoinsieme dei dati totali. Dato che risulta essere conveniente che l'applicazione sia all'oscuro di che che shard contenga i dati su cui sta lavorando, viene attivato un processo chiamato `mongos` che è una sorta di router che sa dove sono suddivisi tutti i dati, così l'applicazione può connettersi ad esso e può emettere richieste nel solito modo.

Quando si attiva lo sharding bisogna anche scegliere una chiave da una collezione e usare i valori della chiave per dividere i dati; questa chiave è chiamata shard key. Ad esempio se si ha una collezione di documenti che rappresentano persone, se si sceglie "name" come shard key, uno shard potrebbe contenere i nomi che cominciano dalla A alla F, uno shard con i nomi dalla G alla P e l'ultimo dalla Q alla Z, se si aggiungono nuovi shard MongoDB bilancia il carico di dati tra i vari shard.

Lo sharding è una buona idea quando ad esempio si termina lo spazio su disco della macchina su cui si lavora, se si vuole scrivere dati con velocità maggiore rispetto alla velocità a cui il server MongoDB lavora e se si vuole mantenere una larga porzione di dati in memoria per aumentare le performance.

2.4 La shell di MongoDB

Il server del database di MongoDB viene attivato tramite un eseguibile chiamato `mongod`. La shell di comando di MongoDB viene attivata caricando l'eseguibile `mongo` che si connette a uno specifico processo `mongod`. La shell è basata sul linguaggio JavaScript ed uno strumento che consente di amministrare e manipolare i dati del database.

Una volta attivata la shell, se nessun database viene specificato all'avvio, la shell seleziona un database di default chiamato `test`. Creare un database o una collezione non è necessario in MongoDB, infatti un database e una collazione vengono creati solo quando viene inserito un documento.

Dato che si sta usando la shell in JavaScript i documenti devono essere specificati nel formato JSON, JavaScript Object Notation. Un esempio di documento è il seguente:

```
{ username: "smith" }
```

2.4.1 Inserire un documento

Per inserire un documento in una collezione si usa il metodo `insert` nel seguente modo:

```
db.users.insert({ username: "smith" })
```

Questa operazione aggiungerà al documento una chiave chiamata `_id`. Questa chiave può essere considerata come la chiave primaria del documento ed è un identificatore unico e globale nel sistema. Ogni documento in MongoDB deve avere un `_id`, se non è presente nel documento ne viene creato uno in automatico.

Quando viene eseguito un inserimento, il driver con cui si sta lavorando converte la struttura dati in BSON, che è un formato binario capace di memorizzare qualsiasi documento di MongoDB come una stringa di byte. Il database riceve il documento in formato BSON, controlla la validità del campo `_id` e che la dimensione del documento non superi i 4MB e salva il documento nel database.

2.4.2 Leggere un documento

Una volta inseriti dei documenti all'interno di una collezione è possibile controllare la loro esistenza con il metodo `find`, che ritorna tutti i documenti all'interno di una collezione o il metodo `findOne`, che ritorna un documento della collezione.

```
db.users.find()
```

Quello che viene ritornato dalla shell è il seguente documento:

```
{ _id: ObjectId("4bf9bec50e32f82523389314"), username: "smith" }
```

Si può aggiungere al metodo `find` un `query selector`, che è un documento che viene confrontato con tutti i documenti della collezione per trovare il documento che si sta cercando.

```
db.users.find({ username: "smith" })
```

2.4.3 Eliminare un documento

Il metodo `remove` elimina uno o più documenti da una collezione. Chiamato senza alcun parametro elimina tutti i documenti di una collezione, se invece si vuole rimuovere un solo documento da una collezione bisogna specificare un criterio di rimozione, come nell'esempio seguente:

```
db.users.remove({username: "smith" })
```

In questo caso viene rimosso il documento con il valore `smith` associato alla chiave `username`.

Se invece si vuole eliminare una collezione assieme a tutti i sui indici bisogna utilizzare il metodo `drop`.

```
db.users.drop()
```

2.4.4 Linguaggio di interrogazione

In questa sezione viene fornito un elenco dei più importanti query selectors.

Query selector

Il modo più semplice per specificare una query è utilizzando un selettore costituito da una coppia chiave valore che viene confrontata con i documenti di una collezione per trovare il documento che si sta cercando.

```
db.users.find({username: "smith"})
```

Valori numerici

Per trovare documenti le cui chiavi contengano valori di una certa gamma si utilizzano gli operatori `$lt`, `$lte`, `$gt` e `$gte` che corrispondono rispettivamente a `<`, `<=`, `>` e `>=`. Nel seguente esempio si cercano tutti gli utenti con età maggiore o uguale a 30 anni:

```
db.users.find({age:{$gte: 30}})
```

Operatori insiemistici

Gli operatori `$in`, `$all` e `$nin` prendono come predicato una lista di uno o più valori.

- `$in` ritorna un documento se uno dei valori elencati trova un corrispondente tra i documenti della collezione.
- `$nin` ritorna un documento solo quando nessuno degli elementi forniti trova un corrispondente.
- `$all` ritorna un documento se ogni valore fornito trova un corrispondente.

Operatori booleani

- `$ne` viene utilizzato per negare uno specifico valore.
- `$not` nega il risultato di un altro operatore di MongoDB o di una regolare interrogazione.
- `$or` esprime l'operatore logico OR.
- `$and` esprime l'operatore logico AND.
- `$exists` viene utilizzato per vedere se un documento contiene una particolare chiave.

Viene fornito un esempio dell'operatore `$not`. Si voglia trovare tutti gli utenti con cognome che non cominci con la lettera B:

```
db.users.find({last_name: {$not: /^B/}})
```

Confronto di documenti nidificati

Il modello di dati di MongoDB permette di memorizzare all'interno di un documento altri documenti, detti documenti nidificati. Si supponga di avere il seguente documento che corrisponde a un prodotto dove la chiave `details` punta a un documento nidificato.

```
{ _id: ObjectId("4c4b1476238d3b4dd5003981"),
  slug: "wheel-barrow-9092",
  sku: "9092",
  details: {
      model_num: 4039283402,
      manufacturer: "Acme",
      manufacturer_id: 432,
      color: "Green"
     }
}
```

Se ad esempio si vogliono trovare tutti prodotti forniti dall'azienda Acme, bisogna usare la seguente interrogazione:

```
db.products.find({"details.manufacturer_id": 432})
```

Il punto tra le chiavi `details` e `manufacturer` indica al sistema di interrogazione di cercare una chiave chiamata `details` che punta a un oggetto con una chiave interna chiamata `manufacturer_id` e poi trovare il valore della chiave interna.

Array

Gli array sono utilizzati per memorizzare liste di stringhe, oggetti ID e documenti. Si consideri il seguente documento che rappresenta un prodotto:

```
{ _id: ObjectId("4c4b1476238d3b4dd5003981"),
  slug: "wheel-barrow-9092",
  sku: "9092",
  tags: ["tools", "equipment", "soil"]
}
```

Per trovare tutti i documenti che hanno come tag `soil` bisogna scrivere la seguente interrogazione:

```
db.products.find({tags: "soil"})
```

Si nota subito che viene usata la stessa sintassi che si usa per un'interrogazione su un valore di un documento.

Se invece si ha un documento con associata ad una chiave un array di documenti bisognerà usare la notazione a punto, la stessa utilizzata per trovare un documento nidificato.

Opzioni di interrogazione

Quando viene emessa un'interrogazione si possono applicare una serie di opzioni che vincolano ulteriormente il risultato dell'interrogazione. Di seguito vengono elencate le più importanti.

- Le proiezioni servono per selezionare un sottoinsieme di campi per ogni documento che viene ritornato dall'interrogazione. La seguente interrogazione ritorna tutti i documenti della collezione `users` solamente con il campo `username` e il campo `_id`, incluso di default.

  ```
  db.users.find({}, {username: 1})
  ```

- `$slice` permette di trovare un certo numero di documenti all'interno di una collezione, ad esempio i primi 10 documenti.

- `skip` salta un numero specificato di documenti del risultato dell'interrogazione.

- `limit` limita il numero di documenti ritornati da un'interrogazione.

- `sort` ordina in ordine crescente o decrescente il risultato di un'interrogazione in base a uno o più campi dei documenti.

2.4.5 Aggiornare un documento

MongoDB fornisce due modi per aggiornare un documento: si può rimpiazzare completamente il documento oppure si possono utilizzare delle combinazioni di operatori di aggiornamento per modificare uno specifico campo all'interno di un documento, ovvero effettuando un aggiornamento mirato.

Per rimpiazzare completamente un documento, bisogna prima che il database ritorni il documento, modificarlo dal lato client e emettere l'aggiornamento con il documento modificato, come nel seguente esempio.

Supponiamo di avere il seguente documento salvato nella collezione `users`:

```
{ _id: ObjectId("4bf9bec50e32f82523389314"),
  name: "john",
  username: "smith",
  email: "jsmith@gmail.com"
}
```

Si vuole modificare il valore del campo `email` sostituendolo con un nuovo valore. Per prima cosa se non si è in possesso del campo `_id` bisogna procurarselo tramite un'operazione di `find()`, ritornare il documento, modificare localmente il campo email e alla fine passare il documento modificato al metodo `update`.

```
var doc = db.users.findOne({"_id": ObjectId("4bf9bec50e32f82523389314")})
documento.email = "johnsm@gmail.com"
db.users.update({"_id": ObjectId("4bf9bec50e32f82523389314")}, doc )
```

Utilizzando invece il metodo dell'aggiornamento mirato, al metodo update bisogna passare due argomenti. Il primo specifica quale documento aggiornare e il secondo definisce come il documento deve essere aggiornato.

```
db.users.update({"_id": ObjectId("4bf9bec50e32f82523389314")},
                {$set: {email: "johnsm@gmail.com"}})
```

L'approccio dell'aggiornamento mirato di solito consente migliori performance, in quanto si elimina il tempo di andata e ritorno del documento dal server, pertanto questo metodo utilizza meno tempo per serializzare e trasmettere i dati.

2.4.5.1 Upserts

Un upserts è un tipo speciale di aggiornamento. Se non viene trovato nessun documento che corrisponda al query selector specificato, un nuovo documento viene

creato e inserito. Gli attribuiti del nuovo documento sono quelli del documento query selector e del documento per l'aggiornamento. Se invece il documento cercato esiste, viene aggiornato normalmente.

```
db.users.update({"username": "white")},
                {$set: {email: "d.white@gmail.com"}}, true )
```

Nel precedente esempio il terzo argomento indica che si tratta di un upserts, quindi se non viene trovato nessun documento che soddisfi i criteri di ricerca, allora verrà creato il seguente documento:

```
{ _id: ObjectId("50684b9ebbf3a045fb5c0a94"),
  username: "white",
  email: "d.white@gmail.com"
}
```

2.4.5.2 Metodo findAndModify

Il comando `findAndModify` consente di aggiornare atomicamente un documento, in quanto modifica il documento e lo ritorna in una singola operazione, il documento viene ritornato prima che l'aggiornamento venga eseguito. Questo comando può essere utilizzato per gestire code di lavoro, macchine a stati o per implementare delle transazioni di base.

La forma generale è la seguente:

```
db.runCommand( {findAndModify: <collection>,
                <options> })
```

Le opzioni più importanti da inserire sono le seguenti:

- `query`: un documento query selector per trovare il documento che si cerca.
- `update`: un documento che specifica l'aggiornamento da eseguire.
- `remove`: un valore booleano, che quando assume valore true, rimuove l'oggetto e poi lo ritorna.
- `new`: un valore booleano, se true, ritorna l'oggetto modificato come appare dopo che l'aggiornamento è stato applicato.

2.4.5.3 Operatori di aggiornamento

I principali operatori di aggiornamento sono i seguenti:

- `$inc`: questo operatore viene utilizzato per incrementare o decrementare un valore numerico o per sommare o sottrarre.
- `$set`: viene utilizzato per impostare il valore di una chiave in un documento, se la chiave esiste già il valore viene sovrascritto.
- `$unset`: rimuove la chiave fornita come argomento.
- `$rename`: rinomina il nome di una chiave.

Esistono operatori anche per manipolare gli array:

- `$push`: aggiunge un singolo valore a un array.
- `$pushAll`: aggiunge una lista di valori.
- `$addToSet`: aggiunge un valore ad un array ma solo se non esiste già all'interno dell'array. Per aggiungere un insieme di valori con la stessa modalità bisogna utilizzare l'operatore `$addToSet` in combinazione con l'operatore `$each`.
- `$pop`: rimuove l'ultimo elemento inserito in un array.
- `$pull`: rimuove un elemento da un array specificando il valore dell'elemento da rimuovere.
- `$pullAll`: rimuove una lista di valori specificati.

2.4.6 Funzioni di aggregazione

MongoDB fornisce una serie di strumenti di aggregazione che vanno oltre le semplici opzioni di interrogazione.

Count

`count` è il più semplice strumento di aggregazione e ritorna il numero di documenti all'interno di una collezione.

```
db.users.count()
```

Distinct

`distinct` è un comando che produce una lista di valori distinti per una particolare chiave specificata.

```
db.users.distinct("age")
```

Group

`group` è usato per raggruppare documenti con lo scopo di calcolare un valore di aggregazione basato sul loro contenuto. Per far funzionare correttamente la funzione `group` bisogna inserire i seguenti argomenti:

- `key`: è un documento che descrive il campo da raggruppare.
- `initial`: è un documento che verrà usato come documento base per i risultati dell'aggregazione. Quando la funzione `reduce` verrà invocata, questo documento iniziale verrà usato come primo valore dell'aggregazione.
- `reduce`: è una funzione JavaScript che esegue l'aggregazione. Questa funzione riceve due argomenti: il documento corrente che viene iterato e un documento di aggregazione per memorizzare i risultati dell'aggregazione. Il valore iniziale del documento di aggregazione sarà il valore del documento initial.
- `cond`: è un query selector che filtra i documenti su cui avverrà l'aggregazione.
- `finalize`: è una funzione JavaScript che verrà applicata a ogni documento risultante dall'aggregazione prima che sia emesso il risultato finale.

Map-reduce

`map-reduce` è uno strumento di aggregazione più potente e flessibile del comando `group`, ad esempio si può avere un maggior controllo sulla chiave di raggruppamento, si hanno più opzioni di output.

Il funzionamento del `map-reduce` può essere riassunto in due passi. Nel primo viene scritta la funzione `map` che è applicata a ogni documenti della collezione, definisce su quale chiave si stanno raggruppando i documenti e impacchetta i dati necessari per essere elaborati. Questa funzione deve invocare il metodo `emit()` per selezionare le chiavi e i valori da aggregare.

Il secondo passo consiste nello scrivere la funzione `reduce` che riceve una chiave e una lista di valori, tipicamente questa funzione viene iterata sulla lista di valori e li aggrega. Questa funzione ritorna un valore che ha la stessa struttura dei valori forniti nell'array.

2.4.7 Comandi di amministrazione

I comandi di amministrazione di MongoDB forniscono una serie di strumenti per ottenere delle informazioni sul processo `mongod`.

```
show dbs
```

Stampa una lista di tutti i database presenti nel sistema.

```
show collections
```

Dopo aver selezionato un database, il comando stampa una lista di tutte le collezioni presenti nel database.

```
db.stats()
```

Fornisce una serie di informazioni sul database che si sta usando come ad esempio, il numero di collezioni, numero di oggetti, la dimensioni dei dati, la dimensione di memorizzazione dei dati, il numero di indici e la loro dimensione. Il metodo `stats()` può essere applicato anche ad una collezione.

Capitolo 3

Progettazione e realizzazione in MongoDB e MySQL

In questo capitolo verrà illustrato come sono stati creati i due database implementati con MongoDB e Mysql. Nella prima parte verrà spiegato com'è stato sviluppato il database realizzato con MongoDB, nella seconda parte viene descritto com'è si passati da un database orientato ai documenti al database relazionale realizzato con MySQL. Sostanzialmente i due database sono implementati in modo differente ma rappresentano la stessa realtà ovvero un database per un'applicazione e-commerce per la vendita di prodotti da giardinaggio.

Infine nell'ultima parte del capitolo verranno affrontate le principali differenze che distinguono i due metodi di implementazione del database dal punto di vista del metodo di progettazione dello schema e del modello di dati utilizzato per rappresentare i dati.

In generale i database memorizzano informazioni sui prodotti venduti e sugli utenti che accedono all'applicazione. In particolare vengono memorizzate informazioni basilari che descrivono un prodotto come ad esempio gli identificatori, il nome, una breve descrizione, i dettagli relativi alle caratteristiche del modello e il prezzo. I prodotti vengo anche suddivisi in categorie. Per quanto riguarda gli utenti si memorizzano informazioni essenziali per la loro identificazione: nome, cognome, username e password. Si memorizzano inoltre informazioni sui metodi di pagamento e dei recapiti per spedizione della merce.

L'applicazione e-commerce fornisce la possibilità di recensire i prodotti presenti e di votare le recensioni scritte dagli utenti. L'ambito commerciale dell'applicazione è rappresentano dalla memorizzazione di informazioni sugli ordini effettuati dagli utenti.

3.1 Database e-commerce con MongoDB

Per la creazione del database e-commerce si è preso spunto dall'esempio tratto da KYLE BANKER, 2012, *MongoDB In Action*, 1ª ed. New York: Manning pag. 55 nel quale si sono considerate una serie entità di un'applicazione e-commerce e si mostra come possono essere modellate con MongoDB. In particolare vengono implementati prodotti, categorie, utenti, ordini e recensioni.

3.1.1 Documento prodotto

Un documento prodotto viene inserito nella collezione `products`. Dato che MongoDB non forza l'uso di uno schema prefissato, un documento che rappresenta un prodotto avrà spazio al suo interno per qualsiasi attributo richieda la sua descrizione.

Un prodotto contiene delle informazioni di base come `name`, `sku`, `slug` e `description` con il campo `_id` di default. L'acronimo Sku significa Stock Keeping Unit, ovvero un articolo di magazzino identificato con un codice univoco. È presente una chiave `details` che punta a un documento nidificato che contiene varie informazioni sul prodotto che possono cambiare a seconda del prodotto considerato. Per semplicità di progetto verranno considerati sempre gli stessi attributi del documento della chiave `datails`.

La chiave `pricing` punta a un documento con chiavi che indicano il prezzo al dettaglio e il prezzo di vendita; viene fornita la chiave `price_history` a cui è associato un array di documenti che descrive la storia dell'evoluzione del prezzo del prodotto.

Alla chiave `category_ids` è associato un array di oggetti ID, ognuno di essi funziona come un puntatore al campo `_id` di un documento categoria in modo da implementare una relazione molti a molti, un modo alternativo per sopperire alla mancanza dei join in MongoDB.

```
{ _id: new ObjectId("4c4b1476238d3b4dd5003981"),
  slug: "wheel-barrow-9092",
  sku: "9092",
  name: "Extra Large Wheel Barrow",
  description: "Heavy duty wheel barrow...",
  details: {
     weight: 47,
     weight_units: "lbs",
     model_num: 4039283402,
     manufacturer: "Acme",
     color: "Green"
  },
  pricing: {
```

```
   retail: 589700,
   sale: 489700,
},
price_history: [
    { retail: 529700,
      sale: 429700,
      start: new Date(2010, 4, 1),
      end: new Date(2010, 4, 8)
    },
    { retail: 529700,
      sale: 529700,
      start: new Date(2010, 4, 9),
      end: new Date(2010, 4, 16)
    },
],
category_ids: [new ObjectId("6a5b1476238d3b4dd5000048"),
                  new ObjectId("6a5b1476238d3b4dd5000049")],
main_cat_id: new ObjectId("6a5b1476238d3b4dd5000048"),  }
```

In questo primo documento si sono viste due importati caratteristiche del modello di dati di MongoDB. La prima è la possibilità di memorizzare all'interno di un documento altri documenti e array di documenti, che è una delle più importanti innovazioni portate dal modello di dati di MongoDB, è va sotto il nome di embed. La seconda è il riferimento tra documenti diversi, link, che concettualmente si avvicina molto all'utilizzo delle chiavi esterne nei database relazionali.

L'operazione di riferire un documento può essere effettuata in due modi diversi. Il metodo più semplice e diretto per riferire i documenti è farlo manualmente memorizzando l'`_id` del documento riferito nel documento che si sta attualmente inserendo o modificando; in alternativa all'intero `_id` è possibile inserire qualsiasi altro termine che identifichi il documento riferito. È il caso del campo `main_cat_id` e del campo `category_ids` che implementano rispettivamente una relazione uno a molti e molti a molti.

Il secondo metodo, più formale, per riferire un documento è tramite l'utilizzo di DBRef. Con DBRef il riferimento è memorizzato come un documento innestato nella forma seguente:

```
{ $ref : <collectionname>, $id : <id value>[, $db : <database name>] }
```

`<collectionname>` rappresenta la collezione riferita e `<id value>` rappresenta il valore dell'_id del documento a cui si fa riferimento.

Entrambi i metodi non garantiscono l'esistenza del documento riferito, infatti è possibile inserire un `_id` di un documento che non esiste senza che il database dia alcuna segnalazione. Infine il metodo che utilizza DBRef memorizza semplicemente un sotto-documento, che prima di tutto è uno spreco di spazio rispetto alla memorizzazione di un semplice campo aggiuntivo, e inoltre non si sta effettivamente referenziando una collezione nel senso relazionale del termine, in quanto MongoDB non supporta join e non è possibile de-referenziare un documento al volo, bisogna effettuare una seconda interrogazione al database.

3.1.2 Documento categoria

Un documento categoria rappresenta una categoria a cui uno o più prodotti possono appartenere. All'interno del documento sono memorizzati dei campi standard come `_id`, `slug`, `name` e `description`. Dato che le categorie verranno rappresentate come una gerarchia, alla chiave `parent_id` viene associata la categoria padre del documento e alla chiave `ancestor` è associato un array di documenti che contiene informazioni sulle categorie antenate del documento.

```
{ _id: new ObjectId("6a5b1476238d3b4dd5000048"),
  slug: "gardening-tools",
  ancestors: [{ name: "Home",
                _id: new ObjectId("8b87fb1476238d3b4dd500003"),
                slug: "home"
              },
              { name: "Outdoors",
                _id: new ObjectId("9a9fb1476238d3b4dd5000001"),
                slug: "outdoors"
              }
  ],
  parent_id: new ObjectId("9a9fb1476238d3b4dd5000001"),
  name: "Gardening Tools",
  description: "Gardening gadgets galore!", }
```

3.1.3 Documento utente

Un documento della collezione `users` rappresenta un utente dell'applicazione e-commerce. Nel documento sono contenuti campi che indicano username, password, email, nome e cognome. Alla chiave `address` è associato un array di documenti che indicano i vari indirizzi di spedizione dove per ciascuno di essi viene indicato il

nome, la via, città, stato e zip. Infine la chiave `payment_methods` indica i metodi di pagamento utilizzati dall'utente.

```
{ _id: new ObjectId("4c4b1476238d3b4dd5000001"),
  username: "kbanker",
  email: "kylebanker@gmail.com",
  first_name: "Kyle",
  last_name: "Banker",
  hashed_password: "bd1cfa194c3a603e7186780824b04419",
  addresses: [
    { name: "home",
      street: "588 5th Street",
      city: "Brooklyn",
      state: "NY", zip: 11215},
    { name: "work",
      street: "1 E. 23rd Street",
      city: "New York",
      state: "NY", zip: 10010}
  ],
  payment_methods: [
    { name: "VISA",
      last_four: 2127,
      crypted_number: "43f6ba1dfda6b8106dc7",
      expiration_date: new Date(2014, 4)}
  ] }
```

3.1.4 Documento ordine

Un documento della collezione `orders` contiene un campo `user_id` che memorizza un certo `_id` di un utente in modo da implementare una relazione uno a molti tra la collezione `users` e `orders`. Infatti nell'applicazione e-commerce un utente può effettuare un numero indefinito di ordini e ogni ordine è effettuato da un unico utente.

Inoltre nel documento vengono anche memorizzati lo stato dell'ordine e la data di acquisto. Alla chiave `line_items` è associato un array contenente un elenco dei prodotti acquistati dall'utente e in particolare per ogni prodotto si vuole sapere il nome, sku, la quantità e il prezzo, oltre alla chiave `_id` che punta a un certo prodotto della collezione `products`. Infine viene indicato l'indirizzo di spedizione dell'utente che ha effettuato l'ordine e la chiave `sub_total` che indica il prezzo totale dell'ordine.

```
{ _id: ObjectId("6a5b1476238d3b4dd5000048")
  user_id: ObjectId("4c4b1476238d3b4dd5000001")
  purchase_data: new Date(),
  state: "CART",
  line_items: [
     { _id: ObjectId("4c4b1476238d3b4dd5003981"),
       sku: "9092",
       name: "Extra Large Wheel Barrow",
       quantity: 1,
       pricing: {
         retail: 5897,
         sale: 4897, }
     },
     { _id: ObjectId("4c4b1476238d3b4dd5003981"),
       sku: "10027",
       name: "Rubberized Work Glove, Black",
       quantity: 2,
       pricing: {
         retail: 1499,
         sale: 1299 }
     }
  ],
  shipping_address: {
     street: "588 5th Street",
     city: "Brooklyn",
     state: "NY",
     zip: 11215 },
  sub_total: 6196 }
```

3.1.5 Documento recensione

Un documento nella collezione `reviews` rappresenta una recensione. Una recensione è scritta da un utente su un determinato prodotto, questa relazione è rappresentata dalle chiavi `product_id` e `user_id` che memorizza rispettivamente l'`_id` del prodotto recensito e dell'utente che ha scritto la recensione. Nel documento ci sono informazioni standard come la data di realizzazione della recensione, il titolo, il testo e la valutazione. È inserita anche una la chiave `username` dell'utente come informazione aggiuntiva oltre l'`user_id`.

In un documento recensione vengono memorizzati anche i voti associati alla recensione. Alla chiave `voter_ids` è associato un array di `_id` degli utenti che hanno votato la recensione e con la chiave `helpful_votes` si memorizza il numero di votanti.

```
{ _id: new ObjectId("4c4b1476238d3b4dd5000041"),
  product_id: new ObjectId("4c4b1476238d3b4dd5003981"),
  date: new Date(2010, 5, 7),
  title: "Amazing",
  text: "Has a squeaky wheel, but still a darn good wheel barrow.",
  rating: 4,
  user_id: new ObjectId("4c4b1476238d3b4dd5000041"),
  username: "dgreenthumb",
  helpful_votes: 3,
  voter_ids: [ new ObjectId("4c4b1476238d3b4dd5000041"),
               new ObjectId("7a4f0376238d3b4dd5000003"),
               new ObjectId("92c21476238d3b4dd5000032") ] }
```

3.2 Database e-commerce con MySQL

Il database realizzato con MySQL sostanzialmente rappresenta la stessa realtà descritta dal database realizzato con MongoDB, ovvero la realizzazione di un'applicazione e-commerce per la vendita di prodotti da giardinaggio. Il database è stato ottenuto con un processo di reverse engineering cominciato con lo studio dei dettagli e del funzionamento del database realizzato con MongoDB da cui si sono ottenuti una insieme di requisiti che individuano i problemi che l'applicazione deve affrontare, che poi sono stati organizzati per ottenere i requisiti strutturati. Dalla definizione dei requisiti strutturati si è passati alla progettazione concettuale con la creazione di uno schema concettuale da cui è stato ottenuto uno schema logico tramite la progettazione logica. Infine è stata affrontata la fase di progettazione fisica implementata con MySQL.

3.2.1 Requisiti strutturati

Un prodotto viene identificato univocamente con uno speciale ID e per ogni prodotto si rappresenta: nome, slug, sku e una descrizione. Per ogni prodotto si vuole tenere conto del prezzo che possiede attualmente e dei prezzi che ha avuto in precedenza, inoltre si vogliono sapere le caratteristiche di ogni modello di un determinato prodotto come il peso, colore, unità di peso e costruttore. Per ogni prodotto si vuole

conoscere anche la categoria a cui appartiene; un prodotto può appartenere a più di una categoria.

Ogni categoria viene identificata univocamente con un ID, di cui si vuole memorizzare il nome, descrizione, slug e la categoria immediatamente superiore.

Un utente viene identificato univocamente con un ID, si conosce inoltre il nome, cognome, username, password e email. Per ogni utente si vogliono conoscere i recapiti per la spedizione della merce, di cui si vuole sapere la strada, nome, città, stato, zip e inoltre per ogni utente si voglio sapere i metodi di pagamento, di cui si vuole sapere il nome, last_four, la data di scadenza, crypted_number.

Un ordine viene identificato con un ID, si conoscono inoltre la data di realizzazione dell'ordine e il suo stato. Un ordine viene effettuato da un unico utente e può comprendere un numero indefinito di prodotti di cui si vuole sapere la quantità.

Una recensione viene scritta da un utente su un unico prodotto, per ogni recensione si vogliono memorizzare il titolo, la data, il testo e la valutazione. Una recensione inoltre può ottenere dei voti dagli utenti.

3.2.2 Progettazione concettuale

La strategia utilizzata nella progettazione concettuale della base di dati è la strategia mista. Essa infatti permette di suddividere lo schema iniziale nei concetti fondamentali ed analizzarli distintamente, tramite la strategia bottom-up, attraverso la definizione di uno schema scheletro che verrà via via raffinato tramite la strategia top-down.

Il modello concettuale utilizzato è il modello Entità-Relazione. Come risultato della progettazione concettuale si è ottenuto lo schema Entità-Relazione di Figura 3.1.

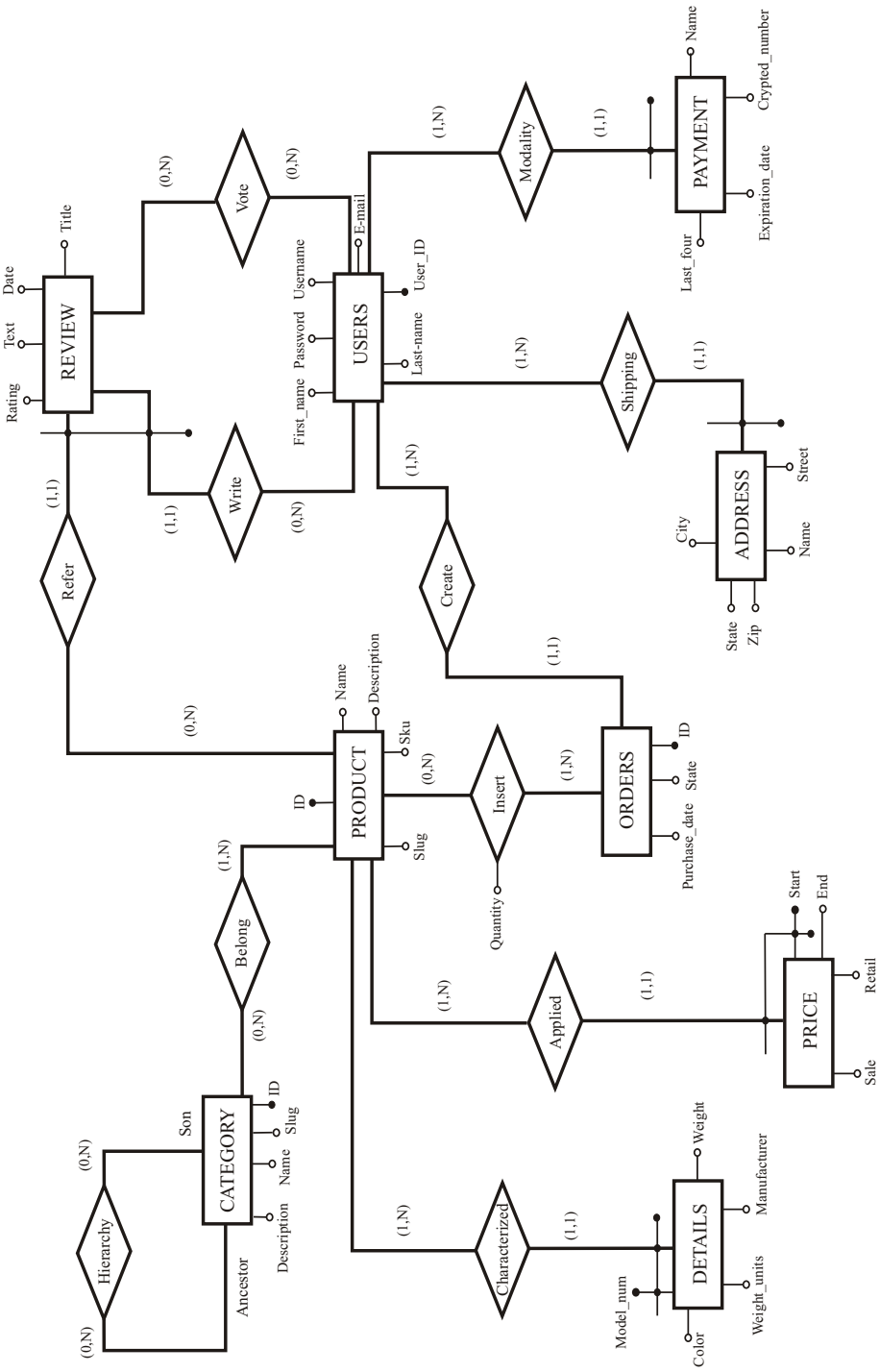

Figura 3.1: Schema ER e-commerce

3.2.3 Progettazione logica

Dato che il precedente schema Entità-Relazione non necessita di una fase di ristrutturazione per semplificare la fase di traduzione verso il modello logico si è passati alla fase di progettazione logica che consiste nella traduzione tra modelli di dati diversi. A partire dallo schema E-R attraverso opportune operazioni di traduzione si costruirà uno schema logico equivalente. Il modello logico utilizzato nel nostro caso sarà il modello logico relazionale. Alla fine di questa fase avremo un schema logico che rappresenterà le stesse informazioni di quello E-R.

Come risultato della fase di progettazione logica si è ottenuto lo schema logico relazione di Figura 3.2.

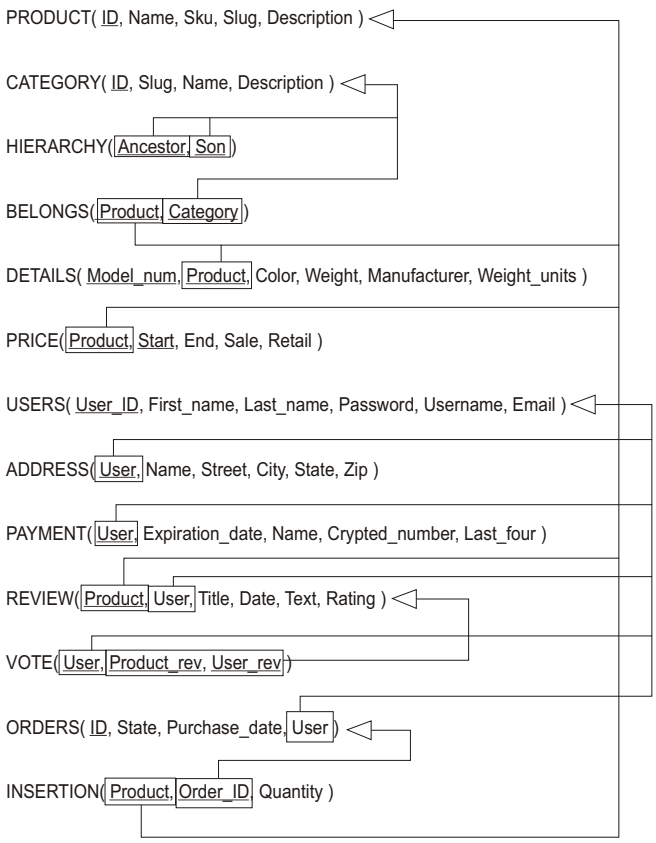

Figura 3.2: Schema logico relazionale

La formulazione in SQL dello schema della base di dati e dei vincoli di integrità viene riportata in Appendice A.

3.3 Principi di progettazione

La progettazione di un database è un processo nel quale viene scelta la migliore rappresentazione della realtà che si vuole descrivere, della natura dei dati e dei requisiti che l'applicazione deve soddisfare.

Come si è visto nella sezione precedente i principi di progettazione di un database relazionale sono ben stabiliti e seguono una serie di regole ben precise. Il modello relazionale si basa su due concetti, relazione e tabella, di natura diversa ma facilmente riconducibili l'uno all'altro. La nozione di relazione proviene dalla matematica, in particolare dalla teoria degli insiemi, mentre il concetto di tabella è semplice e intuitivo. Le tabelle risultano naturali e comprensibili anche per gli utenti finali, d'altra parte la disponibilità di una formalizzazione semplice e precisa ha permesso anche uno sviluppo teorico a supporto del modello.

Inoltre la progettazione di una base di dati segue un determinato procedimento in modo da garantire principalmente la generalità rispetto alle applicazioni e ai sistemi, la qualità del prodotto in termini di correttezza, completezza e efficienza rispetto alle risorse impiegate e la facilità d'uso dei modelli di riferimento.

Nell'ambito delle basi di dati relazionali si è consolidata negli anni una metodologia di progetto che ha dato prova di soddisfare le proprietà descritte. Tale metodologia è articolata in tre fasi: la progettazione logica, il cui scopo è di rappresentare le specifiche informali della realtà di interesse; la progettazione logica, che consiste nella traduzione dello schema concettuale ottenuto dalla fase precedente nel modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati; la progettazione fisica nella quale lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione.

Invece MongoDB manca di regole per la progettazione, in quanto non applica alle collezioni e ai documenti nessun tipo di schema in quanto utilizza prevalentemente uno schema libero. Pertanto la progettazione di uno schema con MongoDB è spesso il risultato di una profonda conoscenza del database che si sta utilizzando, dei requisiti che l'applicazione deve soddisfare, del tipo di dati che si andranno a memorizzare e delle operazioni e aggiornamenti che si dovranno applicare ai dati memorizzati. Da questo si evince che in MongoDB lo schema non è solo in funzione dei dati che si andranno a memorizzare ma bisogna anche focalizzare l'attenzione su come i dati verranno utilizzati dalle applicazioni.

Nonostante la mancanza di regole precise ci sono delle linee guida che possono servire allo scopo di progettare uno schema che soddisfi le nostre esigenze.

Il primo aspetto da considerare è la struttura di un documento di una collezione. In MongoDB non c'è una specifica dichiarazione dei campi all'interno dei documenti di una collezione, in quanto MongoDB non richiede che i documenti abbiano la stessa struttura. Tuttavia nella pratica la maggior parte delle collezioni sono omogenee in modo da facilitarne l'utilizzo sia da parte di progettisti e programmatori ma anche da parte di utenti casuali.

Una domanda chiave che ci si deve porre quando si progetta uno schema con MongoDB è quando incorporare documenti all'interno di altri, embed, e quando collegare i documenti, link. L'incorporamento è l'annidamento di oggetti e array all'interno di un documento BSON. I link sono riferimenti tra documenti. Le operazioni all'interno di un documento sono facili da eseguire e avvengono sul lato server, invece i riferimenti devono essere trattati sul lato client e l'applicazione effettua operazioni su documenti riferiti emettendo interrogazioni dopo che il documento è stato ritornato dal server.

In generale c'è una semplice regola che funziona per la maggior parte degli schemi ma che non deve essere presa come assoluta: incorporare quando gli oggetti figli appaiono sempre nel contesto del genitore, in caso contrario memorizzare gli oggetti figli in una collezione separata.

Prendiamo come esempio un'applicazione che debba memorizzare dei post pubblicati su un sito web e i relativi commenti ai post. La scelta tra incorporare o linkare dipende dall'applicazione. Se i commenti appaiono sempre all'interno di un post e non c'è bisogno di ordinarli in un modo arbitrario, come ad esempio per data o per importanza, incorporare i commenti all'interno di un documento post magari sotto forma di un array di documenti è un'opzione valida in quanto fornisce prestazioni migliori in lettura. Se invece l'applicazione deve essere in grado di mostrare i commenti dal più recente senza considerare su che post i commenti appaiano, allora è meglio usare i riferimenti e memorizzare i commenti in una collezione separata.

Altro aspetto da considerare sono le operazioni atomiche che l'applicazione deve eseguire, l'aspetto chiave in termini di progettazione di schema è che il campo di applicazione delle proprietà atomiche è il documento, pertanto dobbiamo assicurare che tutti i campi rilevanti per le operazioni atomiche siano nello stesso documento.

Di seguito viene presentato un elenco delle opportune regole da seguire durante la progettazione di uno schema:

- gli oggetti che sono di livello superiore come importanza in genere hanno la loro collezione.
- relazioni molti a molti generalmente sono rappresentate con collegamenti.
- collezioni che contengono pochi oggetti possono tranquillamente esistere come collezioni distinte, in quanto l'intera collezione è rapidamente memorizzata nella cache del server dell'applicazione.

- è più difficile ottenere un livello di visualizzazione per documenti incorporati, per ottenere un'operazione del genere si deve utilizzare il map-reduce.
- se le performance sono un problema, incorporare.

Capitolo 4

Analisi delle prestazioni

Sono stati effettuati due tipi di esperimenti sui database implementati con MySQL e MongoDB. Il primo esperimento riguarda una serie di inserimenti con differenti metodologie di esecuzione per consentire di affermare quale dei due database sia più veloce in fase di memorizzazione dei dati. Il secondo esperimento consiste nella creazione di un'interrogazione per testare le performance di estrazione dei dati dal database.

Gli esperimenti sui database sono stati effettuati utilizzando due differenti PC, uno con un sistema operativo a 32 bit e l'altro con un sistema operativo a 64 bit. Di seguito sono riportate le specifiche dei due PC:

- PC numero 1:
  - Produttore: Acer
  - Modello: Aspire 3810T
  - Processore: Intel(R) Core(TM)2 Duo CPU U9400 @1.40 Ghz 1.40 Ghz
  - Memoria(RAM): 4,00 GB
  - Tipo di Sistema: Windows Vista Home Premium a 32 bit
- PC numero 2:
  - Produttore: Hewlett-Packard
  - Modello: Presario CQ56 Notebook PC
  - Processore: AMD Athlon(tm) II P320 Dual-Core Processor 2.10 Ghz
  - Memoria(RAM): 4,00 GB (3,74 GB utilizzabile)
  - Tipo di Sistema: Windows 7 Home Premium a 64 bit

4.1 Inserimenti

Le prove di inserimento sono state effettuate per quanto riguarda MongoDB in un'unica collezione `garden.product` e per quanto riguarda MySQL nella tabella `product`. L'insieme dei valori di ogni riga della tabella `product` occupano 204 Byte, per poter effettuare degli inserimenti il più possibile equiparabili, in MongoDB per la collezione product sono stati considerati solamente i campi `_id`, `slug`, `sku`, `name` e `description`.

Perciò il documento prodotto per le prove di inserimento avrà la seguente forma:

```
{ _id: new ObjectId("4c4b1476238d3b4dd5003981"),
  slug: "wheel-barrow-9092",
  sku: "9092",
  name: "Extra Large Wheel Barrow",
  description: "Heavy duty wheel barrow..."
}
```

4.1.1 Inserimento in MongoDB

I metodi di inserimento utilizzati per il database MongoDB sono i seguenti:

1. Inserimento tramite istruzioni scritte in un file txt
2. Inserimento dei documenti tramite l'utilizzo dei driver Java
3. Utilizzo dei driver Java per effettuare Bulk Insert

Le operazioni di inserimento sono state interrotte in due casi. Quando è stato raggiunto il limite di 2 GB di memorizzazione imposto dal sistema operativo a 32 bit e nel sistema a 64 bit quando le prestazioni di inserimento calavano vistosamente, in linea di massima si è impostato come limite i 30.000.000 di documenti.

4.1.1.1 Metodo con file TXT

Il file txt è stato creato in automatico utilizzando un programma scritto in Java. Viene stampata su file un'operazione di inserimento nella quale sono inseriti dei valori creati in modo casuale e tramite un ciclo for viene iterata questa operazione per il numero di documenti che si vuole inserire.

Con l'utilizzo di questo metodo purtroppo non è possibile ottenere un tempo di esecuzione per ogni operazione o per tutte le operazioni di inserimento, pertanto il tempo di esecuzione è stato rilevato tramite l'utilizzo di un cronometro.

Nella Tabella 4.1 e nella Tabella 4.2 vengono riportati i risultati ottenuti per l'inserimento nei PC con sistema operativo a 32 e 64 bit.

- Inserimento nel PC a 32 bit:

Documenti inseriti	Tempo di esecuzione
10.000	2.83 sec
50.000	11.18 sec
100.000	23.21 sec
200.000	44.23 sec
300.000	1 min 8.78 sec
500.000	1 min 57.44 sec
1.000.000	3 min 27.27 sec
2.000.000	6 min 31.72 sec
3.000.000	9 min 37.98 sec
4.000.000	13 min 57.52 sec

Tabella 4.1: Inserimento MongoDB con file TXT 32 bit

- Inserimento in PC a 64 bit

Documenti inseriti	Tempo di esecuzione
10.000	1.57 sec
50.000	7.09 sec
100.000	15.03 sec
200.000	31.08 sec
300.000	50.43 sec
500.000	1 min 40.07 sec
1.000.000	3 min 01.01 sec
2.000.000	6 min 26.77 sec
3.000.000	9 min 11.43 sec
4.000.000	11 min 52.40 sec
5.000.000	13 min 57.91 sec
8.000.000	28 min 5.72 sec
10.000.000	46 min 25.34 sec
15.000.000	2 h 16 min 43.98 sec

Tabella 4.2: Inserimento MongoDB con file TXT 64 bit

4.1.1.2 Inserimento semplice con driver Java

Per utilizzare le funzionalità fornite dai driver Java prima di tutto bisogna stabilire una connessione con il database e in particolare alla collezione `product` del database

garden nel modo seguente:

```
Mongo mongo = new Mongo();
//accedere al database garden
DB db = mongo.getDB("garden");
//accedere alla collezione products
DBCollection collezione = db.getCollection("orders");
```

L'oggetto `db` è una connessione al server MongoDB al database specificato, con esso è possibile eseguire le operazioni sul database. L'istanza dell'oggetto `Mongo` rappresenta un insieme di connessioni al database, infatti è possibile utilizzare lo stesso un oggetto della classe Mongo anche con più thread.

Una volta effettuata la connessione al database, viene creato un documento vuoto con la seguente istruzione:

```
BasicDBObject documento = new BasicDBObject();
```

E tramite il metodo `put` è possibile inserire i campi desiderati al suo interno. Chiamando poi il metodo insert è possibile inserire il documento nella collezione desiderata:

```
collezione.insert(documento);
```

Nel complesso il programma creato effettua una connessione alla collezione `product` del database `garden` e con un ciclo for si itera, per il numero di volte desiderato, le operazioni di creazione di un documento, con i valori delle chiavi creati casualmente, e del suo inserimento nella collezione. Per ogni inserimento viene calcolato il tempo di esecuzione.

Nella Tabella 4.3 e nella Tabella 4.4 sono riportati i risultati delle varie prove di inserimento per i PC a 32 e 64 bit.

- Inserimento in PC a 32 bit

Documenti inseriti	Tempo di esecuzione
100	93 ms
500	155 ms
1.000	195 ms
5.000	492 ms
10.000	762 ms
50.000	3.02 sec
100.000	5.76 sec
200.000	11.08 sec
300.000	17.03 sec
500.000	27.16 sec
1.000.000	58.64 sec
2.000.000	1min 53.95 sec
3.000.000	2 min 49.61 sec
4.000.000	3 min 39.36 sec

Tabella 4.3: Inserimento MongoDB con driver Java 32 bit

- Inserimento in PC a 64 bit:

Documenti inseriti	Tempo di esecuzione
100	88 ms
500	135 ms
1.000	244 ms
5.000	732 ms
10.000	1.06 sec
50.000	2.52 sec
100.000	4.32 sec
200.000	7.53 sec
300.000	12.12 sec
500.000	22.04 sec
1.000.000	37.20 sec
2.000.000	1 min 21.15 sec
3.000.000	2 min 13.33 sec
4.000.000	2 min 52.73 sec
5.000.000	4 min 17.22 sec
8.000.000	8 min 6.57 sec
10.000.000	15 min 58.65 sec
15.000.000	28 min 16.45 sec
30.000.000	52 min 44.73 sec

Tabella 4.4: Inserimento MongoDB con driver Java 64 bit

4.1.1.3 Bulk Insert tramite driver Java

Utilizzando i driver Java è possibile inserire più documenti attraverso un'unica chiamata al database, questo è possibile tramite l'implementazione dei Bulk Insert, cioè inserimenti di massa.

Il programma creato in Java crea una lista del tipo List<DBObject> con all'interno oggetti di tipo DBObject nel modo seguente:

```
List<DBObject> batchList = new ArrayList<DBObject>();
```

All'interno di un ciclo for si crea un documento con i valori dei campi casuali e si inserisce all'interno della lista con il metodo add. Infine tramite un'unica chiamata al metodo insert viene inserita nel database tutta la lista.

```
collezione.insert( batchList );
```

Questo metodo è stato utilizzato per inserire liste di 1.000, 5.000, 1.0000 oggetti per vedere come variavano le prestazioni di inserimento a seconda della dimensione della lista utilizzata. Nelle seguenti tabelle vengo riportati tutti i risultati di inserimento sui PC a 32 e 64 bit.

- Bulk Insert con 1.000 documenti per lista S.O. a 32 bit:

Documenti inseriti	Tempo di esecuzione
1.000	83 ms
5.000	197 ms
10.000	360 ms
50.000	1.52 sec
100.000	2.99 sec
200.000	5.85 sec
300.000	8.63 sec
500.000	14.51 sec
1.000.000	28.85 sec
2.000.000	57.18 sec
3.000.000	1 min 27.12 sec
4.000.000	1 min 56.05 sec

Tabella 4.5: Bulk insert 1.000 MongoDB 32 bit

- Bulk Insert con 1.000 documenti per lista S.O. a 64 bit:

Documenti inseriti	Tempo di esecuzione
1.000	125 ms
5.000	333 ms
10.000	478 ms
50.000	1.33 sec
100.000	2.34 sec
200.000	4.29 sec
300.000	6.31 sec
500.000	10.56 sec
1.000.000	26.36 sec
2.000.000	1 min 3.31 sec
3.000.000	1 min 51.38 sec
4.000.000	2 min 49.88 sec
5.000.000	4 min 3.96 sec
8.000.000	12 min 14.67 sec
10.000.000	15 min 35.82 sec
15.000.000	28 min 23.08 sec
30.000.000	52 min 31.08 sec

Tabella 4.6: Bulk insert 1.000 MongoDB 64 bit

- Bulk Insert con 5.000 documenti per lista S.O. a 32 bit:

Documenti inseriti	Tempo di esecuzione
5.000	254 ms
10.000	424 ms
50.000	1.63 sec
100.000	3.10 sec
200.000	6.22 sec
300.000	9.44 sec
500.000	15.62 sec
1.000.000	31.05 sec
2.000.000	1 min 2.12 sec
3.000.000	1 min 30.69 sec
4.000.000	2 min 1.19 sec

Tabella 4.7: Bulk insert 5.000 MongoDB 32 bit

- Bulk Insert con 5.000 documenti per lista S.O. a 64 bit:

Documenti inseriti	Tempo di esecuzione
5.000	364 ms
10.000	520 sec
50.000	1.34 sec
100.000	2.46 sec
200.000	4.61 sec
300.000	6.84 sec
500.000	12.08 sec
1.000.000	24.77 sec
2.000.000	1min 18.66 sec
3.000.000	1 min 58.63 sec
4.000.000	2 min 31.76 sec
5.000.000	3 min 28.58 sec
8.000.000	11 min 36.23 sec
10.000.000	13 min 4.74 sec
15.000.000	30 min 42.45 sec
30.000.000	44 min 37.41 sec

Tabella 4.8: Bulk insert 5.000 MongoDB 64 bit

- Bulk Insert con 10.000 documenti per lista S.O. a 32 bit:

Documenti inseriti	Tempo di esecuzione
10.000	398 ms
50.000	1.65 sec
100.000	3.19 sec
200.000	6.35 sec
300.000	9.81 sec
500.000	15.95 sec
1.000.000	34.75 sec
2.000.000	1min 1.45 sec
3.000.000	1 min 33.78 sec
4.000.000	2 min 4.05 sec

Tabella 4.9: Bulk insert 10.000 MongoDB 32 bit

- Bulk Insert con 10.000 documenti per lista S.O. a 64 bit:

Documenti inseriti	Tempo di esecuzione
10.000	478 ms
50.000	1.31 sec
100.000	2.39 sec
200.000	4.95 sec
300.000	7.37 sec
500.000	11.68 sec
1.000.000	23.71 sec
2.000.000	1 min 19.63 sec
3.000.000	1min 57.90 sec
4.000.000	2 min 32.81 sec
5.000.000	3 min 27.88 sec
8.000.000	12 min 21.56 sec
10.000.000	13 min 48.98 sec
15.000.000	23 min 35.23 sec
30.000.000	1 h 2 min 52.11 sec

Tabella 4.10: Bulk insert 10.000 MongoDB 64 bit

4.1.2 Inserimenti in MySQL

I metodi di inserimento per il database MySQL sono i seguenti:

1. Inserimento tramite file CSV
2. Inserimento tramite istruzioni scritte su file txt
3. Utilizzo dei driver Java per implementare i Prepared Statement
4. Utilizzo dei driver Java per implementare i Batch Prepared Statement

Le operazioni di inserimento sono state interrotte quando i tempi di inserimento diventavano eccessivamente lunghi.

4.1.2.1 Inserimenti con file CSV

Il Comma-Separated Values, abbreviato in CSV, è un formato basato su file di testo utilizzato per l'importazione ed esportazione di una tabella di dati. In questo formato, ogni riga della tabella è normalmente rappresentata da una linea di testo, che

a sua volta è divisa in campi separati da un apposito carattere separatore, ciascuno dei quali rappresenta un valore.

Il formato CSV non specifica una codifica di caratteri, né la convenzione per indicare il fine linea, né il carattere da usare come separatore tra campi e nemmeno convenzioni per rappresentare date o numeri, tutti i valori sono considerati come semplici stringhe di testo, è possibile specificare solo se la prima riga è di intestazione o meno. Questi dettagli devono essere specificati dall'utente tutte le volte che si importano o esportano dati in formato CSV in un programma come ad esempio un foglio elettronico.

Il file CSV è stato strutturato nel modo seguente:

```
ID|Name|Sku|Slug|Descriprion
```

Al posto dei campi sono stati inseriti dei valori casuali creati ad hoc. Una volta creato il file CSV è stato caricato dalla shell di comando di MySQL attraverso il comando seguente:

```
LOAD DATA LOCAL INFILE 'nome_file'
INTO TABLE product
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\r\n';
```

Dalle varie prove di inserimento sui PC a 32 e 64 bit sono emersi i risultati di Tabella 4.11 e Tabella 4.12

- Inserimento in PC a 32 bit

Righe inserite	Tempo di esecuzione
100	0.01 sec
500	0.02 sec
1.000	0.06 sec
5.000	0.24 sec
10.000	0.70 sec
50.000	1.91 sec
100.000	3.89 sec
200.000	2 min 17.76 sec
300.000	13 min 21.16 sec
500.000	48 min 25.79 sec
1.000.000	2 h 52 min 22.31 sec
2.000.000	6 h 53 min 35.34 sec

Tabella 4.11: Inserimento MySQL con file CSV 32 bit

- Inserimento in PC a 64 bit:

Righe inserite	Tempo di esecuzione
100	0.13 sec
500	0.14 sec
1.000	0.17 sec
5.000	0.36 sec
10.000	0.45 sec
50.000	1.76 sec
100.000	6.78 sec
200.000	12.35 sec
300.000	13.27 sec
500.000	26.74 sec
1.000.000	52.05 sec
2.000.000	2 min 36.37 sec
3.000.000	11 min 18.10 sec
4.000.000	23 min 34.54 sec
5.000.000	1 h 26 min 43.89 sec
8.000.000	8 h 49 min 41.38 sec

Tabella 4.12: Inserimento MySQL con file CSV 64 bit

4.1.2.2 Inserimenti con file txt

Il file txt è stato creato tramite un programma automatico che stampa l'istruzione di inserimento sul file, all'interno della quale sono inseriti una serie di valori creati casualmente. Tramite un ciclo for queste operazioni vengono ripetute per il numero di volte dichiarato dall'utente in base al numero di righe che si voglio inserire all'interno della tabella.

Il file è stato caricato dalla shell di MySQL con il comando seguente:

```
SOURCE nome_file;
```

Con questo metodo di inserimento non è stato possibile ottenere un tempo complessivo fornito dal database, in quanto si otteneva un tempo di inserimento per ogni operazione, pertanto è stato utilizzato un cronometro per valutare il tempo di esecuzione di ogni prova di inserimento.

- Inserimento in PC a 32 bit:

Righe inserite	Tempo di esecuzione
1.000	2.81 sec
5.000	8.54 sec
10.000	19.59 sec
50.000	1 min 42.67 sec
100.000	4 min 15.24 sec
200.000	9 min 45.69 sec
300.000	25 min 42.08 sec
500.000	1 h 18 min 47.91 sec
1.000.000	3 h 38 min 06.16 sec

Tabella 4.13: Inserimento MySQL con file TXT 32 bit

- Inserimento in PC a 64 bit:

Righe inserite	Tempo di esecuzione
100	4.86 sec
500	19.96 sec
1.000	38.28 sec
5.000	3 min 21.74 sec
10.000	6 min 42.35 sec
50.000	24 min 53.96 sec
100.000	1 h 9 min 12.62 sec
200.000	2 h 23 min 9.33 sec
500.000	7 h 2 min 36.94 sec

Tabella 4.14: Inserimento MySQL con file TXT 64 bit

4.1.2.3 Prepared Statement con driver Java

Utilizzando i driver Java bisogna prima di tutto effettuare una connessione con il database MySQL utilizzando il seguente codice:

```
Connection conn = DriverManager.getConnection
                  ( "jdbc:mysql://localhost:3306/garden?" +
                    "user=root&password=luca");
```

DriverManager è una classe statica che contiene una serie di metodi per gestire la connessione a un database. Quando viene chiamato il metodo `Connection getConnection( String URL )`, il DriverManager decide quale driver JDBC deve usare per connettersi al database.

Il programma creato utilizza i Prepared Statement, grazie ai quali uno statement in SQL viene precompilato e memorizzato in un oggetto PreparedStatement e questo oggetto viene usato per eseguire più volte lo statement SQL. Per creare il Prepared Statement viene utilizzato il metodo `prepareStatement (String sql)` come nel seguente esempio:

```
String query = "insert into product values(?, ?, ?, ?, ?)";
//statement SQL precompilato
PrepareStaement pstmt = connessione.prepareStatement(query);
//oggetto PreparedStatement che contiene lo statement SQL precompilato
```

All'interno dello statement precompilato vengono sostituiti i valori di inserimento con `?`, che è un parametro marcatore utilizzato per indicare dove i valori dei dati che si vogliono inserire sono legati alla query.

Quando il comando SQL, presente nel Prepared Statement, viene inviato al database, quest'ultimo lo compila e rimane in attesa dei dati effettivi che consentano di eseguire il comando stesso. È proprio prima dell'esecuzione del comando che vengono sostituiti i marcatori con i dati: il driver invia i dati al database attraverso l'esecuzione del prepared statement e il database imposta le variabili con i dati ricevuti per eseguire il comando SQL.

Pertanto i marcatori devono essere impostati dopo la creazione dell'oggetto di tipo PreparedStatement ma prima che il comando SQL possa essere eseguito. I metodi, messi a disposizione dall'interfaccia PreparedStatement, hanno la seguente forma: `setTIPO()` dove `TIPO` rappresenta un tipo di dato in Java. Per esempio, il metodo per impostare una Stringa è il seguente:

```
void setString (int parameterIndex, String x)
```

Il primo argomento in input rappresenta l'indice del marcatore all'interno del comando SQL. Il secondo argomento è il valore che andrà a rimpiazzare il marcatore referenziato.

Per dare effettivamente il via all'esecuzione del PreparedStatement bisognerà invocare il metodo `executeUpdate()` sull'oggetto PreparedStatement come nel seguente esempio:

```
pstmt.executeUpdate();
```

Il programma creato per effettuare le prove di inserimento crea un oggetto PreparedStatement e itera l'operazione di sostituzione dei marcatori e delle esecuzione dello statement tante volte quanti sono i dati che si vogliono inserire. Nella Tabella 4.15 e nella Tabella 4.16 sono riportati i risultati delle prove di inserimento.

- Inserimento in PC a 32 bit

Righe inserite	Tempo di esecuzione
100	119 ms
500	562 ms
1.000	1.17 sec
5.000	6.05 sec
10.000	12.05 sec
50.000	1 min 2.06 sec
100.000	2 min 2.56 sec
200.000	5 min 46.93
300.000	23 min 30.88 sec
500.000	1 h 3 min 22.53 sec
1.000.000	3 h 2 min 43.16 sec
2.000.000	9 h 21 min 12.84 sec

Tabella 4.15: Inserimento MySQL con Prepared Statement 32 bit

- Inserimento in PC a 64 bit

Righe inserite	Tempo di esecuzione
100	3.77 sec
500	22.58 sec
1.000	46.82 sec
5.000	3 min 52.53 sec
10.000	7 min 52.54 sec
50.000	34 min 27.32 sec
100.000	1 h 2 min 28.57 sec
200.000	3 h 17 min 22.61 sec

Tabella 4.16: Inserimento MySQL con Prepared Statement 64 bit

4.1.2.4 Batch Prepared Statement con driver Java

L'utilizzo dei Batch Prepared Statement fornisce un meccanismo con cui un gran numero di inserimenti o aggiornamenti possono essere eseguiti in una singola transazione del database per ottenere migliori prestazioni. La differenza sostanziale tra i Prepared Statement e i Batch Prepared Statement è che con i primi era possibile precompilare uno statement SQL, inserire i dati al suo interno e inviare il comando al database, invece con il nuovo metodo è possibile precompilare lo statemente SQL, inserire i dati all'interno dello statement più volte creando così una serie di istruzioni da inviare tutte insieme al database.

Il programma creato in Java crea un oggetto PreparedStatement con uno statement SQL di inserimento, con un ciclo for si iterano le operazioni di sostituzione dei marcatori all'interno dello statement precompilato e di aggiunta dello statement SQL alla coda di batch tramite l'istruzione `void addBatch()`. Infine viene eseguita la coda batch con il metodo `int[] executeBatch()`.

Nella Tabella 4.17 e nella Tabella 4.18 sono riportati i risultati delle prove di inserimento.

- Inserimento in PC a 32 bit

Righe inserite	Tempo di esecuzione
100	47 ms
500	192 ms
1.000	345 ms
5.000	1.68 sec
10.000	2.97 sec
50.000	14.68 sec
100.000	30.41 sec
200.000	1 min 49.27 sec
300.000	8 min 47.10 sec
500.000	1 h 14 min 33.21 sec
1.000.000	3 h 22 min 41.16 sec

Tabella 4.17: Inserimento MySQL Batch 32 bit

- Inserimento in PC a 64 bit

Righe inserite	Tempo di esecuzione
1.000	562 ms
5.000	2.68 sec
10.000	5.01 sec
50.000	22.05 sec
100.000	43.73 sec
200.000	1 min 26.56 sec
300.000	2 min 9.49 sec
500.000	3 min 36.51 sec
1.000.000	7 min 56.08 sec
2.000.000	16 min 30.13 sec
3.000.000	30 min 21.14 sec
4.000.000	52 min 49.92 sec
5.000.000	1 h 46 min 30.52 sec
8.000.000	8 h 4 min 54.91 sec

Tabella 4.18: Inserimento MySQL Batch 64 bit

4.2 Interrogazione dei database

Prima di effettuare le interrogazioni sui database c'è stata una fase preliminare di popolamento dei due database. Questa fase si è resa necessaria prima di tutto per il fatto che l'interrogazione, per quanto riguarda il database MySQL, coinvolge un certo numero di tabelle, ma anche per il fatto che si è voluto vedere come i due database lavorassero con un certo carico di dati al loro interno.

Dopo aver popolato i due database si sono formulate le due interrogazioni che per il risultato che si voleva ottenere sono identiche, ma implementate in modo e con metodologie diverse: per MongoDB è stata usata la funzione map-reduce e per MySQL il linguaggio SQL.

Le interrogazioni sono state prima di tutto valutate separatamente, ovvero i due database sono stati popolati con dati casuali ma diversi e si è valutato il degrado delle prestazioni in interrogazione con l'aumento dei dati. Come secondo approccio si sono popolati i due database con gli stessi identici dati in modo da poter effettuare un confronto di prestazioni tra le due interrogazioni.

L'interrogazione utilizzata per questi esperimenti è stata presa da KYLE BANKER, 2012, *MongoDB In Action*, 1ª ed. New York: Manning pag. 94 - 95 che restituisce per ogni mese il numero di prodotti venduti e il ricavo totale ottenuto dalla vendita dei prodotti.

4.2.1 Popolamento dei database

MongoDB

Per il popolamento del database di MongoDB è stato usato il metodo di inserimento semplice con i driver Java, ovvero viene creato un documento e lo si inserisce con una chiamata al database.

La difficoltà maggiore per popolare in automatico il database sta nel fatto che i documenti delle varie collezioni contengono informazioni di altri documenti di altre collezioni o della stessa collezione, purtroppo in MongoDB non ci sono dei vincoli di integrità referenziale che consentono l'inserimento solo se il documento riferito esiste. Pertanto per garantire che i documenti contengano dei riferimenti validi ad altri documenti, è stato creato un programma che gestisse e creasse dei documenti con riferimenti ad altri documenti effettivamente esistenti.

Questo programma è stato realizzato con l'unico scopo di popolare il database di MongoDB tutto in una volta e a partire dal database vuoto effettuando un certo numero di operazioni in un certo ordine, perciò non vuole essere un programma che possa essere chiamato in qualsiasi momento del ciclo di vita del database.

Il programma realizzato in Java inizia richiedendo di creare cinque ArrayList di oggetti `_id`, una per ogni collezione, che andranno a definire il numero di documenti da inserire per ogni collezione, subito dopo verrà visualizzato un menù con cinque operazioni, ognuna delle quali corrisponde all'inserimento di un numero di documenti pari alla dimensione dell'ArrayList definito in precedenza per una certa collezione. Le operazioni sono cinque, devono essere eseguite in ordine e inseriscono i documenti all'interno delle collezioni rispettivamente in `products`, `category`, `users`, `orders` e `reviews`.

L'idea comune utilizzata per l'inserimento è che alla creazione di un documento si salvino in appositi ArrayList tutte le informazioni che serviranno per i documenti delle collezioni successive. Se un documento ha bisogno di un certo numero di informazioni che provengono da altri documenti basterà semplicemente estrarle in modo casuale o sequenziale dagli ArrayList.

Questo metodo di popolamento presuppone che si sappiano a priori tutte le informazioni che un documento dovrà avere al suo interno, aspetto che in un caso reale potrebbe non essere fattibile, ma come detto in precedenza l'unico scopo del programma è effettuare un popolamento di massa del database con l'intenzione di mantenere dei riferimenti validi e informazioni consistenti tra i documenti di diverse collezioni.

MySQL

Per il popolamento del database di MySQL è stato utilizzato il metodo di inserimento con file CSV in quanto si è rivelato il metodo di inserimento più veloce e anche il

metodo che fornisce più informazioni di inserimento: se la query è andata a buon fine, il tempo di esecuzione dell'inserimento, il numero di tuple inserite e se ci sono state tuple saltate o cancellate durante l'inserimento.

Il programma inizia con la visualizzazione di un menù con un elenco di operazioni, ognuna di esse è un'operazione di inserimento in una tabella del database e viene chiesto all'utente di selezionarne una. Se il database è inizialmente vuoto bisognerà effettuare le operazioni nell'ordine indicato dal menù.

Partendo dall'ipotesi che inizialmente il database sia vuoto, il programma creato popola le tabelle del database con valori generati casualmente e dato che il database deve essere popolato in modo automatico e tutto in una volta è stato utilizzato l'approccio di popolare prima le tabelle senza chiavi esterne e poi di popolare le tabelle che si riferiscono ad altre con chiavi esterne, in modo che durante il primo inserimento dei dati l'ordine delle operazioni indicato dal menù non porti a creare un documento CSV per una tabella che si riferisce ad un'altra che non è stata ancora popolata facendo così fallire l'intero inserimento.

Le prime tre operazioni del menù inserisco un certo numero di tuple stabilite dall'utente all'interno delle tabelle che non hanno chiavi esterne, cioè nelle tabelle `product`, `category` e `users`. Una volta popolate queste tabelle è possibile popolare le altre che riferiscono a queste.

L'idea comune utilizzata per popolare le restanti tabelle è quello di ottenere le chiavi esterne con una query alla tabella riferita tramite l'utilizzo dei driver Java e memorizzarle in un ArrayList. Una volta ottenute tutte le chiavi viene costruito il file CSV, inserendo in modo casuale, le chiavi esterne e gli attributi della tabella in questione. Questo metodo garantisce che i vincoli di integrità referenziali vengano rispettati.

4.2.2 Formulazione delle interrogazioni

MongoDB

L'interrogazione ha lo scopo di ritornare per ogni mese il numero di prodotti venduti e il ricavo totale ottenuto dalla vendita di tali prodotti.

L'interrogazione è stata formulata con uno strumento di aggregazione chiamato `map-reduce`. Il map-reduce è utile per l'elaborazione di insiemi di dati e per operazioni di aggregazione. Il `map-reduce` viene usato in quelle situazioni in cui nel linguaggio SQL si sarebbe utilizzato il `GROUP BY`.

Il `map-reduce` viene invocato tramite un comando del database. Le funzioni `map` e `reduce` sono scritte in JavaScript e eseguite sul lato server. La sinstassi completa del comando è la seguente:

```
db.runCommand(
  { mapreduce : <collection>,
    map : <mapfunction>,
    reduce : <reducefunction>,
    out : <see output options below>
    [, query : <query filter object>]
    [, sort : <sorts the input objects using this key.
               Useful for optimization, like sorting by the
               emit key for fewer reduces>]
    [, limit : <number of objects to return from collection,
                not supported with sharding>]
    [, finalize : <finalizefunction>]
    [, scope : <object where fields go into javascript global scope >]
    [, verbose : true]
  }
);
```

Il `map-reduce` comprende varie opzioni:

- `mapreduce`: a questo campo si associa la collezione su cui effettuare il map-reduce.
- `map`: la funzione mappa è applicata come un metodo a ogni documento della collezione e serve per definire la chiave sulla quale si devono raggruppare i dati e i dati su cui verranno effettuati i calcoli. Dato che la funzione mappa viene applicata a ogni documento della collezione come un metodo, essa ha accesso al riferimento `this`, con cui si può accedere a qualsiasi dato all'interno del documento che la funzione sta processando. La funzione mappa chiama il metodo `emit()` che prende due argomenti: il primo è la chiave su cui si vogliono raggruppare i dati e il secondo argomento è un documento contenente i dati stessi che si vogliono processare nella funzione `reduce`.
- `reduce`: la funzione reduce ha lo scopo di aggregare i vari valori per ogni chiave tramite una funzione definita dall'utente. Alla funzione `reduce` viene passata una chiave e un array di valori emessi dalla funzione mappa, che verranno processati per ottenere un singolo valore come ouput. In pratica l'operazione di riduzione considera una chiave e prende tutti i valori creati dalla funzione mappa e uno ad uno vengono aggregati nel modo desiderato per ritornare un singolo valore. La funzione `reduce` viene invocata più di una volta per la stessa chiave, questo implica che la struttura dell'oggetto ritornato dalla funzione `reduce` deve essere identico al valore emesso dalla funzione mappa.

- `out`: parametro che determina come l'ouput viene ritornato. Se il valore di `out` è una stringa, il risultato del `map-reduce` verrà inserito in una collezione con il nome specificato. `{inline : 1}` - con questa opzione non viene creata nessuna collezione e l'intero `map-reduce` viene eseguito nella RAM e anche il risultato sarà ritornato come risultato del comando stesso. `{ replace : "collectionName" }` - l'output viene inserito nella collezione specificata che rimpiazzerà qualsiasi collezione esistente con lo stesso nome. `{ merge : "collectionName" }` - questa opzione fonderà i nuovi dati ottenuti con i vecchi dati contenuti nella collezione specificata. Nel caso in cui la stessa chiave esista nel risultato e nella vecchia collezione, la nuova chiave verrà sovrascritta alla vecchia. `{ reduce : "collectionName" }` - se esiste un documento nel risultato e nella vecchia collezione per una data chiave, viene applicata la funzione reduce specificata in precedenza sui due valori.
- `query`: un query selector che filtra la collezione su cui verrà applicata la funzione mappa.
- `sort`: un ordinamento applicato all'opzione `query`.
- `limit`: è un valore di tipo integer che specifica un limite che verrà applicato all'opzione `query` e `sort`.
- `finalize`: una funzione Javascript da applicare a ogni documento dopo che la fase di riduzione è stata completata.
- `scope`: un documento che specifica valori che sono globalmente accessibili dalle funzioni `map`, `reduce` e `finalize`.
- `verbose`: valore booleano che quando assume il valore `true` include nel documento ritornato delle statistiche sull'esecuzione del `map-reduce`.

Nel caso della nostra interrogazione la funzione mappa è la seguente:

```
map = function()
      {
        var shipping_month = this.purchase_date.getMonth +
                             '-' + this.purchase_date.getFullYear();
        var items = 0;
        this.line_items.forEach( function(item)
                         {
                            items = items + item.quantity;
                         })
          emit( shipping_month, {total: this.sub_total, items: items });
}
```

Nella prima riga di codice viene creata un variabile che denota il mese e l'anno di creazione dell'ordine, cioè viene definita la chiave su cui raggruppare i dati. Nelle righe successive si calcola la quantità di prodotti realtiva a ogni ordine. Alla fine viene invocata la funzione `emit()` alla quale viene passata la chiave `shipping_month` e un documento che avrà due campi il prezzo totale dell'ordine associato al campo `total` e il numero di prodotti contenuti nell'ordine associato al campo `items`.

Dopo che la funzione mappa è stata applicata si otterrà un insieme di coppie chiave-valore, dove la chiave sarà il mese e l'anno della creazione dell'ordine e il valore è un oggetto JSON contenente i dati che servono per la funzione reduce, ovvero l'importo dell'ordine e il numero totale di prodotti venduti nell'ordine.

La funzione reduce della nostra interrogazione è la seguente:

```
reduce = function( key, values )
         {
            var tmpTotal = 0;
            var tmpItems = 0;
            values.forEach( function(doc)
                   {
                      tmpTotal = tmpTotal + doc.total;
                      tmpItems = tmpItems + doc.items;
                   }
            return( { total: tmpTotal, items: tmpItems });
         }
```

Alla funzione reduce viene passato un array di documenti creati dalla precedente funzione mappa associato a una data chiave. Per ogni documento si prende il valore della chiave total e lo si somma alla variabile temporanea tmpTotal, che serve per tenere aggiornato l'importo totale per gli ordini di un dato mese e lo stesso si fa con il valore del campo items. Questo processo porterà alla creazione di un documento che avrà due campi che rappresentano l'importo totale e il numero totale di prodotti venduti degli ordini per un dato mese.

La funzione map-reduce è stata invocata dalla shell di MongoDB con il seguente comando:

```
db.runCommand( { "mapreduce": "orders", "map": map,
                 "reduce": reduce, "out": "totals" });
```

MySQL

L'interrogazione SQL applicata al database deve restituire per ogni mese il numero di prodotti venduti e il ricavo totale ottenuto dalla vendita di tali prodotti.

```
SELECT     Year(Purchase_date) AS Year, Month(Purchase_date) AS Month,
           SUM(Quantity) AS TotQuantity, SUM(Sale*Quantity) AS TotSale
FROM       product
           JOIN price ON product.ID = price.Product
           JOIN insertion ON product.ID = insertion.Product
           JOIN orders ON insertion.Order_ID = orders.ID
WHERE      price.End = '0000-00-00'
GROUP BY   Year(Purchase_date), Month(Purchase_date);
```

4.2.3 Risultati delle interrogazioni

Come primo test si sono popolati i database separatamente ovvero senza che i dati fossero gli stessi all'interno dei due database, in modo da valutare il loro comportamento all'aumentare del carico di dati memorizzati. Come secondo test si sono popolati i due database in modo da avere al loro interno gli stessi dati, ovvero a un documento inserito in MongoDB corrispondono gli stessi dati in MySQL divisi nelle rispettive tabelle, in questo modo si possono confrontare le prestazioni di interrogazione a parità di dati inseriti nei due database.

MongoDB

L'interrogazione è stata applicata ad entrambi i PC con sistema operativo a 32 e 64 bit e nella Tabella 4.19 e nella Tabella 4.20 vengono riportati i risultati. Viene riportata una colonna chiamata `Documenti inseriti` che indica il numero di documenti inseriti per ogni collezione; con la colonna `Carico` viene indicato il numero totale di documenti presenti nel database; la colonna `Popolamento database` indica il tempo impiegato per il popolamento del database; la colonna `Esecuzione query` indica il tempo di esecuzione dell'interrogazione sul database.

- Interrogazione PC a 32 bit:

Documenti inseriti	Carico	Popolamento database	Esecuzione query
100	500	125 ms	53 ms
500	2.500	561 ms	126 ms
1.000	5.000	1.32 sec	249 ms
5.000	25.000	26.96 sec	1.14 sec
10.000	50.000	4.43 sec	2.36 sec
50.000	250.000	35.69 sec	11.34 sec
100.000	500.000	1 min 8.52 sec	22.49 sec
200.000	1.000.000	2 min 20.87 sec	45.30 sec

Tabella 4.19: Risultati interrogazione MongoDB 32 bit

- Interrogazione PC a 64 bit:

Documenti inseriti	Carico	Popolamento database	Esecuzione query
100	500	384 ms	211 ms
500	2.500	818 ms	218 ms
1.000	5.000	1.30 sec	382 ms
5.000	25.000	3.33 sec	1.76 sec
10.000	50.000	5.83 sec	3.52 sec
50.000	250.000	18.75 sec	16.85 sec
100.000	500.000	36.45 sec	34.20 sec
300.000	1.500.000	2 min 2.40 sec	1 min 45.32 sec
500.000	2.000.000	10 min 53.19 sec	3 min 5.47 sec
1.000.000	5.000.000	20 min 37.23 sec	6 min 8.55 sec
2.000.000	10.000.000	38 min 55.21 sec	12 min 24.34 sec
3.000.000	15.000.000	51 min 35.95 sec	18 min 56.14 sec
5.000.000	25.000.000	2 h 52 min 33.17 sec	30 min 42.55 sec
8.000.000	40.000.000	3h 34 min 34.44 sec	48 min 2.82 sec

Tabella 4.20: Risultati interrogazione MongoDB 64 bit

Dopo l'esecuzione della query con il map-reduce il database restituisce le seguenti informazioni di esecuzione per l'utente:

```
{
   "result" : "totals",
   "timeMillis" : 123,
   "counts" : {
       "input" : 5000,
       "emit" : 5000,
       "reduce" : 60,
       "output" : 12
   },
   "ok" : 1
}
```

Il campo `result` indica il nome della collezione in cui il risultato del map-reduce viene memorizzato; il campo `timeMillis` indica il tempo di esecuzione del map-reduce; il campo `input` indica il numero di oggetti che sono stati analizzati; il campo `emit` indica il numero di volte che è stata chiamato il metodo `emit`; il campo `reduce` indica il numero di volte che è stata chiamata la funzione `reduce`; il campo `output` indica il numero di documenti presenti nella collezione di output; il campo `ok` indica se l'interrogazione è andata a buon fine altrimenti compare un messaggio di errore.

MySQL

L'interrogazione è stata applicata ad entrambi i PC con sistema operativo a 32 e 64 bit. Nela Tabella 4.21 e nella Tabella 4.22 vengono riportati i risultati dell'interrogazione. La colonna `Righe inserite` indica il numero di righe inserite nelle tabella senza riferimenti, cioè `product`, `users`, `category` da cui poi si sono ricavate in modo casuale il numero di righe per le altre tabelle, in pratica è una colonna volta a differenziare i vari inserimenti; la colonna `Carico` indica il numero totale di righe presenti nel database; la colonna `Popolamento database` indica il tempo impiegato per il popolamento del database; la colonna `Esecuzione query` indica il tempo di esecuzione dell'interrogazione sul database; la colonna `Cancellazione` indica il tempo di cancellazione di tutte le righe presenti nelle tabelle del database.

- Interrogazione PC a 32 bit:

Righe inserite	Carico	Popolamento database	Esecuzione query
100	4.024	1.03 sec	0.01 sec
500	18.415	1.92 sec	0.02 sec
1.000	41.068	3.76 sec	0.05 sec
5.000	189.957	12.91 sec	0.59 sec
10.000	512.782	59.21 sec	0.86 sec
50.000	2.458.691	3 h 33 min 26.14 sec	1 min 4.78 sec
100.000	4.129.292	24 h 30 min 26.17 sec	10 min 41.56 sec

Tabella 4.21: Risultati interrogazione MySQL 32 bit

- Interrogazione PC a 64 bit

Righe inserite	Carico	Popolamento database	Esecuzione query
100	4.482	1.71 sec	0.05 sec
500	24.498	3.08 sec	0.09 sec
1.000	50.252	4.54 sec	0.10 sec
5.000	238.389	19.15 sec	0.29 sec
10.000	415.249	28.43 sec	0.57 sec
50.000	1.925.712	2 min 22.66 sec	2.73 sec
100.000	4.255.702	3 min 21.02 sec	4.06 sec
300.000	14.266.151	3 h 44 min 33.15 sec	27.16 sec
500.000	22.760.144	7 h 18 min 13.99 sec	5 min 18.34 sec

Tabella 4.22: Risultati interrogazione MySQL 64 bit

Confronto tra MongoDB e MySQL

Le stesse interrogazioni implementate in precedenza sono state applicate ai database popolati con i medesimi dati, nella Tabella 4.23 e nella Tabella 4.24 vengono riportati i risultati delle interrogazioni per i database di MongoDB e MySQL sui PC con sistema operativo a 32 e 64 bit.

La colonna `Dati inseriti` delle due tabelle indica ilo numero di documenti inseriti nel database di MongoDB a cui corrispondono i dati inseriti nel databse MySQL tra le varie tabelle sottoforma di righe; la colonna `Query MongoDB` indica i tempi di esecuzione dell'interrogazione sul database MongoDB e la colonna `Query MySQL` indica i tempi di esecuzione dell'interrogazione sul database MySQL.

- Interrogazione PC a 32 bit

Dati inseriti	Query MongoDB	Query MySQL
100	78 ms	0.05 sec
500	187 ms	0.08 sec
1.000	280 ms	0.22 sec
5.000	1.19 sec	0.91 sec
10.000	2.28 sec	1.75 sec
50.000	11.23 sec	35.46 sec
100.000	22.35 sec	1 min 42.01 sec
200.000	45.97 sec	3 min 12.33 sec

Tabella 4.23: Confronto interrogazione 32 bit

- Interrogazione PC a 64 bit

Dati inseriti	Query MongoDB	Query MySQL
100	109 ms	0.06 sec
500	202 ms	0.09 sec
1.000	374 ms	0.12 sec
5.000	1.79 sec	0.23 sec
10.000	3.48 sec	0.41 sec
50.000	17.52 sec	2.01 sec
100.000	34.30 sec	9.73 sec
200.000	58.12 sec	14.76 sec
300.000	1 min 43.12 sec	32.23 sec
500.000	2 min 49.53 sec	2 min 16.33 sec
1.000.000	5 min 51.56 sec	5 min 21.05 sec
2.000.000	13 min 56.06 sec	21 min 10.33 sec
3.000.000	19 min 24.55 sec	1 h 2 min 23.69 sec

Tabella 4.24: Confronto interrogazione 64 bit

4.2.4 Analisi dei risultati dei test delle prestazioni

In questa sezione verranno analizzate le prestazioni di inserimento e di interrogazione ottenute dai database MySQL e MongoDB, cercando di individuare le cause e le motivazioni che hanno portato ad ottenere questi risultati.

Analisi dei test di inserimento

I test di inserimento sul database di MongoDB sono stati effettuati con i seguenti metodi: inserimento con file TXT, inserimenti semplici con driver java e inserimenti di massa con driver java con 1.000, 5.000 e 10.000 documenti per lista.

Dai test è emerso che il metodo più efficiente per inserire documenti in una collezione del database è il metodo di inserimento di massa con 1.000 documenti per lista per quanto riguarda gli inserimenti effettuati sul PC con sistema operativo a 32 bit e il metodo di inserimento di massa con 5.000 documenti per lista per quanto riguarda il PC con sistema operativo a 64 bit.

Nel grafico di Figura 4.1 vengono riportati i risultati di inserimento in MongoDB per il PC con sistema operativo a 32 bit. Sono stati considerati gli inserimenti a partire da 100.000 documenti fino a 4.000.000 di documenti. Dal grafico si può evincere che il miglior metodo di inserimento, anche se di poco, sia quello degli inserimenti di massa con 1000 documenti per lista. Hanno ottime prestazioni anche gli inserimenti di massa con 5.000 e 10.000 documenti per lista con andamenti simili a quelle con 1.000 documenti per lista. I metodi peggiori di inserimento si sono rivelati il metodo di inserimento con driver java seguito dagli inserimenti con file TXT.

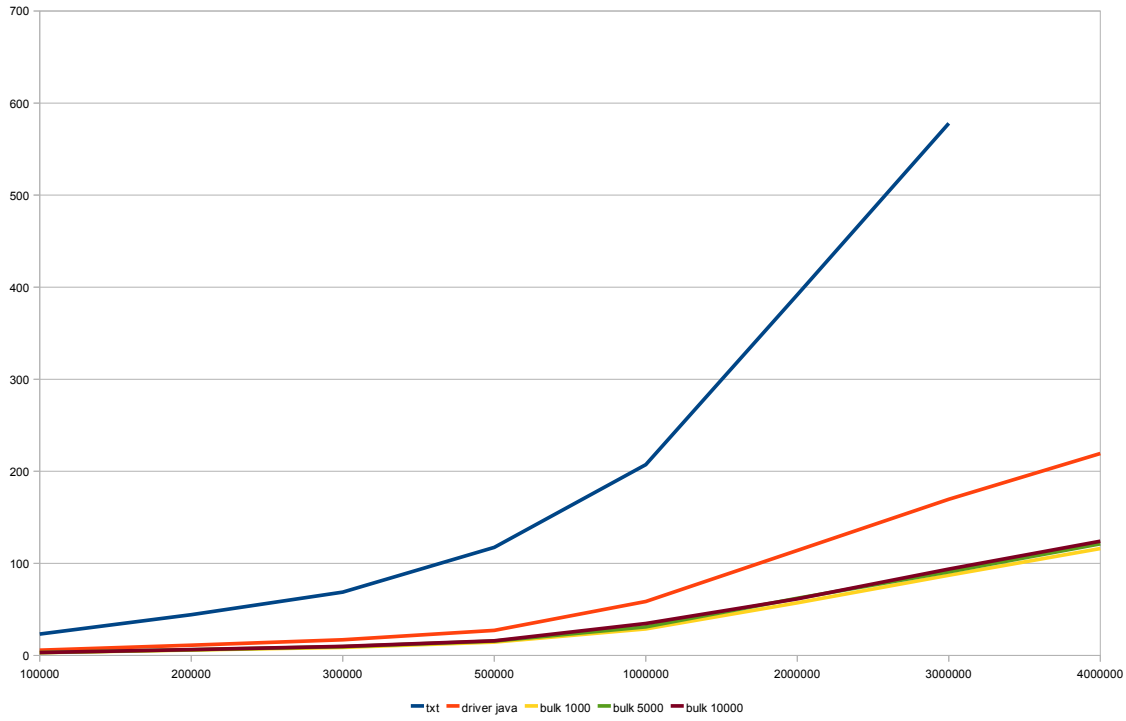

Figura 4.1: Confronto inserimenti MongoDB 32 bit

Nel grafico di Figura 4.2 vengono invece riportati i risultati di inserimento per il PC con sistema operativo a 64 bit, si sono considerati i dati relativi agli inserimenti tra 1.000.000 e 30.000.000 di documenti. Come si può notare dal grafico il metodo peggiore di inserimento rimane il metodo di inserimento con file TXT, gli altri metodi hanno un andamento meno costante rispetto al caso del sistema operativo a 32 bit. Si ha un miglioramento del metodo di inserimento con driver java che si allinea agli andamenti degli inserimenti di massa. Complessivamente il metodo più efficiente di inserimento risulta essere il metodo di inserimento di massa con 5.000 documenti per lista.

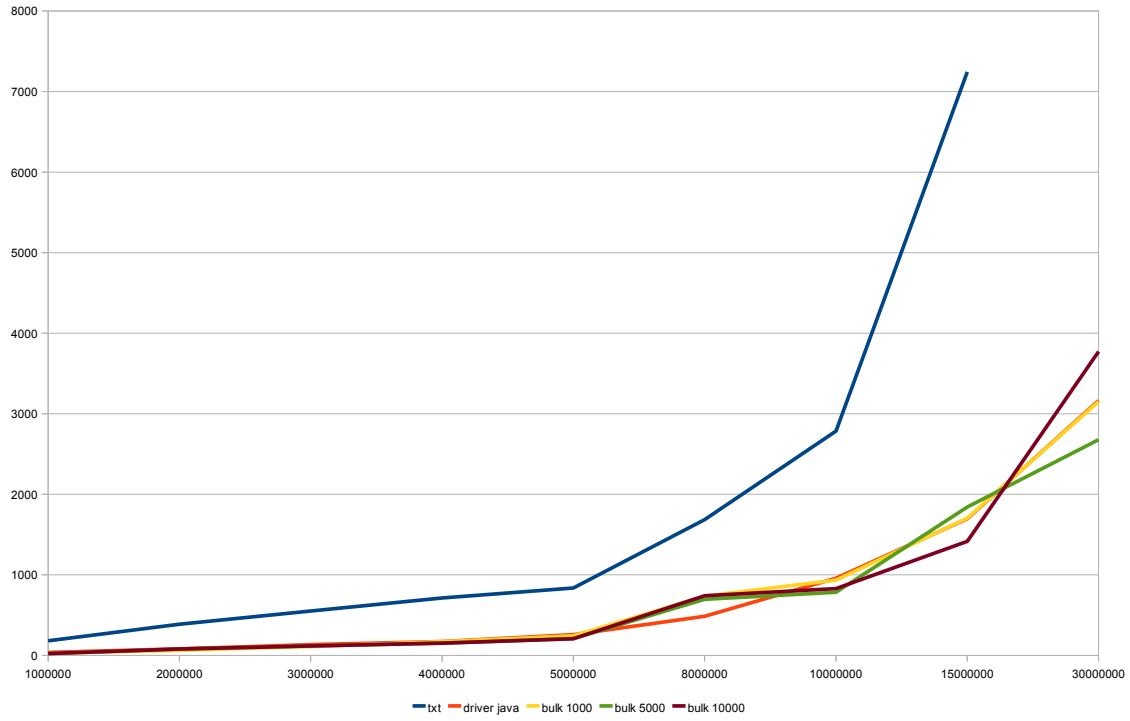

Figura 4.2: Confronto inserimenti MongoDB 64 bit

I test di inserimento nel database MySQL sono stati effettuati con i seguenti metodi: inserimento con file CSV, inserimento con file TXT, inserimento tramite Prepared Statement con driver Java e inserimento tramite Batch Prepared Statement con driver Java.

Dai test è emerso che il metodo più efficiente per inserire dati in una tabella del database è il metodo di inserimento con file CSV per entrambi i PC con sistema operativo a 32 e 64 bit.

Nel grafico di Figura 4.3 vengono riportati i risultati di inserimento per il PC con sistema operativo a 32 bit dove si sono considerati gli inserimenti da 100.000 a 1.000.000 di righe. Come si può notare dal grafico, i quattro metodi di inserimento hanno tutti lo stesso andamento ma il miglior metodo di inserimento risulta essere il metodo di inserimento con file CSV, seguono il metodo di inserimento con i Prepared Statement e con i Batch Prepared Statement, infine il peggior metodo risulta quello con file TXT.

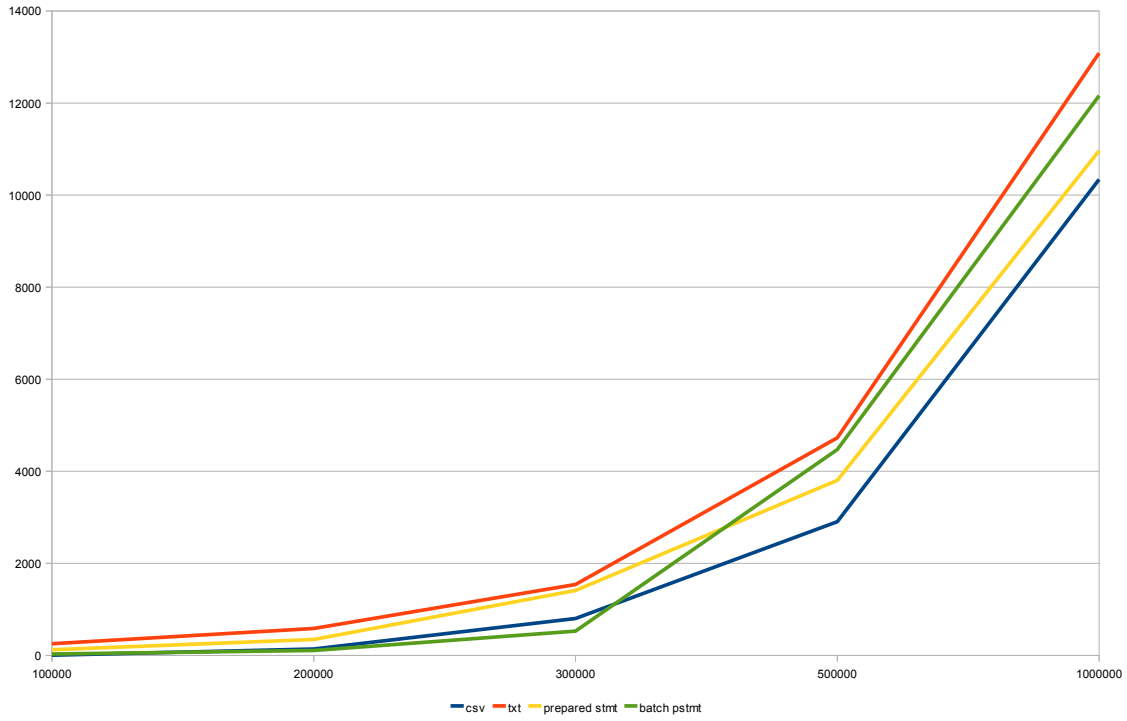

Figura 4.3: Confronto inserimenti MySQL 32 bit

Nel grafico di Figura 4.4 vengono riportati i risultati di inserimento per il PC con sistema operativo a 64 bit dove si sono considerati gli inserimenti da 10.000 a 5.000.000 di righe. Dal grafico si può notare un notevole peggioramento dei metodi di inserimento con file TXT e con i Prepared Statement, probabilmente dovuto dalla lentezza di esecuzione del PC con sistema operativo a 64 bit. Il miglior metodo di inserimento risulta essere ancora il metodo di inserimento con file CSV seguito dal metodo di inserimento con i Batch Prepared Statement.

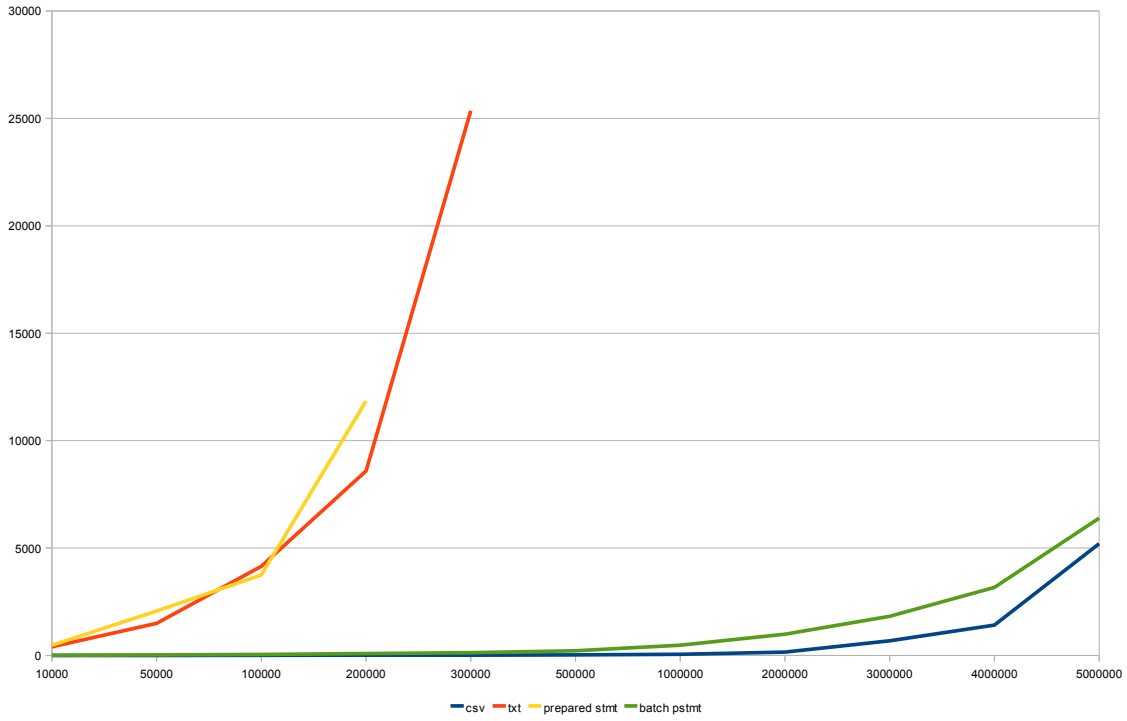

Figura 4.4: Confronto inserimenti MySQL 64 bit

Infine si sono messi a confronto i metodi più efficienti nei seguenti grafici. Il grafico di Figura 4.5 mette a confronto il metodo di inserimento con file CSV applicato al database MySQL con il metodo di inserimento di massa con 1.000 documenti per lista applicato al database MongoDB sul PC con sistema operativo a 32 bit.

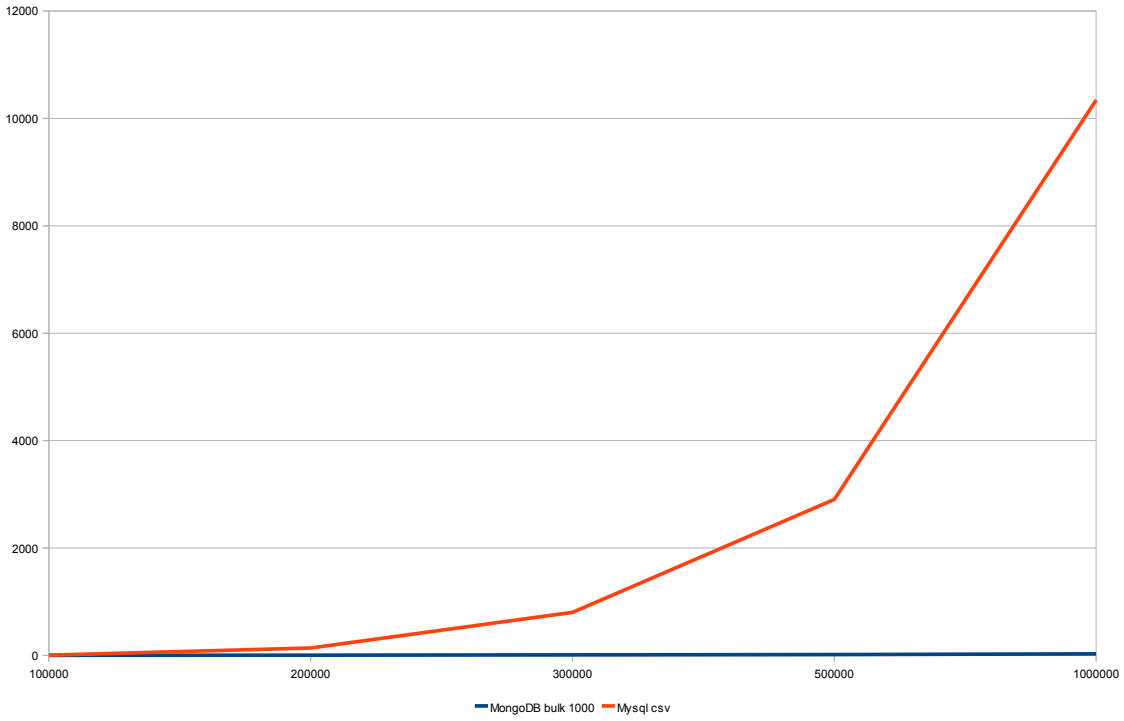

Figura 4.5: Confronto inserimenti MySQL MongoDB 32 bit

Il grafico di Figura 4.6 mette a confronto il metodo di inserimento con file CSV applicato al database MySQL con il metodo di inserimento di massa con 5.000 documenti per lista applicato al database MongoDB sul PC con sistema operativo a 64 bit.

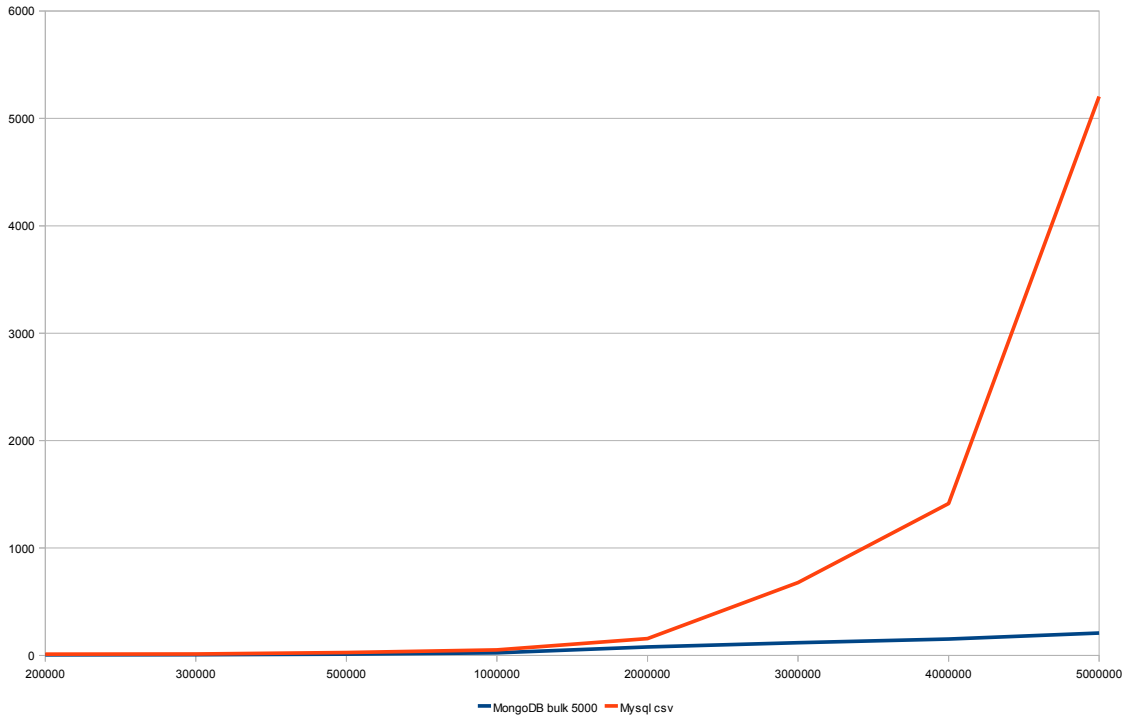

Figura 4.6: Confronto inserimenti MySQL e MongoDB 64 bit

Come si può evincere dai grafici, per il caso di studio MongoDB risulta essere molto più veloce in fase di inserimento rispetto a MySQL. Questa forte disparità in inserimento dei due database può essere motivata da vari fattori di implementazione di MongoDB e MySQL che sono elencati di seguito:

- Di default in MongoDB tutte le operazioni di scrittura sono *fire-and-forget*, il che vuol dire che tutte le scritture sono mandate al database attraverso un socket TCP senza che sia richiesta una risposta dal database. Se l'utente vuole una risposta dal database, è possibile emettere una scrittura usando il *safe mode* fornito da tutti i driver in modo da forzare una risposta dal database. Inoltre di default in MongoDB viene abilitato il journaling che impone che ogni scrittura venga committata. Se il server viene spento in modo non corretto il journaling permette di ripristinare i dati a uno stato consistente quando si riavvia il server.

- MongoDB usa i file mappati in memoria per accedere ai dati memorizzati. Quando viene attivato il server, il memory mapped files usa la memoria virtuale del sistema operativo per mappare i dati richiesti in memoria, è poi responsabilità del sistema operativo gestire lo scaricamento dei dati su disco e la loro paginazione. Questo metodo porta a certi vantaggi. Il codice di gestione della memoria è semplice e breve, perché la maggior parte del lavoro viene svolto dal sistema operativo; la dimensione virtuale del processo server di MongoDB è spesso superiore all'intero insieme di dati; fornendo un'appropriata quantità di memoria, MongoDB può tenere molti più dati in memoria riducendo il bisogno di operazioni di input e output su disco. L'unica limitazione è che nei sistemi operativi a 32 bit il server di MongoDB è limitato a 2 GB di dati.

- InnoDB è un motore per il salvataggio di dati per MySQL, la sua caratteristica principale è quella di supportare le transazioni di tipo ACID. InnoDB fornisce un compromesso tra velocità e durevolezza in quanto usa una nuova tecnica di scaricamento file chiamata *doublewrite* che fornisce sicurezza nel recupero dei dati a seguito di un crash del sistema operativo. *Doublewrite* significa che prima di scrivere una pagina in un file dati, InnoDB per prima cosa le scrive i dati in un doublewrite buffer. Solo dopo che la scrittura e lo scaricamento al double write buffer è completato, InnoDB scrive la pagina nella posizione appropriata nel file di dati. Questa doppia scrittura potrebbe essere un motivo della maggiore lentezza in inserimento.

- L'imposizione di vincoli *not null*, *unique*, *primary key* e di *foreign key* che garantiscono la consistenza dei dati costringe il database a controllare la validità dei valori inseriti o modificati, causando di conseguenza un ritardo nelle risposte del database.

Analisi dei test di interrogazione

I test di interrogazione sono stati effettuati in due modi differenti. Il primo è stato quello di inserire un certo numero di dati nei database MySQL e MongoDB casualmente per testare principalmente come i due database si comportassero in fase di interrogazione all'aumentare del carico di dati da gestire.

Dai risultati ottenuti si può notare che per il PC con sistema operativo a 32 bit il database MySQL ha buone prestazioni in interrogazione fino a un carico di circa di 2.000.000 di righe divise in tutte le tabelle, dopo questa soglia i tempi di esecuzione dell'interrogazione aumentano esponenzialmente con un calo vistoso delle prestazioni. Lo stesso comportamento si ottiene con il PC con sistema operativo a 64 bit, dove il database ha ottime prestazioni in interrogazione gestendo un carico fino a 10.000.000 di righe, aumentando la quantità dei dati i tempi di esecuzione dell'interrogazione aumentano esponenzialmente.

Comportamento diverso ha invece il database MongoDB che per entrambi i PC con sistema operativo a 32 bit e 64 bit all'aumentare dei documenti inseriti nelle collezioni i tempi di esecuzione aumentano in modo più lineare rispetto ai tempi di esecuzione riscontrati con il database MySQL.

Il secondo esperimento, più significativo per il confronto dei due database in fase di interrogazione, consiste nel popolare i due database con gli stessi dati, ovvero per ogni documento inserito in MongoDB sono stati inseriti gli stessi dati nel database MySQL nelle varie tabelle. Questo approccio consente di interrogare i due database con lo stesso carico di dati, in modo da poter ottenere un valido metodo di confronto per poter valutare quale tecnica di interrogazione risulti più efficiente, il map-reduce per MongoDB o il linguaggio SQL e il comando GROUP BY per MySQL.

Nel grafico di Figura 4.7 vengono riportati i confronti tra i tempi di esecuzione dei due database sul PC con sistema operativo a 32 bit a parità di dati inseriti.

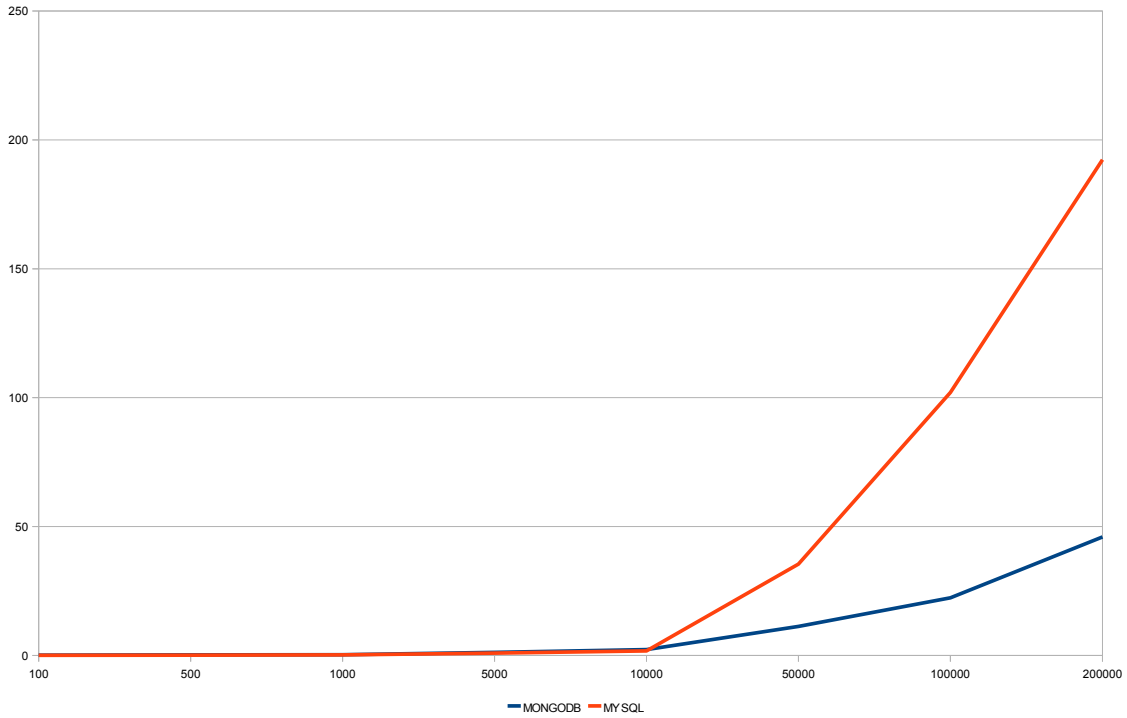

Figura 4.7: Confronto interrogazione MySQL e MongoDB 32 bit

Nel grafico di Figura 4.8 vengono riportati i confronti tra i tempi di esecuzione dei due database sul PC con sistema operativo a 64 bit a parità di dati inseriti.

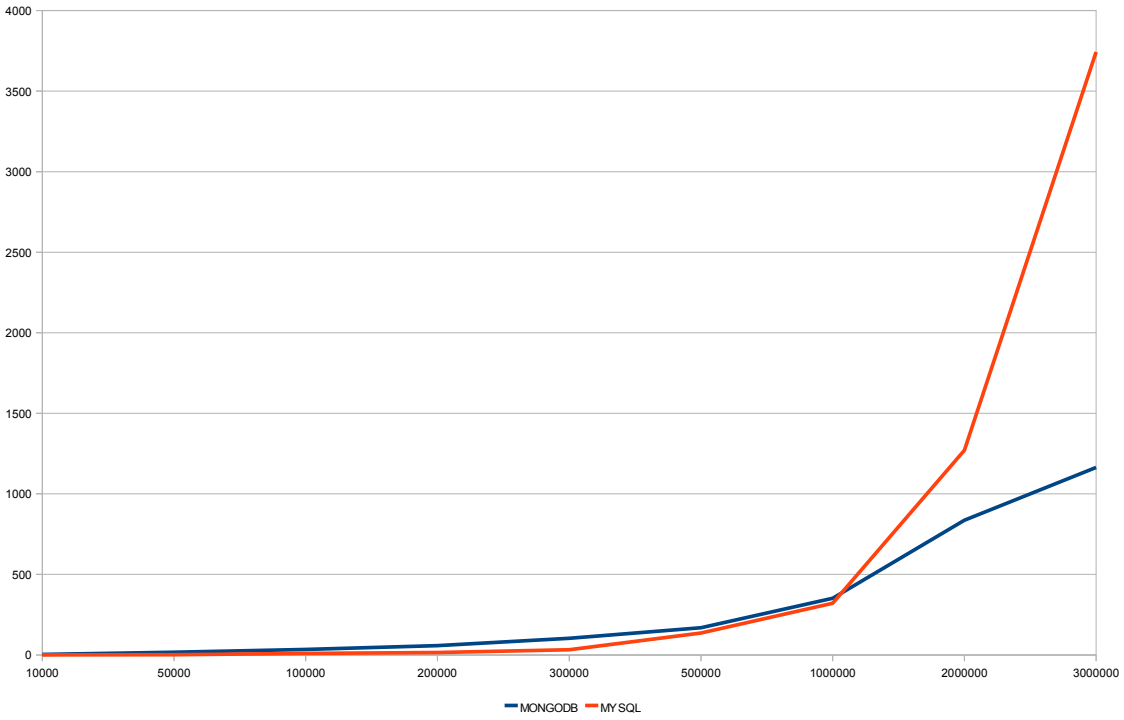

Figura 4.8: Confronto interrogazione MySQL e MongoDB 64 bit

Come si può notare dai grafici l'esecuzione del map-reduce di MongoDB risulta essere più efficiente per elevate quantità di dati inseriti, invece MySQL risulta essere quattro volte più lento nel PC con sistema operativo a 32 bit e tre volte più lento nel PC con sistema operativo a 64 bit. Invece per quantità di dati non elevate MySQL in fase di interrogazione risulta essere più veloce di MongoDB.

Questa lentezza in esecuzione per piccole quantità di dati può essere spiegata dal modo in cui il map-reduce di MongoDB è implementato e da come viene eseguito. Bisogna considerare che il map-reduce esegue due funzioni, la funzione map e la funzione reduce, scritte in JavaScript che di conseguenza eseguite dall'interprete JavaScript e non dal database stesso. Inoltre i dati in MongoDB sono memorizzati in formato BSON e invece Javascript gestisce oggetti in formato JSON. Tenendo

presente questi due aspetti, il map-reduce durante la sua esecuzione esegue i seguenti passi: conversione da BSON a JSON, esecuzione funzione map, conversione da JSON a BSON, conversione da BSON a JSON, esecuzione funzione reduce e conversione da JSON a BSON.

Pertanto è lecito pensare che per piccole quantità di dati il tempo impiegato per le varie conversioni dei dati risulti più elevato dell'esecuzione delle funzioni rispetto alla quantità di dati gestita.

MySQL risulta essere invece più lento per elevate quantità di dati in quanto le tre operazioni di join dell'interrogazione costano molto considerando il tempo complessivo dell'interrogazione e sembra costare di più all'aumentare del carico di dati su cui viene chiamata l'interrogazione.

Capitolo 5

Denormalizzazione

La maggior parte dei moderni linguaggi di programmazione supporta la programmazione orientata agli oggetti, grazie alla quale si modellano le entità nel nostro codice principalmente con oggetti o con l'utilizzo di strutture di oggetti innestati.

Purtroppo le strutture dati che si usano per conservare i dati non si sono evolute di pari passo con i linguaggi di programmazione. Infatti per circa 30 anni il modello relazionale è stato l'unica scelta disponibile per l'archiviazione e la gestione dei dati. Esso si basa su due concetti fondamentali, la relazione e la tabella. La presenza contemporanea di questi due concetti l'uno formale e l'altro intuitivo, è responsabile del grande successo ottenuto dal modello relazionale.

Ma la profonda differenza tra il modo in cui si modella la realtà nel codice, tramite gli oggetti, e il modo in cui vengono rappresentati i dati in memoria, tramite le tabelle, è stata sempre fonte di difficoltà per i programmatori. Infatti ogni linguaggio di programmazione è fornito di strumenti chiamati ORM, sistemi di Object-Relational Mapping, che favoriscono l'integrazione dei sistemi software aderenti al paradigma della programmazione ad oggetti con i sistemi RDBMS.

L'innovazione principale portata da MongoDB, grazie al quale è diventato popolare in breve tempo, è stato il nuovo modello di dati basato su documenti di tipo JSON, che grazie ad esso permette di modellare le entità più facilmente e in modo più espressivo a livello di applicazione. Pertanto disponendo dello stesso modello di dati nel nostro codice e nel database è possibile semplificare i compiti di sviluppo applicativo ed eliminare lo strato complesso di codice per la mappatura, che altrimenti sarebbe necessario.

5.1 MongoDB come database denormalizzato

In MongoDB i dati sono memorizzati come sono rappresentati nel programma applicativo e questa caratteristica consente a MongoDB di essere adatto a poter modellizzare le entità che sostengono la maggior parte delle moderne applicazioni web.

Negli ultimi anni le nuove applicazioni web hanno dovuto affrontare nuove sfide e problematiche dovute al cambiamento di diversi fattori connessi ad aspetti di memorizzazione e gestione dei dati che fin dall'avvento dei database relazionali non si erano presentati.

Alcune di questi sono la memorizzazione di molti più dati che ha portato ad un aumento repentino delle dimensioni dei database e un conseguente maggior sforzo di gestione dei dati, motivazioni che hanno spinto i database ad essere distribuiti e scalati su più macchine. L'evoluzione dei sistemi di memorizzazione dei siti web, da semplici file statici a complessi database. La necessità di raggiungere tempi di risposta sotto al secondo nei confronti di interrogazioni effettuate sul database da molti utenti concorrenti. Lo sviluppo di applicazioni nate con lo scopo di memorizzare dati tanto velocemente quanto vengono processate le query. La presenza e gestione di dati mutevoli nel tempo e non strutturati.

Tutte queste nuove necessità hanno portato a rivalutare i requisiti di un database e ad effettuare compromessi quando si sviluppa un software, elaborando nuove risposte che si allontanano concettualmente e strutturalmente dai tradizionali database relazionali. È il caso di MongoDB e del suo innovativo modello dei dati basato sui documenti.

Il modello dei dati di MonogDB non si differenzia semplicemente per la struttura dati utilizzata per la memorizzazione dei dati, ma anche per il modo diverso con cui vengono memorizzati i dati al suo interno. Nei documenti possono essere memorizzati dati sotto forma di coppie chiave/valore fino a 16 MB. Alle chiavi è possibile associare semplici stringhe o valori numerici, array, ma anche documenti innestati e array di documenti. Infatti una delle caratteristiche più importanti del modello di dati di MongoDB è l'assenza di uno schema fisso che consente di modellare i dati in forma aggregata grazie alla possibilità di incorporare documenti permettendo così di lavorare con un oggetto in modo olistico.

Tutte queste nuove proprietà sono in netto contrasto con la teoria del modello relazionale e in particolare rispetto alla normalizzazione dello schema del database. La normalizzazione è un passo fondamentale per la realizzazione dello schema di un database relazionale, in quanto questa procedura determina la qualità dello schema. Uno schema normalizzato è privo di ridondanze in quanto questo processo si fonda su un semplice criterio: se una relazione presenta più concetti tra loro dipendenti, la si decompone in relazioni più piccole, una per ogni concetto. Inoltre uno schema ben normalizzato garantisce che operazioni di inserimento, aggiornamento e cancel-

lazione mantengano la consistenza e non generino anomalie nei dati per garantire una maggiore semplicità della loro gestione.

A differenza dei database relazionali MongoDB non impone alcuna sorta di schema ai documenti e alle collezioni, quindi ha la possibilità di memorizzare qualsiasi tipo di informazione all'interno di più documenti della stessa collezione o anche in diverse collezioni. MongoDB manca totalmente di operazioni di join e di chiavi esterne, questo fatto consente di scrivere query più semplici e rende la loro esecuzione più veloce. Infine MongoDB è facile da scalare, ovvero di distribuire i dati su più macchine, cosa che non sarebbe così semplice se si ha un database normalizzato che impone uno schema fisso fortemente riferito.

Perciò parlando in termini di database relazionali MongoDB è un DBMS non-normalizzato o denormalizzato. Anche se gli aggettivi denormalizzato o non-normalizzato non si possono associare a MongoDB senza precisarne il significato. Un database denormalizzato è un database normalizzato dove sono state introdotte deliberatamente delle ridondanze per ottenere un particolare guadagno; un database non-normalizzato è un database disorganizzato dove nessuno si è preoccupato di dove i dati sono memorizzati.

Ovviamente MongoDB non è un database denormalizzato perché non nasce con lo scopo di essere normalizzato per poi introdurre ridondanze e non è nemmeno un database non-normalizzato, perché pur avendo dati ridondanti e situati in posti diversi, ha una sua logica di memorizzazione infatti in MongoDB è possibile sia denormalizzare che normalizzare i dati . Ma definire MongoDB un database denormalizzato o non-normalizzato puntualizza semplicemente il fatto che al suo interno i dati vengono memorizzati in modo diverso rispetto ai database relazionali. L'aspetto più importante da tenere in considerazione è che si hanno molti dati duplicati e memorizzati in luoghi diversi nel database.

5.1.1 Normalizzare e denormalizzare: pro e contro

Normalizzare o denormalizzare un database ha dei pro e dei contro. I database normalizzati funzionano bene nelle condizioni in cui le applicazioni eseguono operazioni di scrittura ad alta intensità e il carico delle operazioni in scrittura è maggiore del carico in lettura. Questo è motivato dal fotto che nei database relazionali le tabelle sono solitamente di piccole dimensioni e i dati sono divisi verticalmente tra molte tabelle. Questo permette di avere migliori prestazioni se sono abbastanza piccole da essere inserite nel buffer. Gli aggiornamenti sono molto veloci perché i dati da aggiornare sono memorizzati in una singola locazione e non ci sono duplicati e per lo stesso motivo i dati possono essere facilmente inseriti. Inoltre l'assenza di dati ridondanti o duplicati in diverse locazioni permette che non avvengano anomalie di aggiornamento, inserimento e cancellazione.

Ma i database normalizzati presentano anche alcuni svantaggi. Non vengono memorizzati valori derivati dal calcolo di altri valori del database, se non si memorizzano questi valori pre-calcolati deve essere l'applicazione a doverli calcolare al volo quando necessario. Se l'applicazione cambia velocemente nel tempo c'è il rischio di non essere in grado di riprodurre dei risultati riguardanti lo stato iniziale dei dati prima del cambiamento. Dato che ogni fatto è memorizzato esattamente in un singolo luogo può risultare difficoltoso raggruppare insieme tutte le informazioni in un'unica query ciò richiede l'utilizzo di molti join su più tabelle. L'utilizzo di molti join porta ad affrontare problemi di prestazioni in quanto un'operazione di join risulta molto più costosa di un'operazione di lettura su una singola tabella.

Invece i database denormalizzati funzionano molto bene quando il carico di operazioni in lettura è maggiore del carico in scrittura in quanto i dati sono presenti nello stesso luogo e non c'è bisogno di operazioni di join perciò la selezione è molto veloce; inoltre c'è un utilizzo più efficace degli indici.

Lo svantaggio più importante dei database denormalizzati è la presenza di informazioni duplicate in più locazioni del database che rende gli aggiornamenti e gli inserimenti dei dati più complesso e costoso in quanto bisogna affrontare problemi di inconsistenza dei dati. Sistemare le inconsistenze dei dati diventa compito dell'applicazione. Se consideriamo la situazione in cui all'interno di un sistema ogni utente ha una lista di username dei suoi amici e alcuni degli utenti del sistema cambiano il proprio username, in un database normalizzato questa è una semplice operazione di UPDATE sul database. Invece in un database denormalizzato deve esistere un meccanismo per aggiornare gli username in tutti i luoghi in cui compare quell'username da modificare. Molti servizi che creano e utilizzano database hanno dei programmi che costantemente operano sul database per gestire queste inconsistenze. Quindi quando si denormalizzano i dati bisogna tenere presente che l'applicazione dovrà essere più complessa.

Come si è visto i database normalizzati e non-normalizzati presentano certi vantaggi e svantaggi, pertanto la scelta di adottare una delle due soluzioni parte da motivazioni differenti. Con i database relazionali si partiva dal dominio degli oggetti e li si rappresentava in modo che virtualmente qualsiasi query poteva essere espressa. Quando c'è la necessità di ottimizzare le performance, si guarda alle query che si stanno eseguendo in quel momento e si fondono le tabelle per creare righe più lunghe e diminuire l'uso di join. Con i database non relazionali si parte direttamente dall'analisi delle query che si devo eseguire sul database per definire il modello dei dati.

MongoDB fornisce la possibilità sia di denormalizzare i dati, tramite l'incapsulamento di documenti annidati, sia di normalizzare i dati con l'utilizzo dei link manuali o l'utilizzo del DBRef() , la scelta tra le due strategie o della loro combinazione dipende dalla natura dell'applicazione che si sta implementando e di quello che si vuole offrire con essa. Ci sono dei fattori di decisione che possono far propendere verso

una soluzione piuttosto che un'altra e sono i seguenti.

Bisogna considerare il rapporto che c'è tra operazioni di lettura e scrittura, vedere se si è disposti a pagare un prezzo elevato ogni 10000 letture per far si che una rara occorrenza che viene modificata più velocemente e in modo consistente, dipende dai propositi dell'applicazione. Bisogna inoltre se le letture o le scrittura debbano essere più veloci, nel caso delle letture è meglio denormalizzare. Bisogna considerare quanto spesso i dati che si riferiscono tra loro cambiano nel tempo, meno cambiano e più conviene denormalizzare i dati. Valutare quanto sia importante la consistenza dai dati all'interno del sistema, perché è possibile avere applicazioni che tollerino brevi periodi di inconsistenza dei dati o che richiedano dei vincoli meno stringenti. Infine bisogna anche valutare anche per quali ambiti di utilizzo il database scelto è adatto; MongoDB è ottimo per elaborare e memorizzare dati real time di analisi e di logging.

5.1.2 Denormalizzazione in ambito distribuito

Allargando il discorso sulla denormalizzazione in ambito distribuito, le moderne applicazioni stanno diventando distribuite a causa dell'aumento dell'insieme di dati da gestire, l'aumento degli utenti concorrenti che accedono ai dati e richiedono operazioni su di esse domandano elevate prestazioni in risposta. Le nuove tecnologie database necessitano di essere veloci non solo su un singolo server ma anche su server multipli dato che un sistema distribuito inevitabilmente deve essere scalabile orizzontalmente e altamente disponibile.

Tutti questi obiettivi e problematiche hanno portato in ambito distribuito a sacrificare alcune proprietà care ai database relazionali rivolgendosi a tecnologie che fornissero alternative a seconda degli obiettivi posti. Un esempio sono le proprietà ACID per le transazioni. Con l'acronimo ACID si indicano quattro proprietà delle transazioni Atomicità, Consistenza, Isolamento e Durata; tali proprietà garantiscono che una transazione si un'unità di lavoro indivisibile che pur eseguita in concorrenza con altre transazioni per l'accesso di risorse concorrenti garantisca di rilasciare le risorse in uno stato consistente e che i suoi aggiornamenti siano persistenti.

Ma se si considera un database distribuito diventa molto difficile raggiungere questo tipo di consistenza e isolamento che può essere ottenuto facilmente su una singola macchina. Ma il problema non è solo la difficoltà ma piuttosto arrivare a tali obiettavi finisce per andare spesso in conflitto con le ragioni per cui si è optato per un sistema distribuito.

Il problema più importante in un sistema distribuito è la consistenza dei dati. I database relazionali forniscono una robusta consistenza dei dati, sono stati adattati per fornire un'elevata disponibilità, ma sono difficili e costosi da partizionare. I nuovi database non relazionali hanno effettuato un compromesso tra consistenza e disponibilità dato che la tolleranza al partizionamento è fondamentale per un sistema

distribuito. Nella maggior parte dei casi è stata sacrificata la consistenza dei dati, ma nel senso che si è operata una distinzione tra consistenza locale e assoluta consistenza globale del sistema.

I requisiti di consistenza di certe applicazioni non sono molto stringenti, quindi se si accetta una consistenza non perfetta, si possono raggiungere elevati miglioramenti nelle prestazioni. Infatti con applicazioni in cui i database utilizzati sono suddivisi su centinaia di nodi, la penalità pagata nelle performance per effettuare il lock di un database per una modifica è molto elevato. Se l'applicazione ha frequenti operazioni di scrittura, si stanno serializzando tutte le scritture perdendo i vantaggi del database distribuito. In pratica in un database che fornisce una consistenza eventuale dei dati i cambiamenti si propagano su tutti i nodi prima che il database arrivi a uno stato consistente.

Se la consistenza non è garantita dal database, diventa un problema che deve affrontare l'applicazione. Quando si sceglie la disponibilità rispetto alla consistenza attuando una denormalizzazione dei dati, la penalità da pagare è la maggiore complessità dell'applicazione.

5.2 Denormalizzazione del database MySQL

Come ultimo esperimento si è voluto proprio affrontare l'aspetto della denormalizzazione. In MongoDB il map-reduce è stato applicato su un'unica collezione dove all'interno erano memorizzati i documenti contenenti tutte le informazioni necessarie per ottenere una risposta alla query. In particolare nell'interrogazione analizzata nel capitolo precedente il map-reduce veniva applicato unicamente alla collezione `orders` senza dover effettuare interrogazioni aggiuntive ad altre collezioni per reperire le informazioni necessarie per il calcolo dei valori richiesti. In questo modo la query ha prestazioni migliori e non c'è bisogno di effettuare operazioni di join, questi sono i vantaggi di avere un database denormalizzato.

A differenza della query di MongoDB, la query effettuata sul database MySQL doveva essere chiamata su più tabelle utilizzando di conseguenza un certo numero di join. Nello specifico l'interrogazione doveva reperire dati da quattro tabelle: `orders`, `insertion`, `price` e `product`. Per questo motivo venivano chiamate tre operazioni di join per fondere le varie informazioni delle tabelle.

Un'operazione di join risulta molto più costosa di un'operazione di lettura associata a una selezione. Questo è lo svantaggio di avere un database normalizzato e fortemente riferito perché impone la scrittura di query più complesse e un abbassamento delle performance direttamente proporzionale al numero di join utilizzato.

Pertanto l'obiettivo dell'ultimo esperimento è quello di denormalizzare il database implementato con MySQL per vedere se inserendo tutti i dati all'interno di un'unica tabella il reperimento delle informazioni risulta più veloce.

Prendendo come esempio un documento della collezione `orders` del database di MongoDB, che contiene al suo interno l'id utente, l'username e le informazioni per il recapito della merce di un utente; i prodotti ordinati dall'utente con relativo id, sku, name e prezzo, oltre agli attributi propri dell'ordine cioè la quantità di merce per ogni prodotto, un id e la data di creazione dell'ordine, si è inserito nella tabella `orders` tutte queste informazioni in modo da non dover utilizzare dei join nell'interrogazione SQL.

Il risultato di questa denormalizzazione perciò ha lo scopo di aumentare le prestazioni in lettura sul database MySQL cercando di metterlo alla pari con il database di MongoDB anch'esso denormalizzato. Lo svantaggio principale della denormalizzazione è la presenza di dati duplicati in più tabelle, ma questo è il prezzo da pagare per poter ottenere maggiori prestazioni di interrogazione.

5.2.1 Progettazione del database denormalizzato

Lo schema ER è stato modificato per inserire nella tabella `orders` gli attributi `ID`, chiave primaria di tipo intero auto incrementante per differenziare le tuple, `Order_ID`, `Order_state`, `Purchase_date`, sono stati aggiunti gli attributi della tabella `users` `User_ID` e `Username`; gli attributi della tabella `product` `Product_ID`, `Sku`, `Name` assieme al prezzo del prodotto `Sale`; il recapito degli utenti della tabella `address` con gli attributi `State`, `Zip`, `City` e `Street` e l'attributo `Quantity` che indica il numero di prodotti inseriti per ogni ordine.

In Figura 5.1 viene presentato lo schema ER modificato comprendente le entità e le associazioni direttamente collegate alla tabella `orders`.

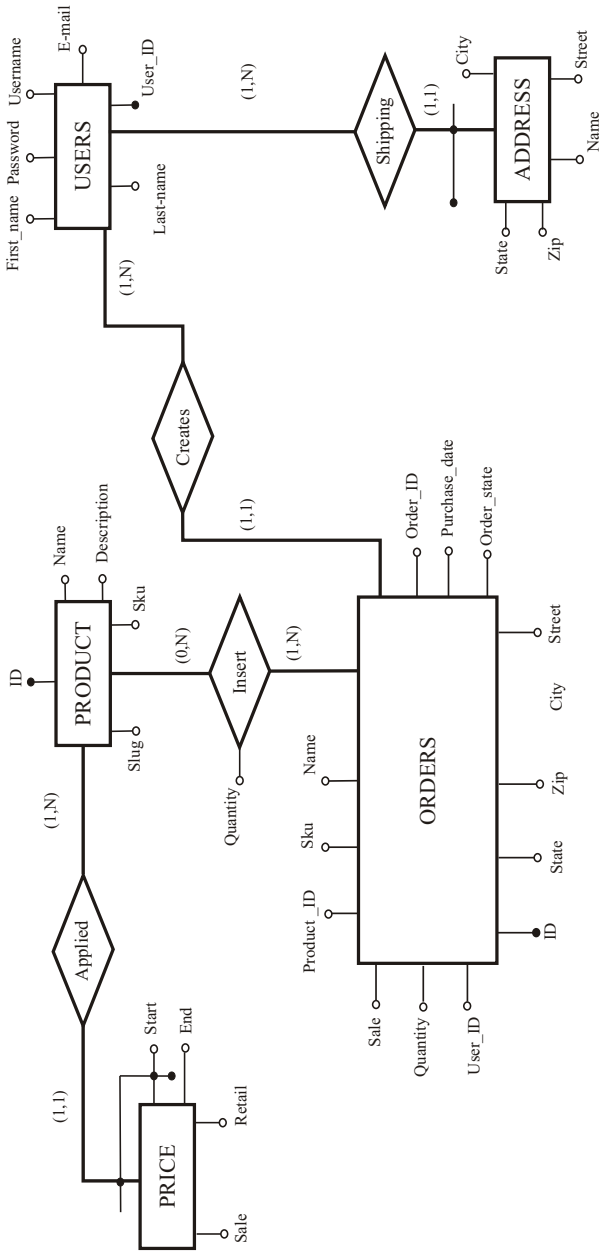

Figura 5.1: Schema ER denormalizzato

A seguito delle modifiche sullo schema ER è stato modificato anche lo schema logico relazionale ottenendo lo schema logico relazionale di Figura 5.2.

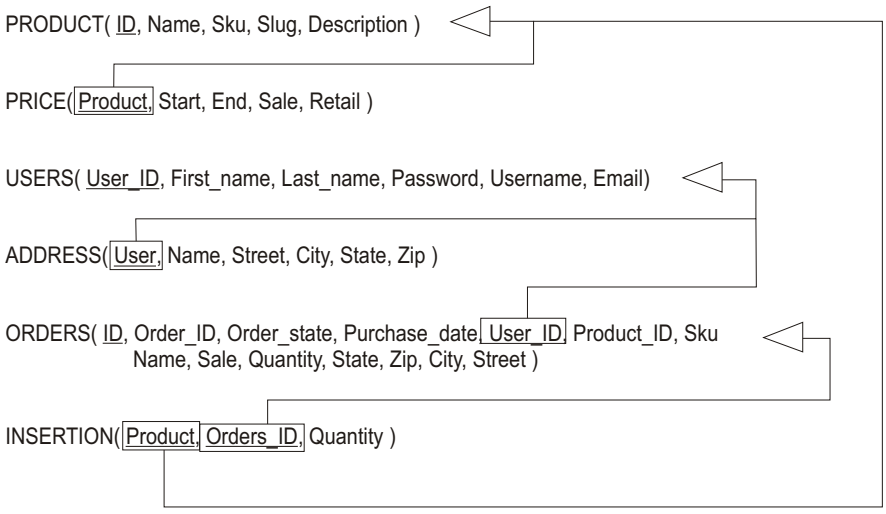

Figura 5.2: Schema logico relazione denormalizzato

Per avvicinirasi ulteriormente al modello dei dati di MongoDB nell'implementazione dello schema SQL sono stati eleminati i vincoli di chiave esterna delle tabelle address, price, orders e insertion. Questa azione è giustificata dal fatto che in MongoDB non esistono vincoli che collegano i vari documenti tra le collezioni e inoltre è possibile aumentare le prestazioni in inserimento dei dati, dato che viene evitato il controllo delle chiavi esterne da parte del database.

Il codice SQL dello schema denormalizzato è riportato nella Sezione A.2 dell'Appendice A.

5.2.2 Inserimenti denormalizzati

Sono stati effettuati dei test di inserimento di dati all'interno delle tabelle del database MySQL denormalizzato, confrontati con i tempi di inserimento in MongoDB. Gli inserimenti in entrambi i database sono stati effettuati con gli stessi dati, ovvero ai dati contenuti in un documento di MongoDB corrispondono gli stessi dati inseriti nelle varie tabelle del database MySQL.

Nella Tabella 5.1 e nella Tabella 5.2 vengono riportati i tempi di esecuzione degli inserimenti effettuati effettuati sui PC con sistema operativo a 32 e 64 bit.

- Inserimenti PC 32 bit:

Dati inseriti	Inserimento MySQL	Inserimento MongoDB
100	0.14 sec	184 ms
500	0.54 sec	509 ms
1.000	0.78 sec	614 ms
5.000	3.36 sec	2.85 sec
10.000	9.16 sec	6.51 sec
50.000	3 min 42.58 sec	27.21 sec
100.000	20 min 7.45 sec	1 min 39.40 sec
200.000	59 min 38.21 sec	2 min 09.09 sec

Tabella 5.1: Confronto inserimenti con MySQL denormalizzato 32 bit

- Inserimenti PC 64 bit:

Dati inseriti	Inserimento MySQL	Inserimento MongoDB
100	0.46 sec	548 ms
500	0.62 sec	689 ms
1.000	1.09 sec	1.17 sec
5.000	3.56 sec	2.86 sec
10.000	5.14 sec	5.02 sec
50.000	51.69 sec	37.71 sec
100.000	1 min 29.91 sec	46.08 sec
200.000	2 min 51.57 sec	2 min 19.88 sec
300.000	7 min 8.31 sec	4 min 17.08 sec
500.000	11 min 26.34 sec	6 min 28.47 sec
1.000.000	32 min 48.66 sec	11 min 33.15 sec
2.000.000	44 min 15.81 sec	27 min 16.29 sec
3.000.000	1 h 14 min 29.56 sec	43 min 24.58 sec

Tabella 5.2: Confronto inserimenti con MySQL denormalizzato 64 bit

5.2.3 Interrogazione denormalizzata

Una volta implementato il nuovo database denormalizzato, è stato popolato con il metodo dell'inserimento con file CSV ed è stata applicata la seguente interrogazione:

```
SELECT      Year( Purchase_date ) AS Year, Month( Purchase_date ) AS Month,
            SUM(Quantity) AS TotQuantity, SUM( Sale*Quantity) AS TotSale
FROM        orders
GROUP BY    Year( Purchase_date ), Month( Purchase_date );
```

Come si può notare ora l'interrogazione risulta molto più semplice rispetto alla precedente in quanto non si necessità più dell'utilizzo di join e ora l'interrogazione è diventata una semplice selezione sulla tabella `orders`.

Oltre alla denormalizzazione si è voluto anche usare un indice. La creazione di indici in un database MySQL permette di evitare che ogni ricerca sia preceduta da una scansione completa delle tabelle utilizzate. L'indicizzazione è stata ideata appositamente per velocizzare l'esecuzione delle query di selezione per l'accesso ai dati in fase di lettura e viene introdotta semplicemente utilizzando l'apposito comando, CREATE INDEX, seguito dal nome del campo o dei campi interessati alla generazione degli indici. Nella progettazione degli indici bisogna tenere conto che, come costo per la velocità che offrono in fase di lettura, comportano un rallentamento in fase di inserimento e aggiornamento dei dati. MySQL consente di creare fino a 16 indici all'interno di una stessa tabella, sono inoltre supportati indici su più colonne, indici multipli relativi a più colonne e indici per ricerche full-text.

Per velocizzare la selezione sulla tabella `orders` è stato inserito un indice nella colonna `Purchase_date` con il seguente comando:

```
CREATE INDEX idx_data ON orders( Purchase_date );
```

Il database MySQL denormalizzato e il database MongoDB sono stati popolati con gli stessi dati, ovvero chiamando le due interrogazioni sui database esse restituiscono lo stesso risultato.

Nella Tabella 5.3 e nella Tabella 5.4 vengono presentati i risultati dell'interrogazione sul database per i computer con sistema operativo a 32 e 64 bit. Dato che i due database sono stati popolati con gli stessi dati, si indica con la colonna `Documenti inseriti` il numero di documenti inseriti in MongoDB a cui corrispondono i dati inseriti in MySQL; nella colonna `Query MySQL` vengono riportati i tempi di esecuzione dell'interrogazione sul database MySQL e nella colonna `Query MongoDB` vengono riportati i tempi di interrogazione sul database MongoDB.

- Interrogazione PC a 32 bit:

Documenti inseriti	Query MySQL	Query MongoDB
100	0.01 sec	67 ms
500	0.02 sec	159 ms
1.000	0.04 sec	333 ms
5.000	0.10 sec	1.19 sec
10.000	0.20 sec	2.37 sec
50.000	27.47 sec	11.71 sec
100.000	51.00 sec	23.31 sec
200.000	1 min 51.24 sec	46.89 sec

Tabella 5.3: Confronto interrogazione denormalizzata 32 bit

- Interrogazione PC a 64 bit:

Documenti inseriti	Query MySQL	Query MongoDB
100	0.02 sec	46 ms
500	0.03 sec	202 ms
1.000	0.09 sec	405 ms
5.000	0.13 sec	1.68 sec
10.000	0.48 sec	3.53 sec
50.000	0.91 sec	16.88 sec
100.000	1.78 sec	34.12 sec
200.000	3.63 sec	1 min 8.61 sec
300.000	23.48 sec	1 min 48.22 sec
500.000	54.16 sec	2 min 49.53 sec
1.000.000	1 min 18.45 sec	6 min 4.02 sec
2.000.000	3 min 36.33 sec	13 min 56.61 sec
3.000.000	8 min 12.69 sec	18 min 22.76 sec

Tabella 5.4: Confronto interrogazione denormalizzata 64 bit

5.2.4 Analisi dei risultati dei test delle prestazioni

Nel grafico di Figura 5.3 e di Figura 5.4 vengono confrontati gli inserimenti sul database MySQL denormalizzato e sul database MongoDB su entrambi i PC con sistema operativo a 32 e 64 bit.

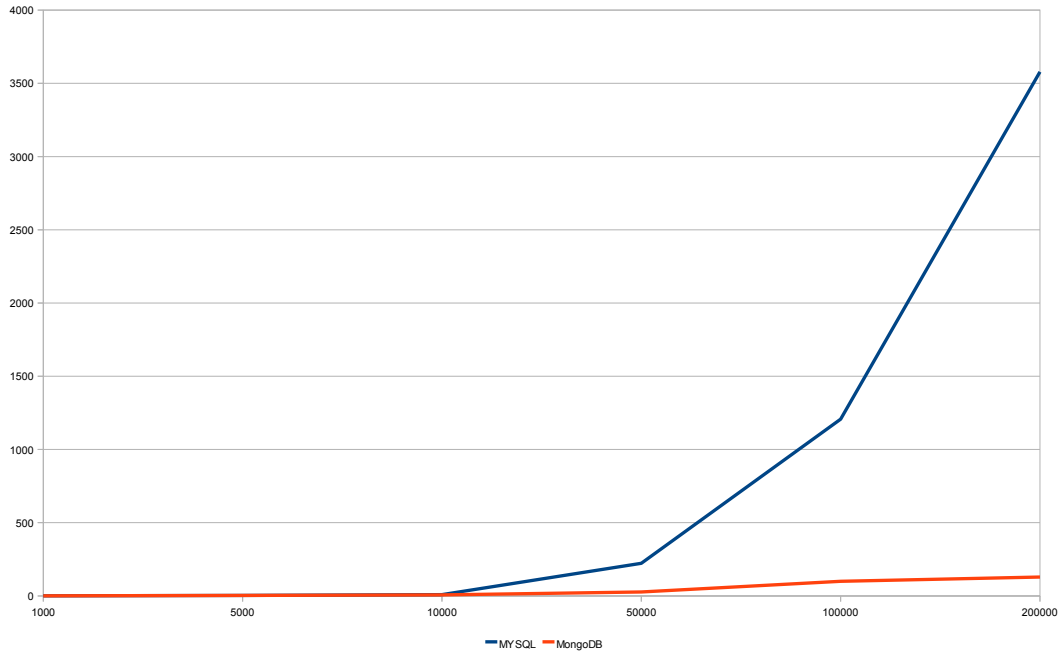

Figura 5.3: Confronto inserimenti MySQL denormalizzato e Mongodb 32 bit

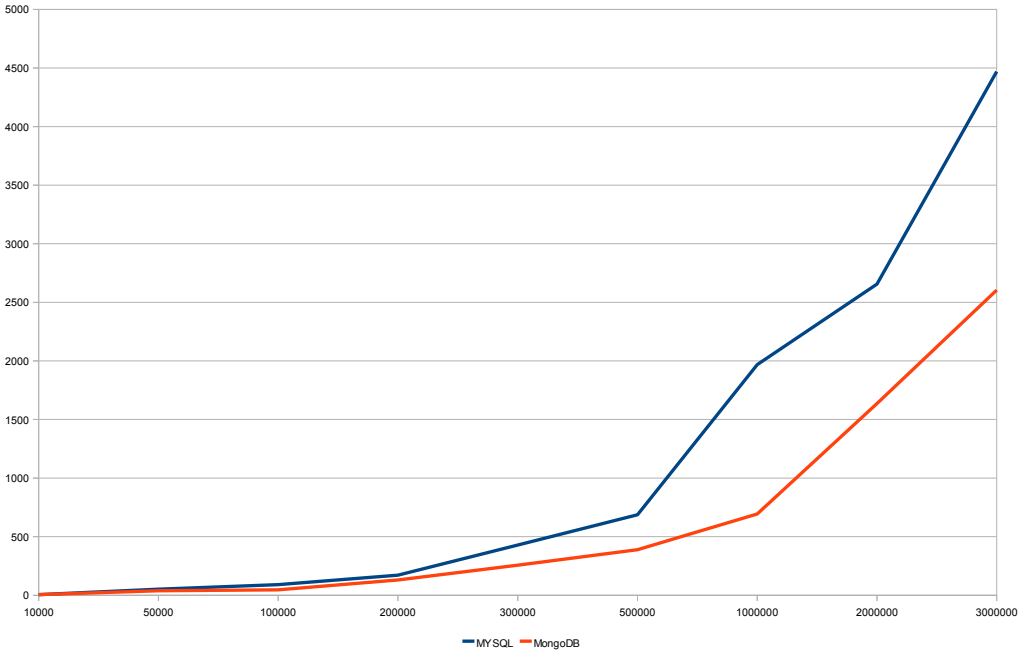

Figura 5.4: Confronto inserimenti MySQL denormalizzato e Mongodb 64 bit

Come si può notare dai grafici i tempi di inserimento di MySQL sono molto maggiori rispetto a quelli di inserimento in MongoDB nonostante si siano eleminati i vincoli di chiave estarne. Questo comportamento è giustificato dal fatto che la denormalizzazione del database ha obbligato all'inserimento di dati duplicati, nello specifico nella tabella orders sono presenti dati presenti anche nelle tabelle users, product, address e price.

Pertanto si può concludere che quando si denormalizza un database per un potenziale aumento di prestazioni in lettura, bisogna tener conto anche del costo che si ha durante l'inserimento dei dati, dato che si stanno memorizzando dati ridondanti in più locazioni del database, perciò se da un lato si ho un guadagno nei tempi di esecuzione delle operazioni di lettura dall'altro le prestazioni in scrittura possono diminuire.

Nel grafico di Figura 5.5 vengono riportati i risultati dei tempi di esecuzione relativi alle interrogazioni sui database nel PC con sistema operativo a 32 bit. Dato che il numero di documenti all'interno della collezione `orders` di MongoDB e il numero di righe all'interno della tabella `orders` di MySQL sono gli stessi si è potuto valutare

le prestazioni di interrogazione a parità di dati inseriti nella collezione e nella tabella. Nel grafico oltre ai tempi di esecuzione delle interrogazioni effettuate sul database MySQL denormalizzato sono presenti anche i tempi di esecuzione dell'interrogazione sul database MySQL normalizzato.

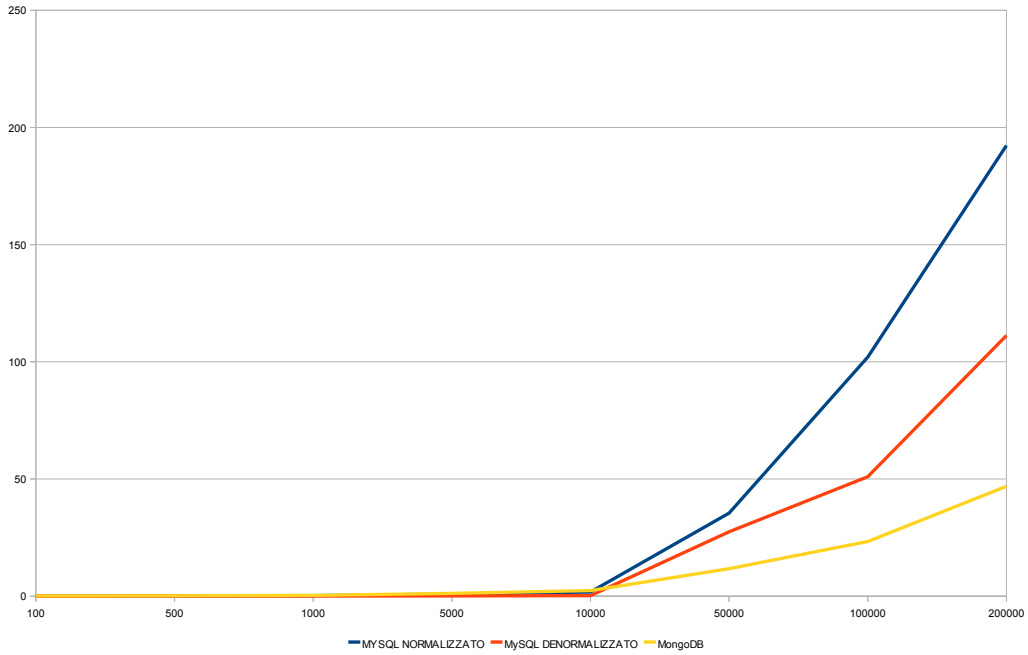

Figura 5.5: Confronto metodi di interrogazione 32 bit

Come si può notare dal grafico di Figura 5.5, denormalizzando il database MySQL si ottiene un miglioramento di circa il 40% considerando l'inserimento di 200.000 documenti. Da notare è il vistoso calo di prestazioni dopo l'inserimento di più di 10.000 righe inserite nella tabella `orders`. Al contrario MongoDB complessivamente risulta avere prestazioni più lineari, anche se per un numero di documenti che va da 100 a 10.000 il database MySQL denormalizzato risulta avere prestazioni migliori.

Invece nel grafico di Figura 5.6 vengono riportati i tempi di esecuzione delle interrogazioni effettuate sul PC con sistema operativo a 64 bit per il database MySQL normalizzato, denormalizzato e per MongoDB.

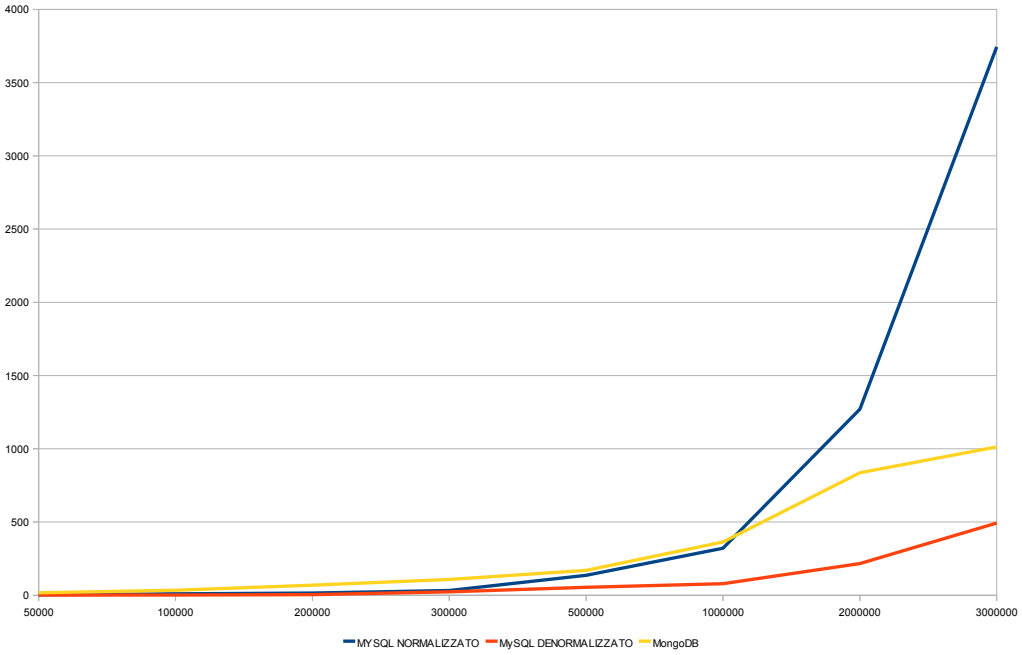

Figura 5.6: Confronto metodi di interrogazione 64 bit

Come si può notare dal grafico di Figura 5.6, a parità di inserimenti, nel PC con sistema operativo a 64 bit l'interrogazione sul database MySQL denormalizzato rivela ottime prestazioni, risultando molto più veloce rispetto all'interrogazione sul database di MongoDB. A differenza del caso del PC a 32 bit non si è verificato un calo vistoso delle prestazioni di interrogazioni ad un certo numero di righe inseriti, cosa che succede invece per i tempi di esecuzione del database MySQL normalizzato quando si raggiunge la soglia di 1.000.000 di documenti.

Quindi possiamo concludere che denormalizzando il database MySQL è possibile ottenere dei notevoli miglioramenti in fase di lettura dei dati. Nel PC con sistema operativo a 32 bit le prestazioni sono migliori fino a 10.000 righe inserite nella tabella `orders`, dopo di che i tempi di interrogazione aumentano quasi in modo esponenzialmente, pertanto è possibile considerare le 10.000 righe come punto critico dopo il quale le prestazioni calano.

Invece nel PC con sistema operativo a 64 bit questo punto critico non è stato trovato, in quanto fino a 3.000.000 di righe inserite le prestazioni di interrogazione risultano cinque volte più veloci rispetto ai tempi di interrogazione sul database

MongoDB.

Bisogna tenere presente che pur avendo ottenuto dei notevoli miglioramenti in fase di lettura denormalizzando il database, si sono introdotti dati ridondanti e duplicati presenti in più locazioni del database, trasferendo la responsabilità di mantenere i dati consistenti, in fase di inserimento e aggiornamento, dal database all'applicazione che risulterà di conseguenza più complessa.

Capitolo 6

Conclusioni

In questa tesi si sono voluti studiare i database NoSQL nati come alternativa ai database relazionali, si è partiti con una trattazione generale degli aspetti principali che hanno portato al loro sviluppo e alle caratteristiche che li accomunano, andando più nello specifico presentando una panoramica dei database NoSQL più rappresentativi.

Come caso di studio si è preso in esame MongoDB, database NoSQL orientato ai documenti, se ne sono studiate le principali caratteristiche, ovvero il nuovo modello di dati basato su documenti JSON, il linguaggio di interrogazione basato su JavaScript, inoltre sono stati presentati alcuni strumenti messi a disposizione da MongoDB come la replicazione, gli indici secondari e la scalabilità orizzontale.

Si è proseguito effettuando una trattazione dettagliata del linguaggio di interrogazione di MongoDB, spiegando e descrivendo il funzionamento e la sintassi delle operazioni di inserimento, aggiornamento e cancellazione. Sono stati presentati anche i più importanti operatori di interrogazione e le funzioni di aggregazione messe a disposizione dal linguaggio.

Lo scopo principale della tesi è quello di effettuare un'analisi delle prestazioni di MongoDB per quanto riguarda le operazioni di scrittura e di lettura dei dati. Per questo motivo MongoDB è stato messo a confronto con il database relazionale MySQL tramite l'implementazione di un caso reale.

È stato progettato un database e-commerce per la vendita di prodotti da giardinaggio implementato prima con il nuovo modello di dati di MongoDB basato sui documenti e poi tramite un'operazione di reverse engineering è stato realizzato il corrispettivo database relazionale con MySQL, tramite l'acquisizione dei requisiti strutturati, la creazione dello schema ER, la traduzione verso lo schema logico e la relativa implementazione in SQL.

Una volta creati i due database si è passati all'analisi delle prestazioni divisa in due fasi.

La prima è stata la valutazione delle prestazioni in inserimento dei due database.

Per ciascun database sono stati effettuati gli inserimenti con diversi metodi, si è valutato il metodo di inserimento più efficiente ed infine i metodi più efficienti dei due database sono stati messi a confronto. È emerso che MongoDB è molto più efficiente in fase di inserimento, prendendo come esempio il confronto dei metodi sul PC con sistema operativo a 64 bit è risultato che MongoDB è venticinque volte più veloce di MySQL.

La seconda fase di test comprende il popolamento dei due database, la formulazione e l'applicazione delle interrogazioni tramite il map-reduce per MongoDB e tramite il linguaggio SQL per MySQL e il reperimento dei tempi di esecuzione.

Prima di tutto sono state applicate le due interrogazioni ai database per esaminare il decadimento delle prestazioni all'aumentare dei dati inseriti, si è potuto constatare che il database MySQL dopo una certa quantità di dati inseriti ha un aumento dei tempi di esecuzione quasi esponenziale, invece MongoDB all'aumentare dei dati mantiene tempi di esecuzione più lineari.

In secondo luogo si sono popolati i due database con gli stessi dati, cioè ai dati inseriti in un documento di MongoDB corrispondono gli stessi dati divisi nelle rispettive tabelle in MySQL, sono state applicate le stesse interrogazioni ed è emerso che il map-reduce di MongoDB per grandi quantità di dati memorizzati è tre o quattro volte più veloce di MySQL. Per quantità di dati poco elevate MySQL risulta essere leggermente più veloce di MongoDB.

Infine nel Capitolo 5 è stato affrontato il problema della denormalizzazione considerando il fatto che MongoDB memorizza dai ridondanti e duplicati al suo interno e in più locazioni del database, sono stati approfonditi i vantaggi e gli svantaggi che questo approccio può portare dal punto dei vista del modello di dati, delle operazioni di scrittura e lettura e dell'applicazione.

Infine è stata effettuata un'operazione di denormalizzazione dello schema del database MySQL per eliminare l'uso di join nell'interrogazione ed è emerso che le prestazioni in fase di lettura hanno un notevole miglioramento rispetto ai tempi di esecuzioni dell'interrogazione sul database normalizzato e rispetto ai tempi di esecuzione del map-reduce di MongoDB è emerso che nel PC con sistema operativo a 64 bit l'interrogazione denormalizzata risulta più veloce.

Si può concludere che MongoDB risulta essere una valida alternativa ai database relazionali grazie all'innovativo modello dei dati basato sui documenti che fornisce una maggiore interazione con i linguaggi di programmazione orientati agli oggetti, all'agile e semplice linguaggio di interrogazione e alla possibilità di essere facilmente scalabile. MongoDB è inoltre in grado di fornire ottime prestazioni in scrittura e in lettura addirittura migliori rispetto ai database relazioni secondo il caso reale trattato.

Appendice A

Progettazione fisica

A.1 Progettazione fisica database MySQL

Verrà ora riportata la formulazione in SQL dello schema della base di dati e dei vincoli di integrità.

Definizione dello schema:

```
CREATE DATABASE garden
```

Tabella PRODUCT:

```
CREATE TABLE product(
    ID CHAR(24) NOT NULL PRIMARY KEY,
    Name CHAR(32),
    Sku CHAR(4),
    Slug CHAR(16),
    Description CHAR(128)
);
```

Tabella CATEGORY:

```
CREATE TABLE category(
    ID CHAR(24) NOT NULL PRIMARY KEY,
    Slug CHAR(16),
    Name CHAR(32),
    Description VARCHAR(128)
);
```

Tabella HIERARCHY:

```
CREATE TABLE hierarchy(
    Ancestor CHAR(24) NOT NULL,
    Son CHAR(24) NOT NULL,
    PRIMARY KEY( Ancestor, Son ),
   FOREIGN KEY( Ancestor ) REFERENCES category( ID )
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    FOREIGN KEY( Son ) REFERENCES category(ID)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);
```

Tabella BELONGS:

```
CREATE TABLE belongs(
       Product CHAR(24) NOT NULL,
       Category CHAR(24) NOT NULL,
       PRIMARY KEY( Product, Category ),
       FOREIGN KEY( Product ) REFERENCES product( ID )
               ON DELETE CASCADE
               ON UPDATE CASCADE,
       FOREIGN KEY( Category ) REFERENCES category( ID )
               ON DELETE CASCADE
               ON UPDATE CASCADE
);
```

Tabella DEATAILS:

```
CREATE TABLE details(
       Model_num CHAR(8) NOT NULL,
       Product CHAR(24) NOT NULL,
       Color CHAR(16),
       Weight CHAR(8),
       Manufacturer CHAR(32),
       Weight_units CHAR(16),
       PRIMARY KEY( Model_num, Product ),
       FOREIGN KEY( Product ) REFERENCES product( ID )
               ON DELETE CASCADE
               ON UPDATE CASCADE
);
```

Tabella PRICE:

```
CREATE TABLE price(
       Product CHAR(24) NOT NULL,
       Start DATE NOT NULL,
       End DATE,
       Sale DOUBLE,
       Retail DOUBLE,
       PRIMARY KEY( Product, Start ),
       FOREIGN KEY( Product ) REFERENCES product( ID )
              ON DELETE CASCADE
              ON UPDATE CASCADE
);
```

Tabella USERS:

```
CREATE TABLE users(
       User_ID CHAR(24) NOT NULL PRIMARY KEY,
       First_name CHAR(32),
       Last_name CHAR(32),
       Password CHAR(32),
       Username CHAR(32) NOT NULL UNIQUE,
       Email CHAR(32)
);
```

Tabella ADDRESS:

```
CREATE TABLE address(
       User CHAR(24) NOT NULL,
       Name CHAR(16) NOT NULL,
       Street CHAR(32),
       City CHAR(16),
       State CHAR(16),
       Zip CHAR(8),
       PRIMARY KEY( User, Name ),
       FOREIGN KEY( User ) REFERENCES users( User_ID )
              ON DELETE CASCADE
              ON UPDATE CASCADE
);
```

Tabella PAYMENT:

```
CREATE TABLE payment(
       User CHAR(24) NOT NULL,
```

```
        Expiration_date DATE,
        Name CHAR(16) NOT NULL,
        Crypted_number CHAR(16),
        Last_four CHAR(4),
        PRIMARY KEY( User, Name ),
        FOREIGN KEY( User ) REFERENCES users( User_ID )
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Tabella REVIEW:

```
CREATE TABLE review(
        Product CHAR(24) NOT NULL,
        User CHAR(24) NOT NULL,
        Title CHAR(64),
        Date DATE,
        Text VARCHAR(512),
        Rating CHAR(4),
        PRIMARY KEY( Product, User ),
        FOREIGN KEY( Product ) REFERENCES product( ID )
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY( User ) REFERENCES users( User_ID )
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Tabella VOTE:

```
CREATE TABLE vote(
        User CHAR(24) NOT NULL,
        Product_rev CHAR(24) NOT NULL,
        User_rev CHAR(24) NOT NULL,
        PRIMARY KEY( User, User_rev, Product_rev ),
        FOREIGN KEY( Product_rev, User_rev ) REFERENCES review( Product, User )
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Tabella ORDERS:

```
CREATE TABLE orders(
       ID CHAR(24) NOT NULL PRIMARY KEY,
       State CHAR(16),
       Purchase_date DATE,
       User CHAR(24) NOT NULL,
       FOREIGN KEY( User ) REFERENCES users( User_ID )
               ON DELETE CASCADE
               ON UPDATE CASCADE
);
```

Tabella INSERTION:

```
CREATE TABLE insertion(
       Product CHAR(24) NOT NULL,
       Order_ID CHAR(24) NOT NULL,
       Quantity INTEGER,
       PRIMARY KEY( Product, Order_ID ),
       FOREIGN KEY( Product ) REFERENCES product( ID )
               ON DELETE CASCADE
               ON UPDATE CASCADE,
       FOREIGN KEY( Order_ID ) REFERENCES orders( ID )
               ON DELETE CASCADE
               ON UPDATE CASCADE
);
```

A.2 Progettazione fisica database MySQL denormalizzato

Verrà ora riportata l'implementazione in SQL dello schema logico-relazionale del database MySQL denormalizzato del Capitolo 5 Sottosezione 5.2.1.

Tabella PRODUCT:

```
CREATE TABLE product(
       ID CHAR(24) NOT NULL PRIMARY KEY,
       Name CHAR(32),
       Sku CHAR(4),
       Slug CHAR(16),
       Description CHAR(128)
);
```

Tabella PRICE:

```
CREATE TABLE price(
       Product CHAR(24) NOT NULL,
       Start DATE NOT NULL,
       End DATE,
       Sale DOUBLE,
       Retail DOUBLE,
       PRIMARY KEY( Product, Start )
);
```

Tabella USERS:

```
CREATE TABLE users(
       User_ID CHAR(24) NOT NULL PRIMARY KEY,
       First_name CHAR(32),
       Last_name CHAR(32),
       Password CHAR(32),
       Username CHAR(32) NOT NULL UNIQUE,
       Email CHAR(32)
);
```

Tabella ADDRESS:

```
CREATE TABLE address(
       User CHAR(24) NOT NULL,
       Name CHAR(16) NOT NULL,
       Street CHAR(32),
       City CHAR(16),
       State CHAR(16),
       Zip CHAR(8),
       PRIMARY KEY( User, Name )
);
```

Tabella ORDERS:

```
CREATE TABLE orders(
       ID INTEGER auto_increment NOT NULL PRIMARY KEY
       Order_ID CHAR(24),
       Order_state CHAR(16),
       Purchase_date DATE,
       User_ID CHAR(24) NOT NULL,
       Product_ID CHAR(24),
       Sku CHAR(4),
```

```
        Name CHAR(32),
        Sale DOUBLE,
        Quantity INTEGER,
        State CHAR(16),
        Zip CHAR(8),
        City CHAR(16),
        Street CHAR(32)
);
```

Tabella INSERTION:

```
CREATE TABLE insertion(
        Product CHAR(24) NOT NULL,
        Order_ID CHAR(24) NOT NULL,
        Quantity INTEGER,
        PRIMARY KEY( Product, Order_ID )
);
```

Bibliografia

[1] Chodorow, K. & Dirolf, M. (2010) *MongoDB: The Definitive Guide,* O'Reilly Media

[2] Chodorow, K. (2011) *50 Tips and Tricks for MongoDB Developers,* O'Reilly Media

[3] Banker, K. (2012) *MongoDB In Action,* New York, Manning

[4] Matthews, M. & Cole, J. & Gradecki, J. D. (2003) *MySQL and Java Developer's Guide,* Indianapolis, Wiley Publishing, Inc.

[5] Plugge, E. & Membrey, P. & Hawkins, T. (2010) *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing,* New York, Apress

[6] Atzeni, P. & Ceri, S. & Paraboschi, S. & Torlone, R. (2002) *Basi di dati Modelli e linguaggi di interrogazione*, Milano, McGraw-Hill

[7] http://cattell.net/datastores/Datastores.pdf

[8] http://www.christof-strauch.de/nosqldbs.pdf

[9] http://www.mongodb.org/

[10] http://strata.oreilly.com/2012/02/nosql-non-relational-database.html

[11] http://dev.mysql.com/doc/refman/5.1/en/innodb-storage-engine.html

[12] http://blog.indigenidigitali.com/l-ecosistema-nosql/

[13] http://nosql-database.org/

[14] http://en.wikipedia.org/wiki/Prepared_statement

[15] http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

[16] http://www.mongovue.com/2010/11/03/yet-another-mongodb-map-reduce-tutorial/

[17] http://kylebanker.com/blog/2009/12/mongodb-map-reduce-basics/

[18] http://blog.serverdensity.com/map-reduce-and-mongodb/

[19] http://blog.mongolab.com/2012/08/why-is-mongodb-wildly-popular/

[20] http://www.ovaistariq.net/199/databases-normalization-or-denormalization-which-is-the-better-technique/

[21] http://www.25hoursaday.com/weblog/CommentView.aspx?guid=324e0852-ba72-4cc4-94bb-66b553fda165