

Data modelling su database non relazionali: confronto di prestazioni su operazioni CRUD

Calcolo e confronto dei costi di elaborazione di diverse query su attributi indicizzati
Database non relazionali in questione MongoDB e CouchDB

Presentata da:
Matteo Santoro
Matricola 0000881608

Relatore:
Prof. Alessandra Lumini

Parole chiave

MongoDB

NoSQL

Embedding

Referencing

Ringraziamenti

Ringrazio la Dott.ssa Alessandra Lumini, la mia relatrice per la tesi, per l'interessante progetto che mi ha dato l'opportunità di svolgere e la sua disponibilità nel lavoro svolto.

Ringrazio i miei genitori per avermi sempre sostenuto in questo percorso, senza di loro non avrei potuto fare quel che sono riuscito a fare.

Ringrazio la mia ragazza per essermi stata vicina in questo percorso, aver condiviso tutto e la ringrazio per il sostegno che mi ha dato.

Ringrazio i miei amici, colleghi per un solo anno ma amici per tanti altri, con i quali ho condiviso parte del mio percorso e le mie avventure fuori e dentro questo campus.

Ringrazio i miei colleghi di corso, colleghi di progetti ed un mio amico di vecchia data che è sempre stato in grado di sostenermi in ogni mia decisione e con il quale ho condiviso sempre tutto.

Sommario

I database sono diventati una parte importante del quotidiano, ogni tipo di applicazione o software che siamo abituati ad utilizzare ha l'esigenza di prendere, elaborare e conservare diversi tipi di dati, è quindi fondamentale poter ottimizzare al meglio queste operazioni. Gli obiettivi di questa tesi sono lo studio comparativo di alcuni DBMS non relazionali e il confronto di diverse soluzioni di modellazione logica e fisica per database non relazionali. Utilizzando come sistemi di gestione due DBMS Document-based non relazionali, MongoDB e CouchDB, ed un DBMS relazionale, Oracle, sarà effettuata un'analisi di diverse soluzioni di modellazione logica dei dati in database documentali e uno studio mirato alla scelta degli attributi sui quali costruire indici. In primo luogo verrà definito un semplice caso di studio su cui effettuare il confronto, basato su due entità in relazione 1:N, sulle quali sarà costruito un opportuno carico di lavoro.

Il mondo dei database relazionali ha un design specifico; composti da righe, colonne e tabelle in cui le relazioni sono facilmente modellabili ed esistono regole ben precise con le quali progettare questi sistemi come ad esempio la normalizzazione che definisce un criterio di qualità per modellare i dati. Al contrario i database non relazionali sono schema-less, senza schema fisso, ed esiste una libertà maggiore di modellazione. In questo lavoro di tesi i dati verranno modellati secondo le tecniche del Referencing ed Embedding che consistono rispettivamente nell'inserimento di una chiave (riferimento) oppure di un intero sotto-documento (embedding) all'interno di un documento per poter esprimere il concetto di relazione tra diverse entità. Per studiare l'opportunità di indicizzare un attributo, ciascuna entità sarà poi composta da due triplette uguali di attributi definiti con differenti livelli di selettività, con la differenza che su ciascun attributo della seconda sarà costruito un indice.

Il carico di lavoro sarà costituito da query definite in modo da poter testare le diverse modellazioni includendo anche predicati di join che non sono solitamente contemplati in modelli documentali. Per ogni tipo di database verranno eseguite le query e registrati i tempi, in modo da poter confrontare le performance dei diversi DBMS sulla base delle operazioni CRUD.

Tra i risultati ottenuti si deve fare una piccola distinzione sul carico di lavoro, in caso di un carico ben distribuito tra query di selezione normale e di join il modello più performante risulta essere quello dell'embedding di B in A mentre ipotizzando un carico di lavoro incentrato su normali query di selezione ed effettuando un numero minore di join le soluzioni di referencing si sono dimostrate le migliori, praticamente eguagliandosi in tempi medi. Relativamente ai DBMS, MongoDB si è dimostrato il più veloce e facile da usare, con potenti strumenti e un linguaggio di interrogazione ottimo al contrario di CouchDB che

ha performance generali peggiori e un dialetto complicato da usare soprattutto in presenza di attributi indicizzati.

Il lavoro è organizzato nei seguenti capitoli. Nel primo capitolo vengono riproposti i principali concetti di modellazione dei dati in database relazionali e non, sono evidenziate le principali differenze tra i due tipi di sistemi e le tipologie di modellazione per database documentali. Nel secondo capitolo viene introdotto il caso di studio che sarà considerato in questo lavoro, dettagliandone le possibili strategie di modellazione e l'insieme delle query selezionate per essere incluse nel carico di lavoro. Nel terzo capitolo viene affrontata la vera e propria sperimentazione, con una descrizione dell'implementazione utilizzata nei diversi DBMS, e dei risultati sperimentali ottenuti. Il confronto tra i risultati degli esperimenti e l'analisi dei dati ottenuti è inclusa nel capitolo 4 assieme ad alcune considerazioni che riguardano le scelte progettuali sia logiche che fisiche. Infine, vengono riportate alcune conclusioni inerenti al lavoro svolto e alcune possibili direzioni di sviluppo futuro di questo lavoro.

Indice

1	Presentazione del problema	11
1.1	Modelli di dati	11
1.1.1	Key-Value	11
1.1.2	A grafo	11
1.1.3	Wide Column	12
1.1.4	Documentali	12
1.1.5	Relazionale	14
1.2	Differenze tra relational/non-relational DB	15
1.3	Soluzioni documentali	17
1.3.1	Modello Referencing	17
1.3.2	Modello Embedding	17
1.4	Il nostro caso di studio	18
2	Analisi benchmark	21
2.1	Modellazione dei dati	21
2.2	Query e operazioni	25
3	Valutazione	27
3.1	MongoDb	27
3.1.1	Caratteristiche	27
3.1.2	Container	29
3.1.3	Organizzazione dei dati	30
3.1.4	Inserimento (CRUD: Creation)	34
3.1.5	Selezione (CRUD: Read)	35
3.1.6	Aggiornamento (CRUD: Update)	37
3.2	CouchDb	40
3.2.1	Caratteristiche	40
3.2.2	Container	42
3.2.3	Popolamento	42
3.2.4	Indici	43
3.2.5	Inserimento (CRUD: Creation)	43

3.2.6	Selezione (CRUD: Read)	45
3.2.7	Aggiornamento (CRUD: Update)	47
3.3	Oracle	51
3.3.1	Introduzione	51
3.3.2	Server Locale	51
3.3.3	Organizzazione dei dati	52
3.3.4	Inserimento (CRUD: Creation)	53
3.3.5	Selezione (CRUD: Read)	54
3.3.6	Aggiornamento (CRUD: Update)	55
4	Analisi dei risultati	57
4.1	Risultati modellazione	57
4.1.1	Operazione di Inserimento	57
4.1.2	Confronto con calcolo dei costi	59
4.1.3	Operazione di Selezione	59
4.2	Considerazioni finali	61
4.2.1	Scelta del sistema di basi di dati non relazionale	62
4.2.2	Scelta degli attributi da indicizzare tramite indice B-Tree	62
4.2.3	Scelta della modellazione documentale più performante	64
4.3	Problematiche riscontrate nell'elaborazione	65
4.3.1	Assenza di join in CouchDb	65
4.3.2	Tempistiche CouchDB	66
4.3.3	Indicizzazione CouchDB	66
5	Conclusioni	67
5.1	Sviluppi futuri	1
Appendice A	Script	3
A.1	Script creazione Dataset	3
A.2	Script creazione immagine MongoDB	5
A.3	Script creazione indici MongoDB	7
A.4	Query update di MongoDB su python con pymongo	8
A.5	Script creazione immagine CouchDB	9
A.6	Script creazione indici per collezioni CouchDB	9
A.7	Script SQL per il setup del DB relazionale	14
A.8	Script python con xlswriter	16
Appendice B	Calcolo dei costi	19
B.1	NL	19
B.2	g	19
B.3	h	19

B.4	NP	19
B.5	Costo del nested loop	20
B.6	Indice clustered	20
B.7	Indice unclustered	20
Bibliografia		21

Elenco delle figure

1.1	Immagine dati in formato tabellare	14
1.2	Attraverso una chiave importata è possibile risalire a chi ha effettuato il dato pagamento	14
1.3	Evoluzione dei database	15
1.4	teorema CAP illustrazione a triangolo	16
1.5	Esempio modello Referencing	17
1.6	Esempio modello Embedding	18
1.7	Caso di studio modello E/R	19
3.1	Logo di mongoDB	27
3.2	Interfaccia grafica MongoDB express	29
3.3	Statistiche di inserimento su MongoDB	35
3.4	Grafico tempi di selezione MongoDB	36
3.5	Grafico di aggiornamento per i valori con selezione sugli attributi di B . . .	38
3.6	Grafico di aggiornamento per i valori con selezione sugli attributi di A . . .	39
3.7	Grafico di aggiornamento per tutti i valori con selezione su tutti gli attributi di A	39
3.8	Logo di CouchDB	40
3.9	Interfaccia grafica per CouchDB	42
3.10	Statistiche di inserimento su CouchDB	45
3.11	Grafico tempi di selezione CouchDB	46
3.12	Grafico di aggiornamento per il referencing di CouchDB	49
3.13	Grafico di aggiornamento per l' embedding di CouchDB	49
3.14	Grafico di aggiornamento per tutti gli attributi di CouchDB	50
3.15	Oracle logo	51
3.16	Services di windows, oracle ORCL deve essere attivo	52
3.17	Pannello di connessione di sql developer per poter eseguire query sul nostro server	52
3.18	Statistiche inserimento dati su Oracle	53
4.1	Grafico di inserimento per i valori dei database non relazionali	57
4.2	Grafico di inserimento per i valori dei database relazionali	58

4.3	Comparazione dati per ogni collezione di MongoDB rispetto i costi stimati	59
4.4	Confronto dati sulla collezione Referencing di A in B presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB	60
4.5	Confronto dati sulla collezione Referencing di B in A presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB	60
4.6	Confronto dati sulla collezione Embedding di A in B presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB	61
4.7	Confronto dati sulla collezione Embedding di B in A presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB	61
4.8	Confronto dati Attributo 1	62
4.9	Confronto dati Attributo 2	63
4.10	Confronto dati Attributo 3	63
4.11	Confronto dati con carico di lavoro sulle query di join	64
4.12	Confronto dati aumentando il carico di lavoro delle query no join	64
4.13	Confronto dati sulle collezioni in MongoDB	65
5.1	Grafico con carico di lavoro massimo su operazioni di selezioni senza join .	68
5.2	Schema di base per un eventuale modellazione ibrida	2

Elenco delle tabelle

3.1	Tabella tempi selezione di MongoDB	37
3.2	Tabella excel con i risultati in millisecondi delle query di aggiornamento sugli attributi di B	38
3.3	Tabella excel con i risultati in millisecondi delle query di aggiornamento sugli attributi di A	38
3.4	Tabella tempi aggiornamento di MongoDB	40
3.5	Tabella tempi selezione di CouchDB	47
3.6	Tabella tempi aggiornamento di CouchDB	50
4.1	Tabella comparazione dimensione e tempo inserimento documenti in MongoDB	58

Capitolo 1

Presentazione del problema

1.1 Modelli di dati

Le basi di dati sono un sistema di gestione dei dati in grado di gestire una grande mole di dati in modo condiviso, persistente, affidabile che rispettano uno standard.

Uno degli standard più noto è il modello relazionale, tipicamente si fa riferimento a sistemi che utilizzano SQL mentre negli ultimi tempi stanno prendendo piede i database non relazionali NoSQL di cui fanno parte numerosi sistemi che si suddividono a loro volta in base al modello del dato.

1.1.1 Key-Value

Ogni collezione contiene una lista di coppie chiave-valore, i quali possono essere ogni tipo di dato come stringhe, interi, URL o BLOB (binary large object), sono dei dizionari.

Database d'esempio:

- Amazon DynamoDB
- Berkeley DB

1.1.2 A grafo

I database basati sul modello a grafo contengono uno o più grafi, i quali a loro volta contengono vertici e archi che rappresentano rispettivamente entità e relazioni dirette tra di loro. Uno degli impieghi migliori per i database a grafo sono le basi di dati per i Social Network che data la natura delle relazioni tra utenti si adattano perfettamente al caso d'uso.

Database d'esempio:

- Neo4J
- OrientDB

1.1.3 Wide Column

Nei database wide-column sono contenute una o più famiglie di colonne che corrispondono a tabelle, ognuna contiene una lista di righe nella forma di coppie chiave-valore, in cui la chiave identifica la riga relativa ad una colonna la quale al proprio interno contiene i dati anch'essi in modalità chiave-valore.

Database d'esempio:

- Google Big table
- Hbase

1.1.4 Documentali

Le basi di dati documentali utilizzano dei documenti composti da coppie di chiavi-valore dove ogni chiave è associata ad uno e un solo valore all'interno della collezione di documenti, in modo simile ad un dizionario.

```
{"first_name" : "John",  
  "last_name" : "Doe",  
  "address" : "New York"},  
  
{"first_name" : "Benjamin",  
  "last_name" : "Button",  
  "address" : "Chicago"},  
  
{"first_name" : "Mycroft",  
  "last_name" : "Holmes",  
  "address" : "London"}
```

All'interno di queste collezioni le relazioni non esistono ma possono essere comunque modellati alcuni concetti, rifacendoci all'esempio di **sopra** archiviando all'interno delle nostre collezioni anche i pagamenti aggiungendo un array per ogni impiegato se presenta dei pagamenti.

```
{"employee_id" : 1,  
  "first_name" : "John",  
  "last_name" : "Doe",  
  "address" : "New York",  
  "payments" : [  
    {"id" : 1,  
     "amount" : 50000,
```

```

        "date" : "01/12/2017"},
    {"id" : 2,
     "amount" : 20000,
     "date" : "01/13/2017"}
  ]
},

{"employee_id" : 2,
 "first_name" : "Benjamin",
 "last_name" : "Button",
 "address" : "Chicago",
 "payments" : [
   {"id" : 3,
    "amount" : 75000,
    "date" : "01/14/2017"}
 ]
},

{"employee_id" : 3,
 "first_name" : "Mycroft",
 "last_name" : "Holmes",
 "address" : "London",
 "payments" : [
   {"id" : 4,
    "amount" : 40000,
    "date" : "01/15/2017"},
   {"id" : 5,
    "amount" : 20000,
    "date" : "01/17/2017"},
   {"id" : 6,
    "amount" : 25000,
    "date" : "01/18/2017"}
 ]
}

```

I DBMS key-value solitamente non hanno un linguaggio standard come SQL, per questo questi tipi di DB vengono chiamati NoSQL, per poter effettuare interrogazioni che vengono quindi effettuate a livello di applicazione tramite particolari dialetti.

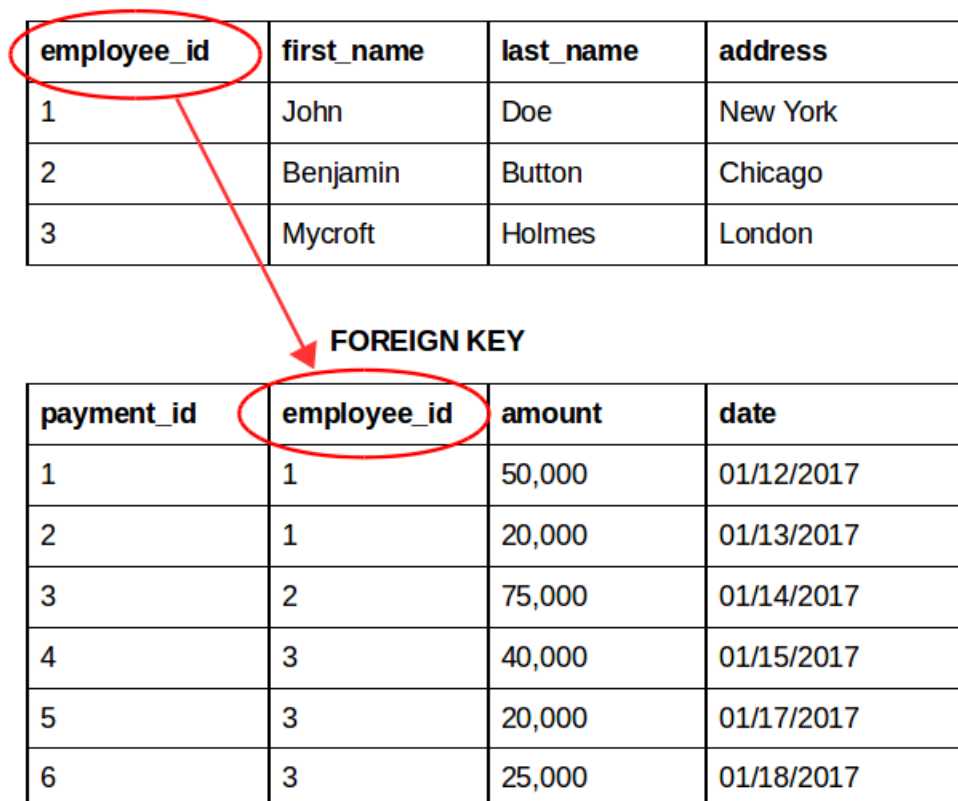
1.1.5 Relazionale

I database relazionali si basano sul modello relazionale, introdotto da E.F.Codd, nel quale i dati sono organizzati in strutture con righe e colonne a formare tabelle. Le righe sono definite **tuple** mentre le colonne sono gli **attributi** come si può vedere a figura 1.1.

employee_id	first_name	last_name	address
1	John	Doe	New York
2	Benjamin	Button	Chicago
3	Mycroft	Holmes	London

Figura 1.1: Immagine dati in formato tabellare

Le relazioni tra tabelle, modellano i legami tra diversi concetti come in figura 1.2, ad esempio se si volesse legare ogni impiegato ad una spesa, si dovrebbe creare una relazione one-to-many con la quale si esprime il fatto che un dato impiegato può effettuare molteplici pagamenti.



employee_id	first_name	last_name	address
1	John	Doe	New York
2	Benjamin	Button	Chicago
3	Mycroft	Holmes	London

FOREIGN KEY

payment_id	employee_id	amount	date
1	1	50,000	01/12/2017
2	1	20,000	01/13/2017
3	2	75,000	01/14/2017
4	3	40,000	01/15/2017
5	3	20,000	01/17/2017
6	3	25,000	01/18/2017

Figura 1.2: Attraverso una chiave importata è possibile risalire a chi ha effettuato il dato pagamento

Per poter effettuare interrogazioni su **RDBMS**(relation DB management systems) si utilizza **SQL**(structured query language) [1] un linguaggio standardizzato per poter eseguire le operazioni CRUD e interrogazioni complesse all'interno di un sistema.

1.2 Differenze tra relational/non-relational DB

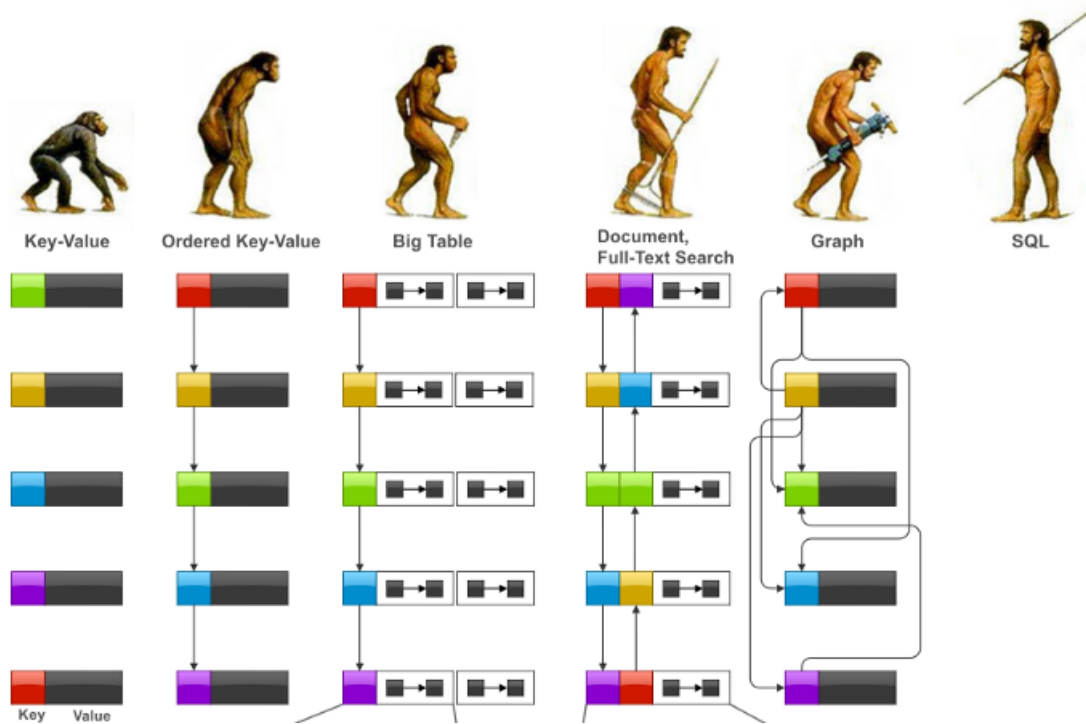


Figura 1.3: Evoluzione dei database

I database non relazionali (figura 1.3) si sono resi popolari per due ragioni:

- Possono scalare orizzontalmente(aumentando i nodi/computer) per adattarsi ai Big Data
- La versatilità dei loro schemi per potersi adattare ad ogni esigenza, al contrario degli schemi rigidi dei RDBMS

La scalabilità orizzontale si può eseguire tenendo però in considerazione il teorema di CAP (figura 1.4), infatti secondo questo teorema un sistema di basi di dati distribuito può garantire almeno 2 delle seguenti 3 proprietà:

- **Consistency** ogni nodo in un cluster distribuito restituisce lo stesso risultato; ogni lettura è quella relativa alla più recente operazione di scrittura in ogni nodo.
- **Availability** Ogni nodo non in errore restituisce una risposta per tutte le richieste di lettura e scrittura in un ragionevole lasso di tempo.

- **Partition tolerance** Il cluster non smette di funzionare nonostante un numero di messaggi persi o in ritardo tra i nodi.

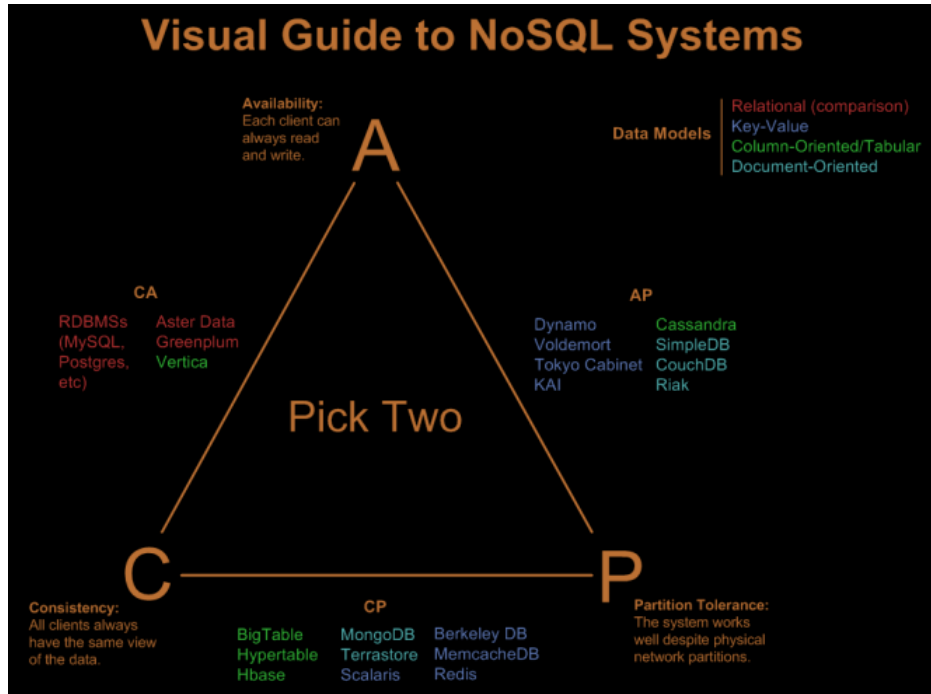


Figura 1.4: teorema CAP illustrazione a triangolo

La proprietà di consistency definita nel teorema di CAP e nelle transazioni ACID è differente, nella seconda la consistency parla dell'integrità del dato (è il dato ad essere consistente) mentre nel primo caso indica lo stato dei nodi consistenti ognuno con l'altro in ogni momento.

I relational-DBMS sono progettati per veloci transazioni che modificano il valore di svariate tuple all'interno di tabelle con una notevole complessità, basti pensare all'esigenza dello sviluppo dei trigger che si occupano di mantenere le tabelle aggiornate e semplificare la vita del SYS admin.

Le query SQL sono espressive e dichiarative, è possibile concentrarsi su cosa una transazione deve realizzare, i RDBMS si occuperanno di **come farlo**, ottimizzeranno la query usando l'algebra relazionale e troveranno il miglior piano di accesso.

Le basi di dati NoSQL sono progettate per gestire in modo efficiente una quantità di dati di gran lunga superiore ai RDBMS, non esistono vincoli relazionali e non devono esserci tabelle, offrono prestazioni maggiori rinunciando in genere alla consistency.

L'accesso ai dati viene effettuato in molti casi attraverso delle API REST e i linguaggi di query non posso ancora definirsi maturi come SQL quindi bisogna occuparsi, quando si scrive una query, di **cosa fare** e **come farlo** in modo efficiente.

1.3 Soluzioni documentali

Data l'impossibilità dell'inserimento di vincoli relazionali nei database NoSQL documentali, le opzioni per poter mantenere una relazione tra entità diverse sono 2:

1.3.1 Modello Referencing

Consiste nel collegare un documento ad un'altro all'interno di una stessa collezione, inserendo un campo che diventerà comune ad entrambi i documenti creando di fatto una relazione tra di essi, a condizione però che il campo sia unico, quindi ad esempio sia un id (identificatore) o una chiave primaria che sappiamo essere univoca (figura 1.5).

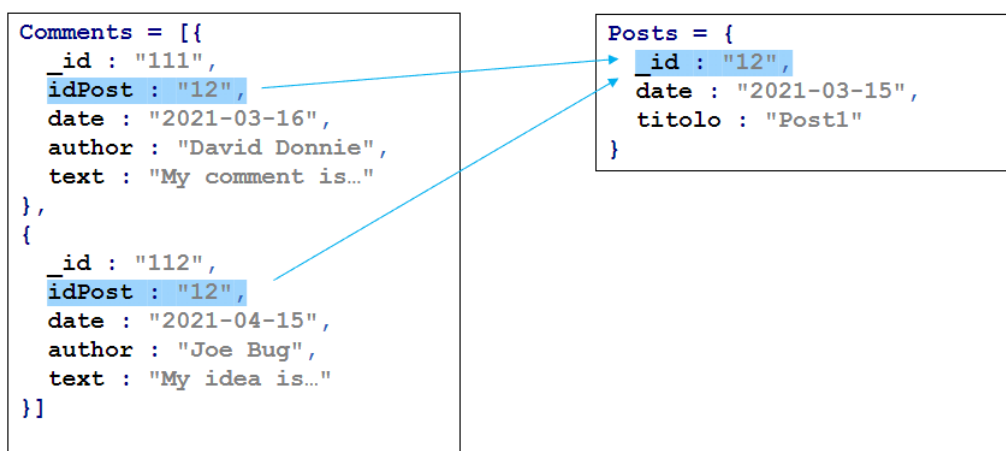


Figura 1.5: Esempio modello Referencing

1.3.2 Modello Embedding

Unisce tutti i documenti che sono in una qualche relazione tra loro all'interno di uno soltanto, creando di fatto dei sotto-documenti che possono essere, se omogenei, a loro volta immagazzinati in un vettore per comodità (figura 1.6).

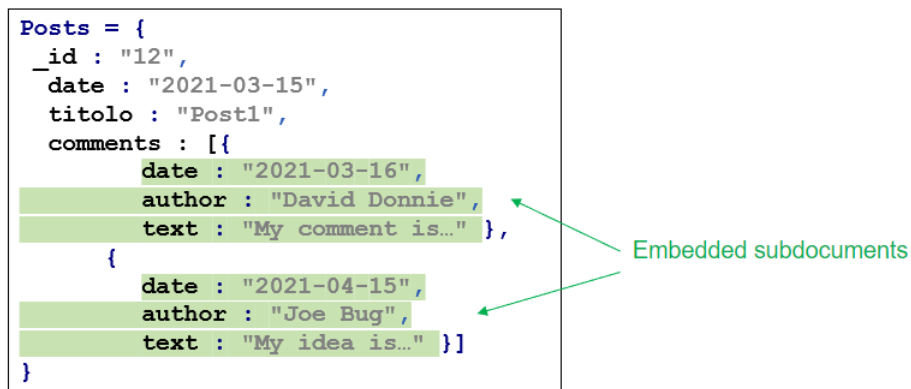


Figura 1.6: Esempio modello Embedding

1.4 Il nostro caso di studio

In questa tesi verranno utilizzati MongoDB e CouchDB come piattaforme di test per database non relazionali mentre Oracle fornirà i dati per la modellazione relazionale del problema.

Per studiare e confrontare le diverse soluzioni di modellazione dei dati nei database documentali si è deciso di utilizzare un semplice caso di studio che include due concetti correlati da una relazione uno a molti.

Il database in figura 1.7 è formato da 2 entità A e B. Per facilitare la comprensione si può pensare ad un blog in cui l'entità A rappresenta i Post e l'entità B i Commenti relativi ad ogni post, collegati da una relazione 1..N -> 1..1, per ogni A esistono 10 elementi di B, ovvero ogni Post ha ricevuto 10 Commenti.

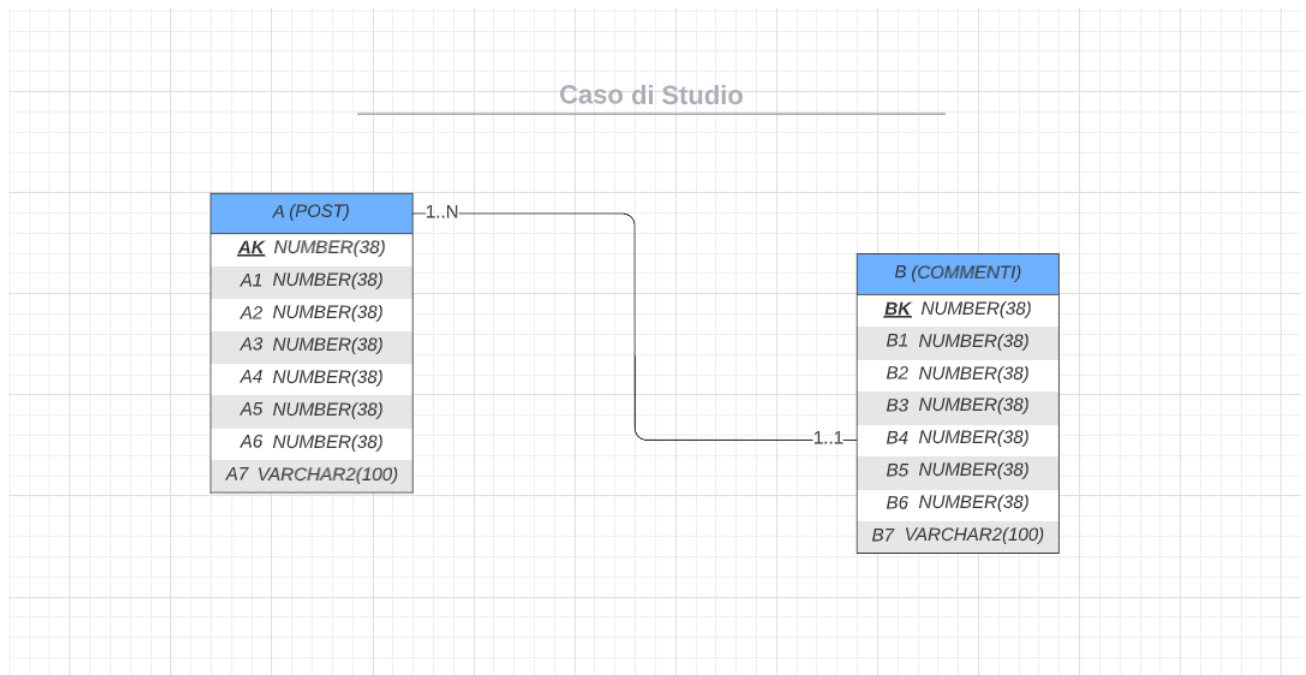
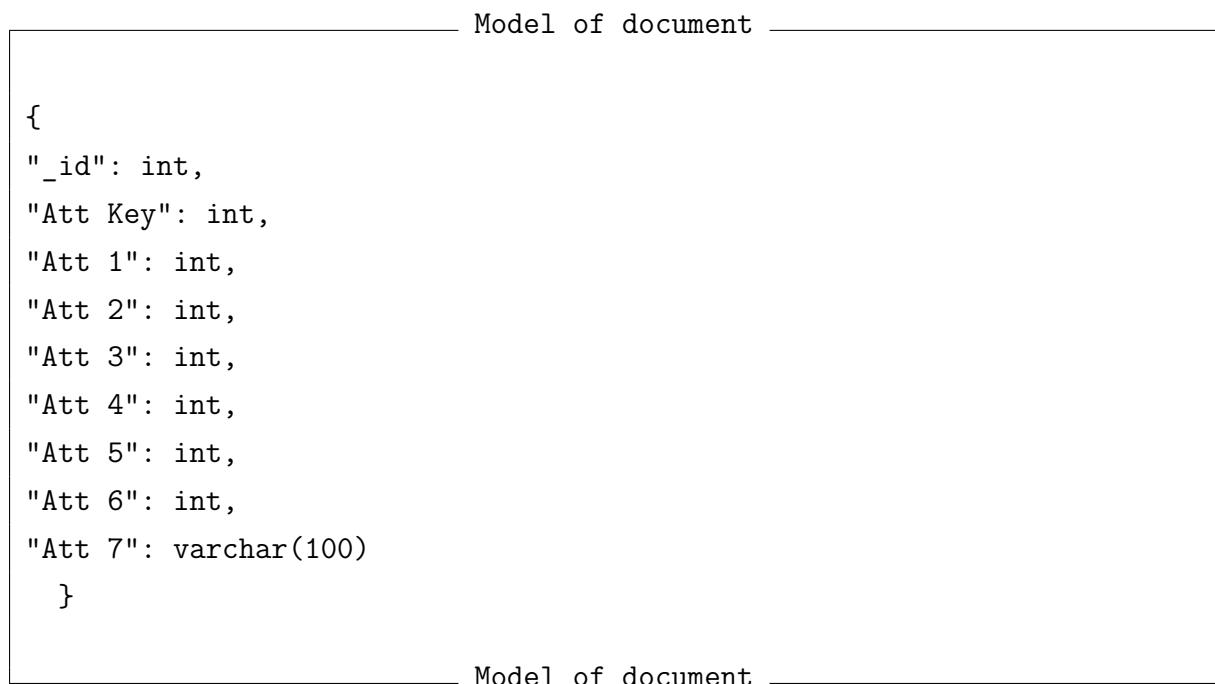


Figura 1.7: Caso di studio modello E/R

Riguardo la modellazione delle entità, ciascuna contiene 2 triplette di attributi ([A|B][1..3|4..6]) con diverso valore di selettività, con la differenza che sulla seconda tripletta sono stati costruiti degli indici su ogni attributo. Inoltre ciascuna entità contiene un campo testuale, inserito per regolare la dimensione del documento, che non viene utilizzato come chiave di ricerca.



Definiti a priori il numero di documenti delle 2 entità entità A e B, si procede alla creazione degli attributi secondo le seguenti linee per la selettività, dove N_X indica il numero di valori distinti dell'attributo X :

$$N_A = 10^{expA}$$

$$N_B = 10^{expB}$$

$$N_{A1} = N_{A4} = \frac{N_A}{10}$$

$$N_{A2} = N_{A5} = \frac{N_A}{10^{round(\frac{expA}{2})}}$$

$$N_{A3} = N_{A6} = \frac{N_A}{10^{expA-1}}$$

$$N_{B1} = N_{B4} = \frac{N_B}{10}$$

$$N_{B2} = N_{B5} = \frac{N_B}{10^{round(\frac{expB}{2})}}$$

$$N_{B3} = N_{B6} = \frac{N_B}{10^{expB-1}}$$

Nei test effettuati sono stati usati i seguenti valori:

$$expA = 5$$

$$expB = 6$$

$$size(A7) = size(B7) = 100 \text{ bytes}$$

Capitolo 2

Analisi benchmark

Il punto di partenza è la creazione dei dataset, eseguita attraverso uno script in python nel quale vanno impostate le grandezze delle entità e vengono generati i file contenenti A e B.

Per vari motivi sono state fatte alcune variazioni di questo script, in particolare per MongoDB e CouchDB lo script crea i dataset in formato JSON e li inserisce in due file sui quali verranno poi effettuate delle operazioni per poter ottenere le collezioni per i nostri casi di studio. Mentre per inserire i dati su Oracle non sono bastati dei file con righe in SQL per l'inserimento di tuple ma è stato necessario creare file di tipo .csv per velocizzare questo inserimento che sarebbe stato altrimenti troppo lento.

Gli script per la creazione dei dataset sono riportati in appendice A.1

2.1 Modellazione dei dati

La modellazione dei dati ha un ruolo fondamentale nel calcolo dei costi di accesso. In questo lavoro sono state valutate 4 diverse rappresentazioni basate sulle soluzioni base dell' **Embedding** e del **Referencing**. Utilizzando un sistema non relazionale esiste la possibilità di esprimere la relazione tra A e B in un modo differente, attraverso il referencing colleghiamo le un entità inserendo un riferimento della seconda e viceversa, per poter poi eseguire ad esempio query di join o voler recuperare tutti i dati dell'entità referenziata c'è il bisogno di mantenere all'interno del database la collezione contenente i rimanenti attributi. Es. nel referencing di B all'interno di A, ci sono due collezioni ossia A con la Foreign Key (BK) relativa a B e l'intera collezione B.

Referencing di A in B

```

{
  "_id": int,
  "BK": int,
  "AK": int,
  "B1": int,
  "B2": int,
  "B3": int,
  "B4": int,
  "B5": int,
  "B6": int,
  "B7": varchar(100)
}

{
  "_id": int,
  "AK": int,
  "A1": int,
  "A2": int,
  "A3": int,
  "A4": int,
  "A5": int,
  "A6": int,
  "A7": varchar(100),
}
```

Referencing di A in B

Referencing di B in A

```

{
  "_id": int,
  "AK": int,
  "A1": int,
  "A2": int,
  "A3": int,
  "A4": int,
  "A5": int,
  "A6": int,
  "A7": varchar(100),
  "B": [
    int,
    .
    .
    int
  ]
}

{
  "_id": int,
  "BK": int,
  "B1": int,
  "B2": int,
  "B3": int,
  "B4": int,
  "B5": int,
  "B6": int,
  "B7": varchar(100)
}
```

Referencing di B in A

La soluzione Embedding consiste nel creare un'unica collezione di documenti che racchiudono al loro interno sia documenti di A che documenti di B.

```

_id": int,
"B1": int,
"B2": int,
"B3": int,
"B4": int,
"B5": int,
"B6": int,
"B7": varchar(100),
"A":
{
  "AK": int,
  "A1": int,
  "A2": int,
  "A3": int,
  "A4": int,
  "A5": int,
  "A6": int,
  "A7": varchar(100)
}

```

```

_id": int,
"AK": int,
"A1": int,
"A2": int,
"A3": int,
"A4": int,
"A5": int,
"A6": int,
"A7": varchar(100),
"B": [
  {
    "BK": int,
    "B1": int,
    "B2": int,
    "B3": int,
    "B4": int,
    "B5": int,
    "B6": int,
    "B7": varchar(100)
  },
  .
  .
  {
    "BK": int,
    "B1": int,
    "B2": int,
    "B3": int,
    "B4": int,
    "B5": int,
    "B6": int,
    "B7": varchar(100)
  }
]

```

2.2 Query e operazioni

Per poter ottenere statistiche da ogni database, sono state decise delle query legate alle operazioni CRUD.

Le prime ad essere eseguite sono state quelle di **inserimento** di dati, i cui risultati si trovano rispettivamente nelle sezioni

- Inserimento per MongoDB
- Inserimento per CouchDB
- Inserimento per Oracle

Le query di **lettura** invece sono le query su cui si è lavorato maggiormente, è stata testata la lettura di ogni attributo di entrambe le entità, utilizzando come chiavi di ricerca sia i campi indicizzati che non, escludendo l'attributo di tipo testuale x7 che serve per poter conservare una descrizione del Post o del Commento.

```
select A.*  
from A  
where Ax='val'
```

```
select B.*  
from B  
where Bx='val'
```

Sono inoltre state eseguite delle query di join che selezionassero campi di entrambi gli oggetti:

```
select A.*, B.*  
from A join B on (A.AK=B.AK)  
where Ax='val'
```

```
select A.*, B.*  
from A join B on (A.AK=B.BK)  
where Bx='val'
```

Le chiavi di ricerca sono valori randomizzati, ottenuti tramite una funzione Python che prende in entrata il campo di ricerca e ne restituisce un valore valido.

```
def get_random_indexed_int_A(ind, N_Att1, N_Att2, N_Att3):
    if "1" in ind or "4" in ind :
        return randint(0, N_Att1 - 1)
    elif "2" in ind or "5" in ind:
        return randint(0, N_Att2 - 1)
    elif "3" in ind or "6" in ind:
        return randint(0, N_Att3 - 1)
```

Le query di **update** sono state strutturate in modo abbastanza simile a quelle di lettura e non richiedono join.

```
update A
set Ay = 'v'
where Ax='val'
```

```
update B
set By = 'v'
where Bx='val'
```

Le statistiche all'interno di questa tesi saranno riportate in forma tabellare o grafica, in entrambi i casi i parametri saranno le collezioni sulle quali sono eseguite le query e gli indici tramite i quali si selezionano i risultati, per differenziare le query di join da quelle senza predicato di join si è deciso di utilizzare come formalismo quello del nome dell'attributo seguito da una j, quindi se ad esempio in una tabella ho un valore in colonna Referencing B in A riga A5j corrisponde alla query:

```
select Referencing_B_in_A.* AS A, B.*
from A join B on (A.AK=B.FAK)
where A5= 'val'
```

Capitolo 3

Valutazione

3.1 MongoDB



Figura 3.1: Logo di mongoDB

3.1.1 Caratteristiche

MongoDB [2] è uno dei più diffusi database non relazionali orientato ai documenti, di tipo NoSQL, utilizza documenti in un formato JSON al posto delle tipiche tabelle dei sistemi relazionali. Più precisamente MongoDB utilizza i BSON ossia JSON binari per rappresentare strutture dati semplici e array associativi (oggetti in MongoDB), un BSON contiene una lista ordinata di elementi appartenenti ai seguenti tipi:

- stringhe
- interi (32 o 64 bit)
- double (numeri a virgola mobile a 64 bit, standard IEEE 754)
- date (numeri interi in millisecondi dal'epoca Unix come riferimento, 1^o gennaio 1970)
- byte array (dati binari)

- booleani (true e false)
- NULL
- oggetto BSON
- array BSON
- espressioni regolari
- codice JavaScript

Queste sono alcune delle caratteristiche importanti di MongoDB:

- **Supporta query ad hoc:** In MongoDB, si può cercare per campo, query di intervallo e supporta ricerche tramite regular expression.
- **Indicizzazione:** E' possibile indicizzare qualsiasi campo in un documento.
- **Replica:** supporta la replica Master Slave. Un master può eseguire letture e scritture e uno slave copia i dati dal master e può essere utilizzato solo per letture o backup (non scritture)
- **Duplicazione dei dati:** MongoDB può essere eseguito su più server. I dati vengono duplicati per mantenere il sistema attivo e anche in condizioni di funzionamento in caso di guasto hardware.
- **Bilanciamento del carico:** Ha una configurazione di bilanciamento del carico automatico a causa dei dati inseriti negli shard.
- Utilizza **JavaScript** invece di Procedure.
- È un database senza schema scritto in C++.
- Fornisce prestazioni elevate.
- Memorizza facilmente file di qualsiasi dimensione senza complicare lo stack.
- Supporta inoltre:
 - Modello di dati JSON con schemi dinamici
 - Partizionamento automatico per scalabilità orizzontale
 - Replica integrata per un'elevata disponibilità

3.1.2 Container

Il server di MongoDB è stato utilizzato all'interno di un container docker, per poter limitare la quantità di memoria cache utilizzata e poter quindi effettuare delle misurazioni più vicine possibili ai casi ideali in cui i documenti devono essere recuperati dalla memoria di massa. Si è creato un container a partire dall'immagine di MongoDB scaricabile da **dockerHub** che è stata successivamente modificata per adattarsi meglio al progetto tramite il seguente docker-compose in cui:

- sono state aggiunte variabili d'ambiente per username e password, per root e user normale
- condivise cartelle e file utili come per la creazione di indici o script per semplificare il tutto
- healthcheck per poter assicurarsi che i server funzionino sempre correttamente
- immagine di MongoDB express [3] per poter avere un interfaccia grafica che fornisse ulteriori informazioni (figura 3.2)

Mongo Express Database: tirocinio

Viewing Database: tirocinio

Collections				Collection Name	+ Create collection
View	Export	[JSON]	Import	A	Del
View	Export	[JSON]	Import	B	Del
View	Export	[JSON]	Import	embedding_A_in_B	Del
View	Export	[JSON]	Import	embedding_B_in_A	Del
View	Export	[JSON]	Import	referencing_A_in_B	Del
View	Export	[JSON]	Import	referencing_B_in_A	Del

Database Stats	
Collections (incl. system.namespaces)	6
Data Size	5.75 GB
Storage Size	613 MB
Avg Obj Size #	1.74 KB
Objects #	3300000
Indexes #	38
Index Size	2.48 GB

Figura 3.2: Interfaccia grafica MongoDB express

Il file docker-compose.yml per la creazione e la messa in atto dell'immagine di MongoDB è riportato in appendice A.2

3.1.3 Organizzazione dei dati

Creazione collezioni

Il caricamento dei dati è stato effettuato tramite importazione di 2 enormi file JSON (si veda paragrafo 2 creazione dei dati) contenenti le 2 collezioni POST (A) e COMMENTI (B). I file sono organizzati in modo da caricare i dati organizzati secondo il modello *"Referencing di A in B"*. All'interno della collezione B la Foreign key di A è già presente alla creazione dopo la generazione con lo script in Python, le altre collezioni si ottengono con le seguenti query in MongoDB. Il parametro \$out posto dopo la query determina la materializzazione della vista.

Referencing

Il referencing di A in B si ottiene facilmente proiettando gli attributi della collezione B, dato che contiene al suo interno la foreign Key di A.

Referencing di A in B

```
db.B.aggregate([
  {
    $project: {
      "_id" : "$BK",
      "BK" : "$BK",
      "AK" : "$FAK",
      "B1" : "$B1",
      "B2" : "$B2",
      "B3" : "$B3",
      "B4" : "$B4",
      "B5" : "$B5",
      "B6" : "$B6",
      "B7" : "$B7"
    }
  }, {
    $out : "referencing_A_in_B"
  }
])
```

Il referencing di B in A invece richiede un join, un' operazione più onerosa per la quale si è dovuto passare il parametro *allowDiskUse=true* per poter utilizzare anche la memoria secondaria. L'operazione ricerca in B tutte le chiavi di A e le aggiunge all'interno del documento A all'array B che arriverà a contenere 10 chiavi BK per ogni AK, dato che il rapporto e' di 1:10.

Referencing di B in A

```
db.B.aggregate(  
[  
  {  
    $group: {  
      _id: {"AK" : "$FAK"}, "BK": {$addToSet : "$BK"}  
    }  
  },  
  {  
    $lookup: {  
      from: 'Ap',  
      localField: '_id.AK',  
      foreignField: 'AK',  
      as: 'AK'  
    }  
  },  
  {  
    $project : {"_id" : 0,"BK.FAK" : 0}  
  },  
  {  
    $unwind: {  
      path: "$AK",  
    }  
  },  
  {  
    $project : {  
      "_id" : "$AK.AK",  
      "AK" : "$AK.AK",  
      "A1" : "$AK.A1",  
      "A2" : "$AK.A2",  
      "A3" : "$AK.A3",  
      "A4" : "$AK.A4",  
      "A5" : "$AK.A5",  
      "A6" : "$AK.A6",  
      "A7" : "$AK.A7",  
      "B" : "$BK"}  
    }  
  },  
  {
```

```

    $out : "referencing_B_in_A"
  }
],{allowDiskUse:true}
)

```

Embedding

Per l'embedding di documenti si è dovuto eseguire comunque dei join ma è stato fondamentale poter aggiungere un intero documento all'interno di un'array tramite il parametro `$$ROOT` che ha permesso un'operazione molto più veloce e compatta.

Embedding di A in B

```

db.B.aggregate(
[
  {
    $lookup: {
      from: 'A',
      localField: 'FAK',
      foreignField: 'AK',
      as: 'A'
    }
  }
],{
  $project: {
    "FAK" : 0,
    "_id" : 0,
    "A._id" : 0
  }
},{
  $project : {
    "_id" : "$BK",
    "B1" : 1,
    "B2" : 1,
    "B3" : 1,
    "B4" : 1,
    "B5" : 1,
    "B6" : 1,
    "B7" : 1,
    "A" : 1
  }
}
)

```

```

    }
  }
  ,{
    $out : "embedding_A_in_B"
  }
]
)

```

Embedding di B in A

```

db.B.aggregate(
[
p{
  $group: {
    _id: {"AK" : "$FAK"}, "BK": {$addToSet : "$$ROOT"}
  }
},
{
  $lookup: {
    from: 'A',
    localField: '_id.AK',
    foreignField: 'AK',
    as: 'AK'
  }
},{
  $project : {"_id" : 0,"BK.FAK" : 0}
},{
  $unwind: {
    path: "$AK",
  }},{
  $project : {
    "_id" : "$AK.AK",
    "AK" : "$AK.AK",
    "A1" : "$AK.A1",
    "A2" : "$AK.A2",
    "A3" : "$AK.A3",
    "A4" : "$AK.A4",
    "A5" : "$AK.A5",
    "A6" : "$AK.A6",
  }
}
]
)

```

```

    "A7" : "$AK.A7",
    "B" : "$BK"}
  },{
    $project : { "A._id" : 0, "B._id" : 0}
  },{
    $out : "embedding_B_in_A"
  }
],{allowDiskUse:true}

```

Indici

Gli indici per gli attributi sono stati creati attraverso uno script in JS che viene eseguito durante la creazione del container.

MongoDB utilizza di default i B-tree come strutture dati per gli indici dei B-tree. Su tutte le collezioni sono stati definiti indici sulle chiavi primarie (A0 e B0) e su 3 chiavi secondarie (A4-6 e B 4-6). Per le collezioni embedded sono stati indicizzati anche gli attributi dei documenti innestati sui quali saranno eseguite delle query, mentre per il referencing è stato definito un indice sul campo importato.

Lo script per creare gli indici è riportato in appendice A.3

3.1.4 Inserimento (CRUD: Creation)

L'applicativo di MongoDB su un sistema Linux contiene già al suo interno gli eseguibili *mongoimport* e *mongoexport* che sono stati utilizzati per poter caricare documenti all'interno delle collezioni dati.

In particolare sono necessarie alcune accortezze rispetto alla normale esecuzione dato che questo progetto ha utilizzato la tecnologia docker, è diventato quindi necessario fornire particolari autorizzazioni, creando quindi il seguente comando

```

mongoimport -collection $COLLECTION -db tirocinio -u $USERNAME -p $PASSWORD \
-host=localhost:27017 -authenticationDatabase admin /path/to/file.json

```

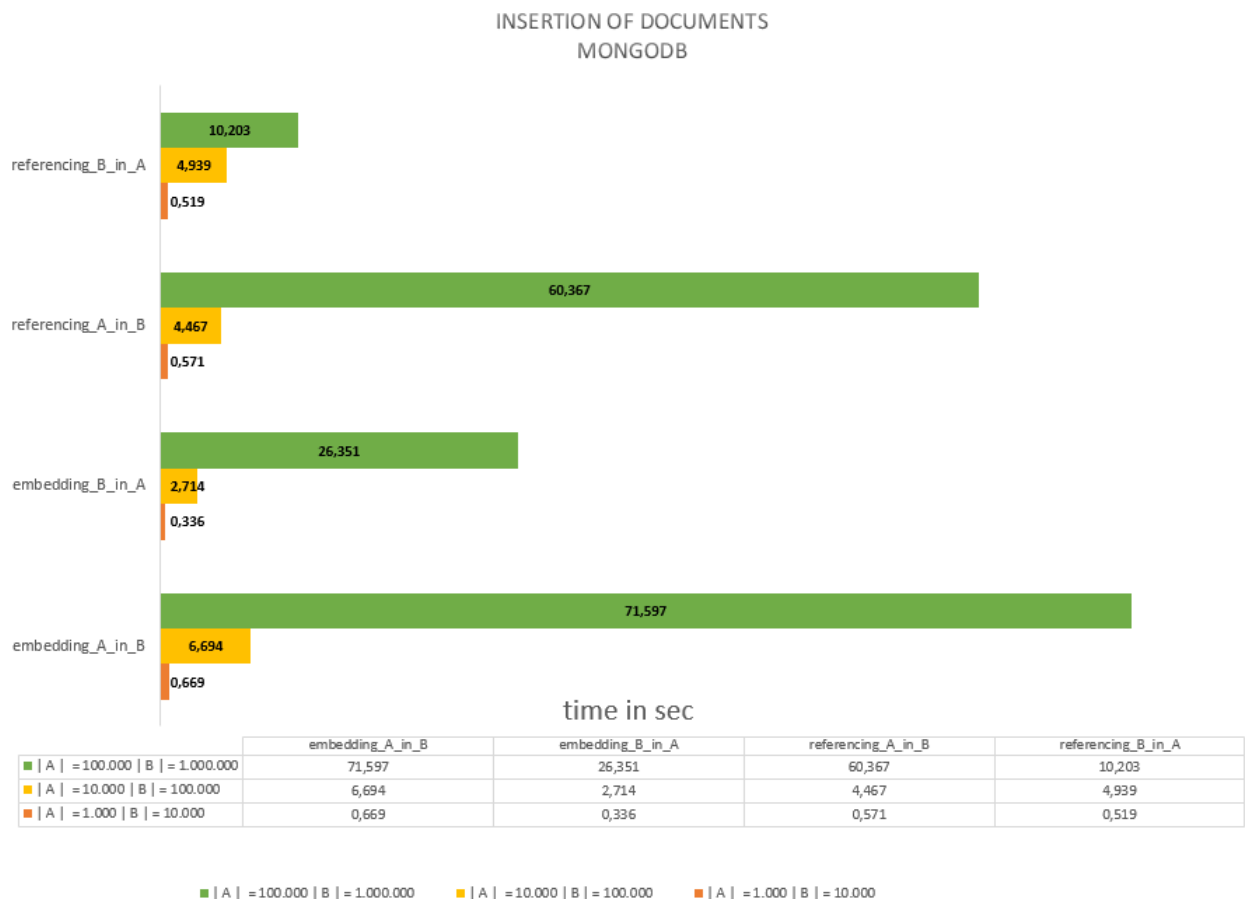


Figura 3.3: Statistiche di inserimento su MongoDB

In figura 3.3 sono riportati i tempi di inserimento per le diverse collezioni....

3.1.5 Selezione (CRUD: Read)

Il grafico in figura 3.4 e la tabella 3.1 riportano i tempi di esecuzione, nell'asse delle ascisse e nella prima colonna della tabella sono riportati gli attributi sui quali sono state eseguite le query, gli attributi di join terminano con la lettera j (Es. A6j) mentre l'asse delle ordinate indica il tempo di esecuzione in millisecondi.

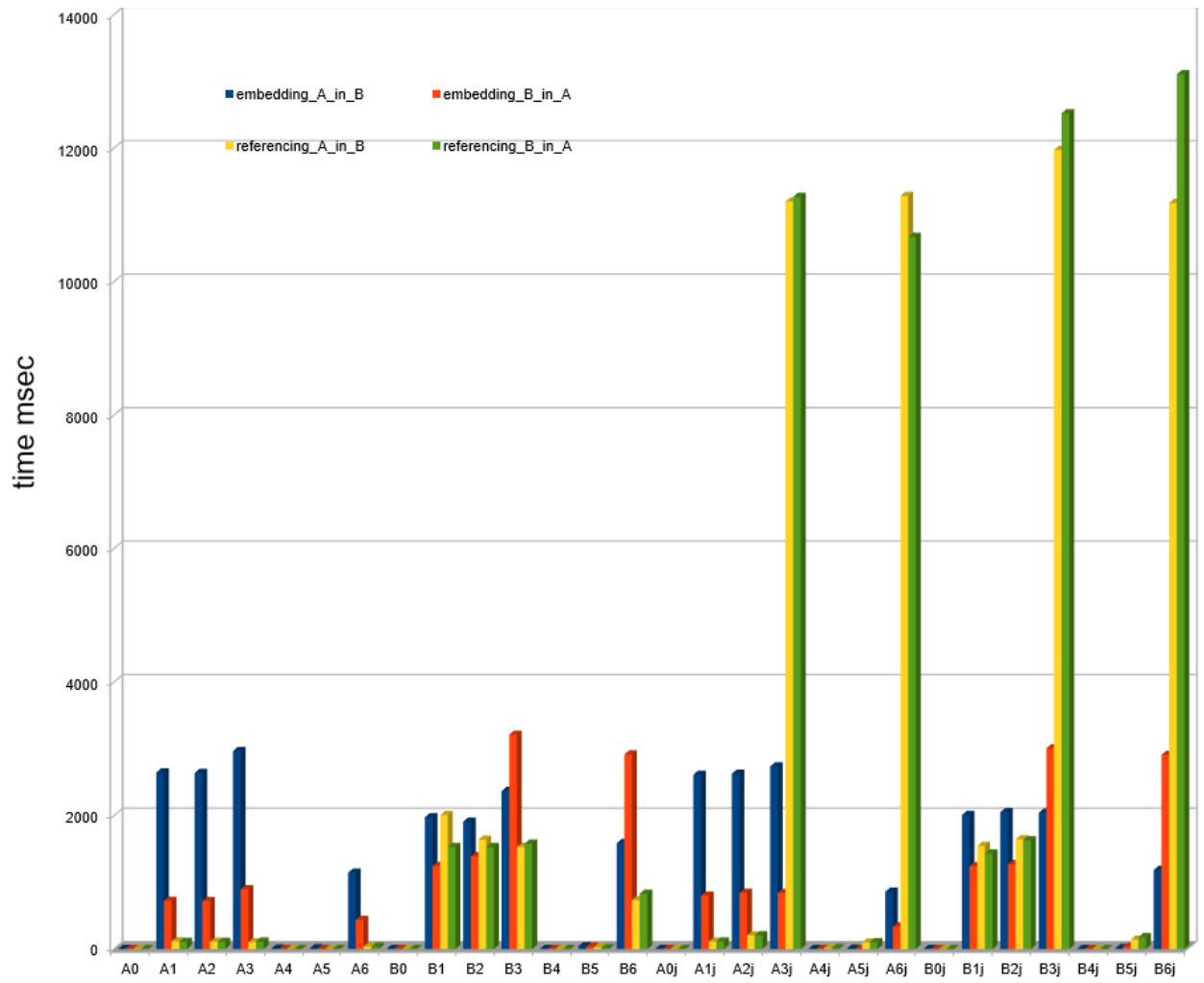


Figura 3.4: Grafico tempi di selezione MongoDB

	embedding_A_in_B	embedding_B_in_A	referencing_A_in_B	referencing_B_in_A
A0	0.3	1.3	0.5	0.4
A1	2645.9	723	115.8	108.3
A2	2638.3	718.3	110.7	109
A3	2966.9	894.8	101.6	113
A4	7	1.3	0.4	1.9
A5	11.2	2.8	2	1.9
A6	1145.6	436.9	35.2	34.7
B0	1.5	0.6	0.4	0.7
B1	1974.1	1243.6	2006.1	1528.4
B2	1908.3	1390.8	1639.8	1527.1
B3	2366.2	3211.2	1529.1	1580
B4	2.3	1.2	1.1	1.1
B5	41	32.8	26.9	12.2
B6	1586.3	2917.1	731.5	828.1
A0j	0	0	2.3	0.8
A1j	2611.7	800.2	109.3	114.2
A2j	2625.6	843.2	205.7	206.8
A3j	2735.9	841.1	11197.6	11268.6
A4j	0.7	1	10.3	12.7
A5j	5.6	0.7	98.8	103.5
A6j	858.5	335.8	11283.8	10668.2
B0j	0	0	0.4	0
B1j	2007.9	1240.5	1543.6	1430.2
B2j	2050.5	1270.5	1643.8	1629.7
B3j	2041	3005.6	11971	12520.2
B4j	0.2	0.1	1.9	1.3
B5j	14.2	32.9	140.9	176.6
B6j	1186	2908.9	11176.7	13104.5

Tabella 3.1: Tabella tempi selezione di MongoDB

3.1.6 Aggiornamento (CRUD: Update)

I documenti sono stati aggiornati usando come chiave di ricerca ciascuno degli attributi di ogni collezione: di conseguenza un update con filtro su A5 o A2 ad esempio, modificherà $\frac{N_A}{10^{\text{round}(\frac{expA}{2})}} = 100$ documenti. In appendice A.4 sono riportate alcune query d'esempio per effettuare l'update su mongoDB tramite modulo pymongo.

Dalle tabelle possiamo mettere a confronto l'operazione di update sugli stessi documenti sia in forma embedded che referenced e tramite un grafico è evidente quanto la modalità embedded sia maggiormente onerosa nel caso in cui gli attributi non siano indicizzati mentre si hanno risultati pressochè identici se la ricerca viene effettuata con un indice.

Tabella 3.2: Tabella excel con i risultati in millisecondi delle query di aggiornamento sugli attributi di B

	embedding_B_in_A	referencing_B_in_A
B0	2	12
B1	22	114
B2	0	0
B3	0	0
B4	0	4
B5	0	1
B6	0	0

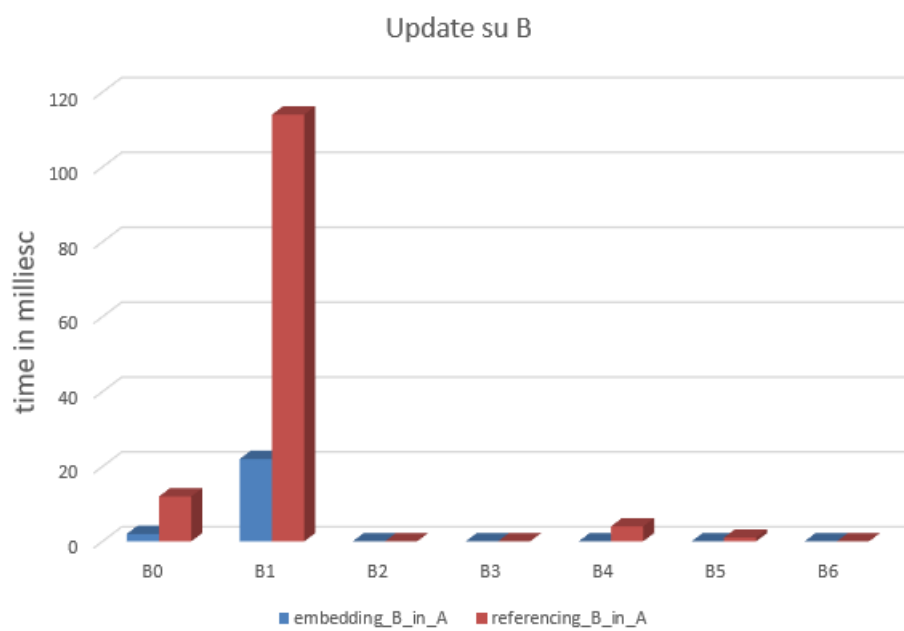


Figura 3.5: Grafico di aggiornamento per i valori con selezione sugli attributi di B

Tabella 3.3: Tabella excel con i risultati in millisecondi delle query di aggiornamento sugli attributi di A

	embedding_A_in_B	referencing_B_in_A
A0	1	1
A1	190	8
A2	3	1
A3	0	0
A4	1	1
A5	0	1
A6	1	0

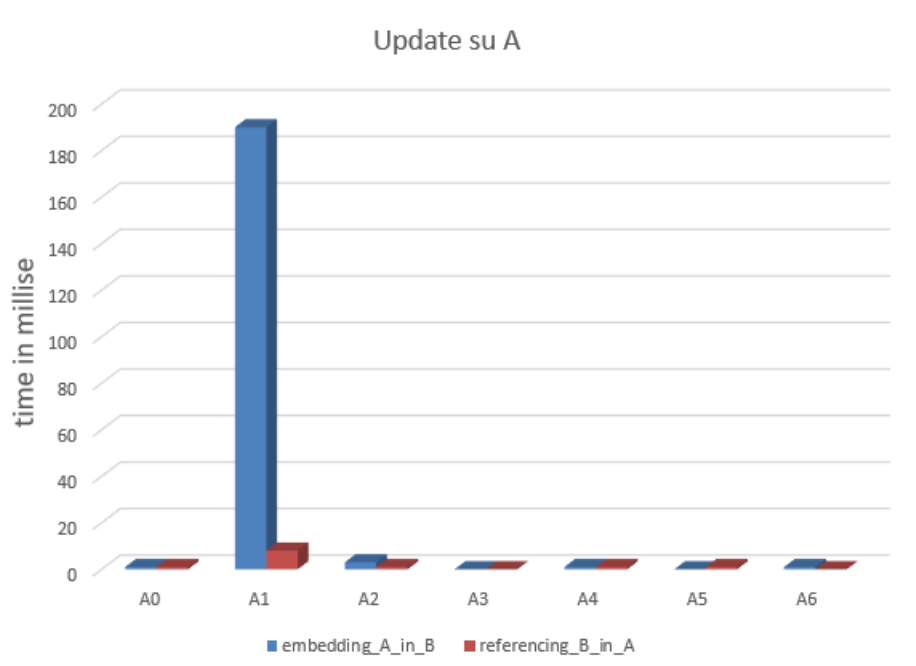


Figura 3.6: Grafico di aggiornamento per i valori con selezione sugli attributi di A

In figura 3.7 e tabella 3.4 sono visibili i tempi per tutti gli attributi di ogni documento.

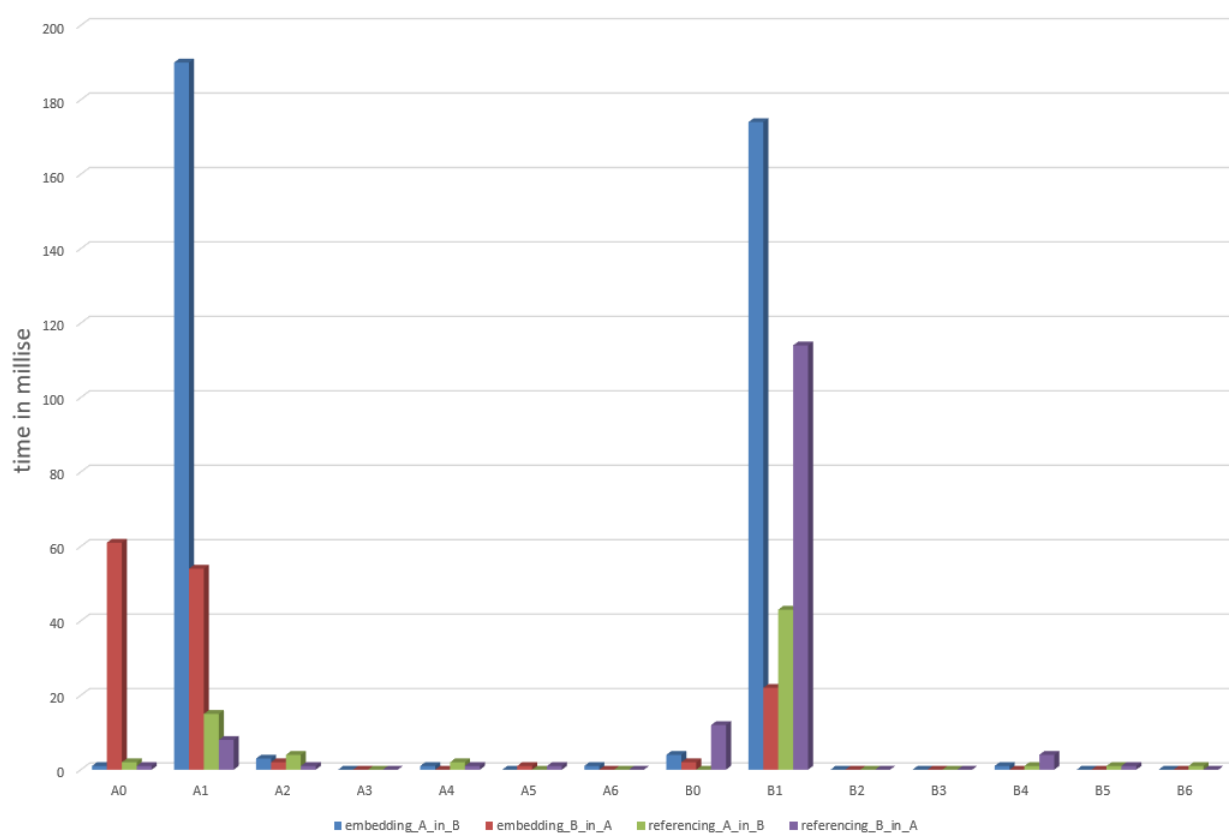


Figura 3.7: Grafico di aggiornamento per tutti i valori con selezione su tutti gli attributi di A

Tabella 3.4: Tabella tempi aggiornamento di MongoDB

	embedding_A_in_B	embedding_B_in_A	referencing_A_in_B	referencing_B_in_A
A0	1	61	2	1
A1	190	54	15	8
A2	3	2	4	1
A3	0	0	0	0
A4	1	0	2	1
A5	0	1	0	1
A6	1	0	0	0
B0	4	2	0	12
B1	174	22	43	114
B2	0	0	0	0
B3	0	0	0	0
B4	1	0	1	4
B5	0	0	1	1
B6	0	0	1	0

3.2 CouchDb



Figura 3.8: Logo di CouchDB

3.2.1 Caratteristiche

CouchDB è un sistema di gestione di basi di dati non relazionali [4]. E' stato creato soprattutto per il mondo Web, progettato per poter gestire sia importanti quantità di richieste attraverso la rete, sia per poter immagazzinare i dati in dispositivi mobili ed essere veloce. Utilizza una API HTTP/JSON per poter gestire le richieste ed inviare i dati,

quindi fa ampio utilizzo di POST, PUT, DELETE e così via. E' diventato un progetto Apache nel 2008, versione OpenSource di CouchServer.

Le principali feature sono:

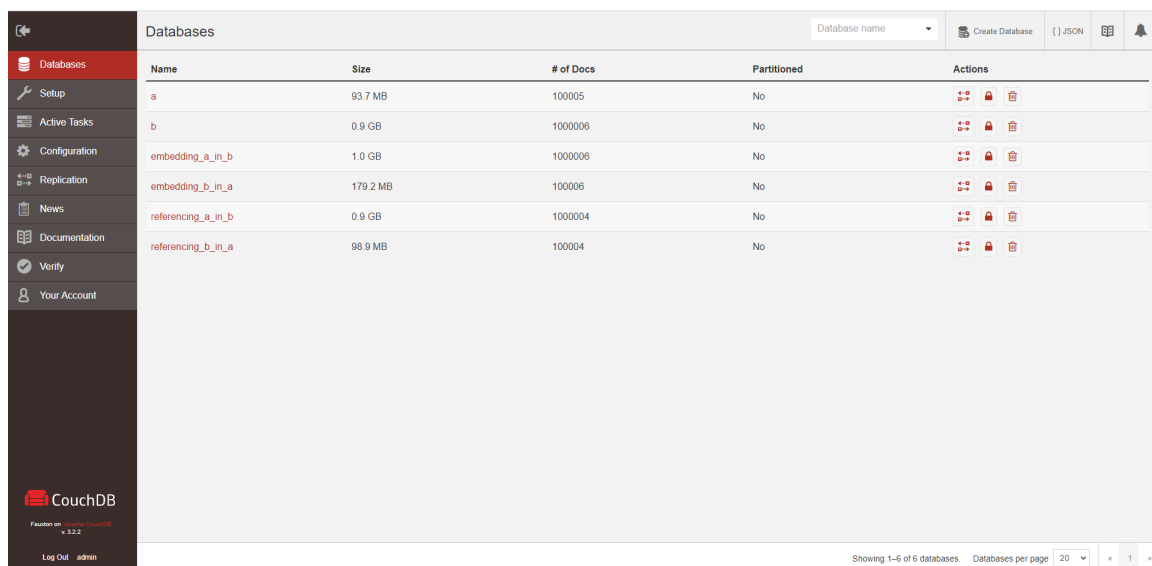
- **Documenti JSON:** CouchDB memorizza i dati nel documento JSON.
- **Interfaccia RESTful:** CouchDB esegue tutte le attività come la replica, l'inserimento di dati, ecc. tramite HTTP.
- **Replica N-Master:** CouchDB facilita l'utilizzo di un numero illimitato di master, creando diverse topologie di replica.
- **Creato per offline:** CouchDB può replicarsi su dispositivi (come telefoni Android) che possono andare offline e gestire la sincronizzazione dei dati quando il dispositivo è di nuovo online.
- **Filtri di replica:** CouchDB ti consente di filtrare con precisione i dati che desideri replicare su nodi diversi.
- **Semantica ACID:** il layout del file CouchDB segue tutte le caratteristiche delle proprietà ACID. Una volta inseriti i dati nel disco, non verranno sovrascritti. Gli aggiornamenti dei documenti (aggiungi, modifica, elimina) seguono Atomicity, ovvero verranno salvati completamente o non verranno salvati affatto. Il database non conterrà documenti parzialmente salvati o modificati. Quasi tutti questi aggiornamenti sono serializzati e un numero qualsiasi di client può leggere un documento senza attendere e senza essere interrotto.
- **Archiviazione dei documenti:** CouchDB è un database NoSQL che segue l'archiviazione dei documenti. I documenti sono l'unità di dati principale in cui ogni campo ha un nome univoco e contiene valori di vari tipi di dati come testo, numero, booleano, elenchi, ecc. I documenti non hanno un limite impostato per la dimensione del testo o il conteggio degli elementi.
- **Autenticazione e supporto della sessione:** CouchDB consente di mantenere aperta l'autenticazione tramite un cookie di sessione come un'applicazione web.
- **Sicurezza:** CouchDB fornisce anche sicurezza a livello di database. Le autorizzazioni per database sono separate in lettori e amministratore. I lettori possono leggere e scrivere nel database.
- **Convalida:** è possibile convalidare i dati inseriti nel database combinandoli con l'autenticazione per garantire che il creatore del documento sia colui che ha effettuato l'accesso.

Per eseguire query su CouchDB si utilizza Mango, un linguaggio di interrogazione JSON, molto simile a Mongo ma con alcune differenze, tra le piu' importanti c'è la mancanza dell'operatore di lookup ossia di join, infatti i tempi che presenterò successivamente sulle query di join sono sostanzialmente la somma di query di selezione sulle varie collezioni.

3.2.2 Container

Per utilizzare CouchDB, come per MongoDB, si è deciso di usare un container docker nel quale caricare i dati e organizzare un interfaccia sulla quale lavorare.

Tramite uno **script docker-compose** si è fissato un limite alla memoria disponibile per il sistema e tramite un interfaccia web-based chiamata Project Fauxton (figura 3.9), è stato possibile interagire con le collezioni di dati per testare query e amministrare generalmente il database.



The screenshot shows the CouchDB Project Fauxton web interface. On the left is a sidebar with navigation links: Databases, Setup, Active Tasks, Configuration, Replication, News, Documentation, Verify, and Your Account. The main area is titled 'Databases' and contains a table with the following columns: Name, Size, # of Docs, Partitioned, and Actions. The table lists six databases: 'a' (93.7 MB, 100005 docs), 'b' (0.9 GB, 1000006 docs), 'embedding_a_in_b' (1.0 GB, 1000006 docs), 'embedding_b_in_a' (179.2 MB, 100006 docs), 'referencing_a_in_b' (0.9 GB, 1000004 docs), and 'referencing_b_in_a' (98.9 MB, 100004 docs). Each database row has three action icons: a plus sign, a lock, and a trash can. At the bottom of the interface, it says 'Showing 1-6 of 6 databases. Databases per page 20'.

Name	Size	# of Docs	Partitioned	Actions
a	93.7 MB	100005	No	[+][lock][trash]
b	0.9 GB	1000006	No	[+][lock][trash]
embedding_a_in_b	1.0 GB	1000006	No	[+][lock][trash]
embedding_b_in_a	179.2 MB	100006	No	[+][lock][trash]
referencing_a_in_b	0.9 GB	1000004	No	[+][lock][trash]
referencing_b_in_a	98.9 MB	100004	No	[+][lock][trash]

Figura 3.9: Interfaccia grafica per CouchDB

3.2.3 Popolamento

Per popolare i database sono stati utilizzati i file JSON precedentemente creati per MongoDB e l'inserimento avviene un documento per volta, attraverso una istruzione POST all'indirizzo locale del server di CouchDB.

CouchDB richiede che il campo `_id` sia una stringa, quindi attraverso il comando `sed`, che è un comando che può essere utilizzato per leggere e adattare i flussi di dati, in uno script in bash si sono potute riutilizzare le collezione esportate da MongoDB che avevano il campo `_id` in forma numerica mentre CouchDB richiede che sia una stringa.

```
cat dataJSON/collection.json | sed -r 's/"_id":([0-9]+)/ "_id": "\1" /' \
» dataCouchDB/collection.json
```

E' stato utilizzato il seguente script in python per il caricamento di dati.

```
import requests
import json

url = "http://admin:admin@127.0.0.1:5984/b"

payload = json.dumps({
    "_id": "738878",
    "BK": 738878,
    "B1": 94207,
    "B2": 176,
    "B3": 3,
    "B4": 94207,
    "B5": 176,
    "B6": 3,
    "B7": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. .... "
})
headers = {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
}

response = requests.request("POST", url, headers=headers, data=payload)
```

3.2.4 Indici

Gli indici all'interno di ogni collezione sono stati creati tramite uno script in python con un procedimento simile a quello usato per MongoDB. Mango è un linguaggio di query dichiarativo JSON per CouchDB. Gli indici di Mango con tipo json, sono costruiti usando viste MapReduce attraverso lo script python riportato in appendice A.6

3.2.5 Inserimento (CRUD: Creation)

CouchDB utilizza una POST così strutturata per inserire un nuovo documento

Request

```
POST /referencing_a_in_b HTTP/1.1
Accept: application/json
```

Content-Length: 81
Content-Type: application/json

```
{
  "_id" : "11986",
  "BK": 11986,
  "FAK": 6680,
  "B1": 7947,
  "B2": 17,
  "B3": 2,
  "B4": 7947,
  "B5": 17,
  "B6": 2,
  "B7": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. ...."
}
```

Response

HTTP/1.1 201 Created
Cache-Control: must-revalidate
Content-Length: 71
Content-Type: application/json
Date: Tue, 13 Aug 2022 15:19:25 GMT
ETag: "1-9c65296036141e575d32ba9c034dd3ee"
Location: http://localhost:5984/db/FishStew
Server: CouchDB (Erlang/OTP)

```
{
  "id": "11986",
  "ok": true,
  "rev": "1-9c65296036141e575d32ba9c034dd3ee"
}
```

Per eseguire inserimenti multipli da python viene richiamata una istruzione di POST per ogni documento JSON del dataset. Il campo `_id`, viene definito come identificatore del documento stesso.

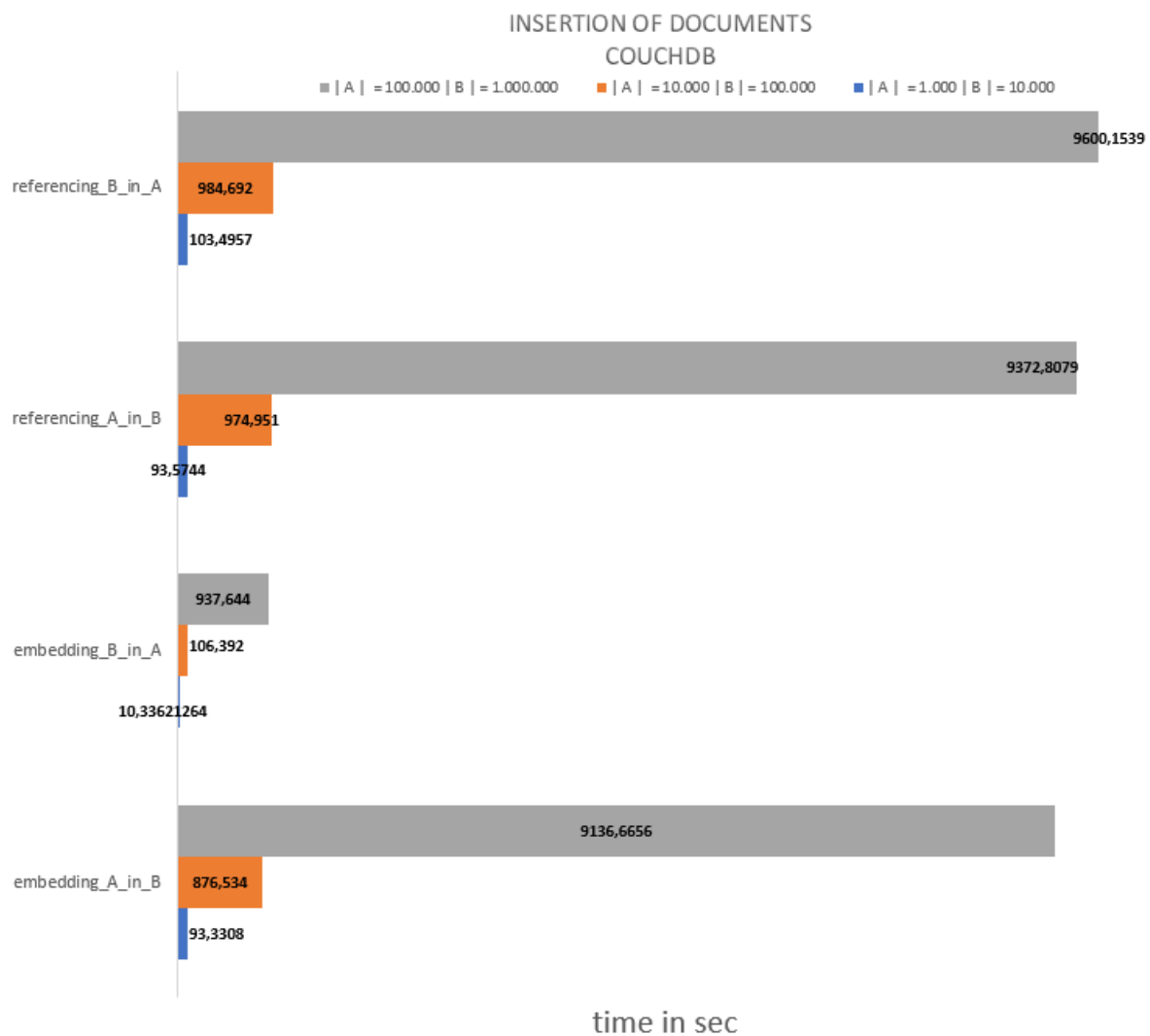


Figura 3.10: Statistiche di inserimento su CouchDB

Purtroppo è risultato che questo è il metodo di gran lunga meno efficiente (figura 3.10), sono stati inseriti un enorme quantità di dati in un volta sola e i tempi sono peggiorati molto, c'è però da precisare che la potenza di couchDB e molti altri database non relazionali si basa sui cluster, e nel nostro caso di studio ci si è limitati ad una sola istanza; nel caso di multiple istanze le request sarebbero state ripartite sui nodi e i tempi sarebbero stati ridotti di molto.

3.2.6 Selezione (CRUD: Read)

Il grafico in figura 3.11 e la tabella 3.5 riportano i tempi di esecuzione, nell'asse delle ascisse e nella prima colonna della tabella sono riportati gli attributi sui quali sono state eseguite le query, gli attributi di join terminano con la lettera j (Es. A6j) mentre l'asse delle ordinate indica il tempo di esecuzione in millisecondi.

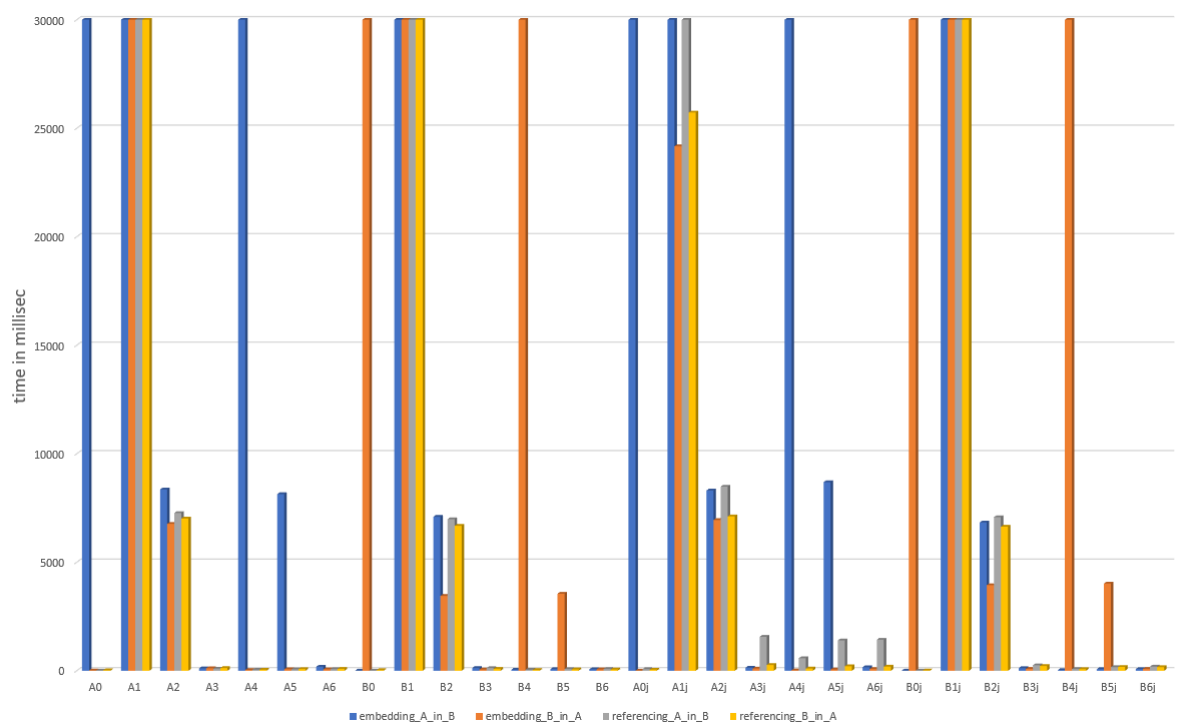


Figura 3.11: Grafico tempi di selezione CouchDB

Operando su couchDB si è notato quanto potessero essere onerose alcune operazioni e quindi i tempi si dilungassero molto, è stata effettuata la scelta di adoperare un timeout ossia un tetto massimo di tempo da non poter superare per ogni operazione di selezione, il timeout in questo caso è stato impostato a 30.000 millisecondi.

Tabella 3.5: Tabella tempi selezione di CouchDB

	embedding_A_in_B	embedding_B_in_A	referencing_A_in_B	referencing_B_in_A
A0	30000	14.986	9.143	22.209
A1	30000	30000	30000	30000
A2	8352.973	6768.937	7265.029	7015.472
A3	111.057	113.379	94.827	124.983
A4	30000	33.695	46.74	47.848
A5	8137.713	71.097	63.073	77.413
A6	183.85	66.69	74.43	84.991
B0	7.244	30000	8.022	27.317
B1	30000	30000	30000	30000
B2	7093.979	3447.942	6983.267	6681.429
B3	127.922	59.113	122.998	88.413
B4	46.944	30000	46.182	32.977
B5	73.509	3547.989	77.656	64.458
B6	67.777	64.645	80.32	58.635
A0j	30000	5.146	75.775	48.13
A1j	30000	24169.927	30000	25726.162
A2j	8306.322	6946.823	8483.63	7113.213
A3j	137.613	100.317	1563.507	266.557
A4j	30000	19.587	575.249	102.009
A5j	8686.885	55.941	1394.937	206.463
A6j	160.572	81.89	1428.167	189.117
B0j	5.126	30000	8.022	13.703
B1j	30000	30000	30000	30000
B2j	6832.531	3942.712	7074.521	6639.417
B3j	122.668	88.21	251.512	220.117
B4j	29.118	30000	81.238	75.745
B5j	73.299	4016.858	167.349	174.979
B6j	77.417	87.048	190.822	168.083

3.2.7 Aggiornamento (CRUD: Update)

Il criterio di aggiornamento usato su CouchDB è lo stesso di MongoDB: i documenti sono stati aggiornati usando come criterio di ricerca a turno un differente attributo della collezione. L'istruzione per la modifica del singolo documento richiede l'esecuzione di una PUT come segue:

Request

```
PUT /referencing_a_in_b/11986 HTTP/1.1
Accept: application/json
Content-Length: 258
Content-Type: application/json
Host: localhost:5984
```

```
{
  "_rev": "1-917fa2381192822767f010b95b45325b",
  "_id" : "11986",
  "BK": 11986,
  "FAK": 6680,
  "B1": 7947,
  "B2": 17,
  "B3": 2,
  "B4": 7947,
  "B5": 17,
  "B6": 2,
  "B7": "Lorem ipsum dolor sit amet, consectetur adipiscing elit. ...."
}
```

Response

```
HTTP/1.1 201 Created
Cache-Control: must-revalidate
Content-Length: 85
Content-Type: application/json
Date: Wed, 14 Aug 2013 20:33:56 GMT
ETag: "2-790895a73b63fb91dd863388398483dd"
Location: http://localhost:5984/recipes/SpaghettiWithMeatballs
Server: CouchDB (Erlang/OTP)
```

```
{
  "id": "11986",
  "ok": true,
  "rev": "2-790895a73b63fb91dd863388398483dd"
}
```

Una conseguenza dell'operazione di aggiornamento è la modifica dell'attributo rev (revision) del documento incrementando il numero di versione.

Attraverso i grafici 3.12 e 3.13 possiamo fare qualche considerazione:

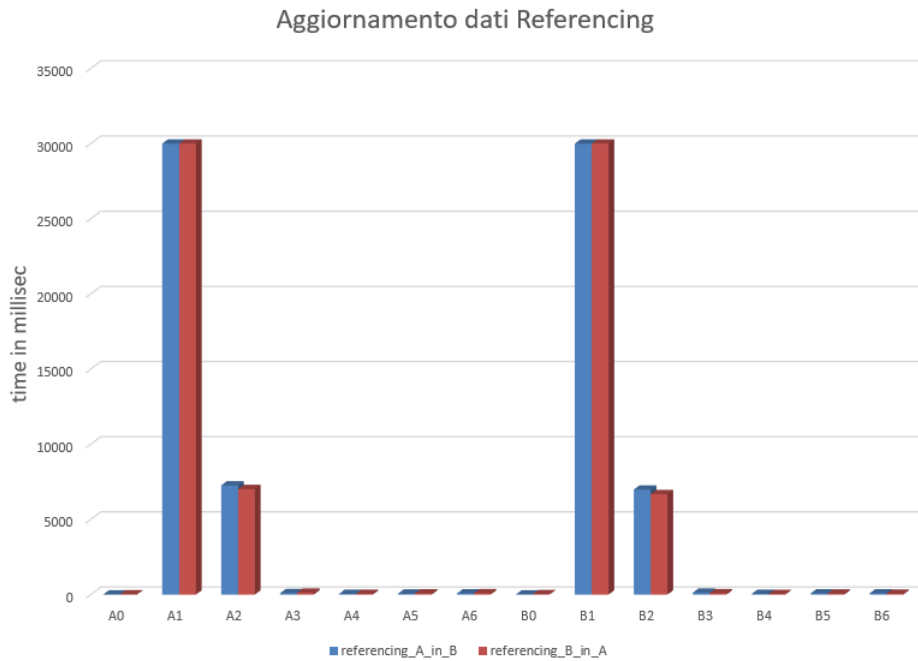


Figura 3.12: Grafico di aggiornamento per il referencing di CouchDB

All'interno del modello di Referencing, figura 3.12, non ci sono particolari problemi i tempi per l'aggiornamento dei vari attributi.

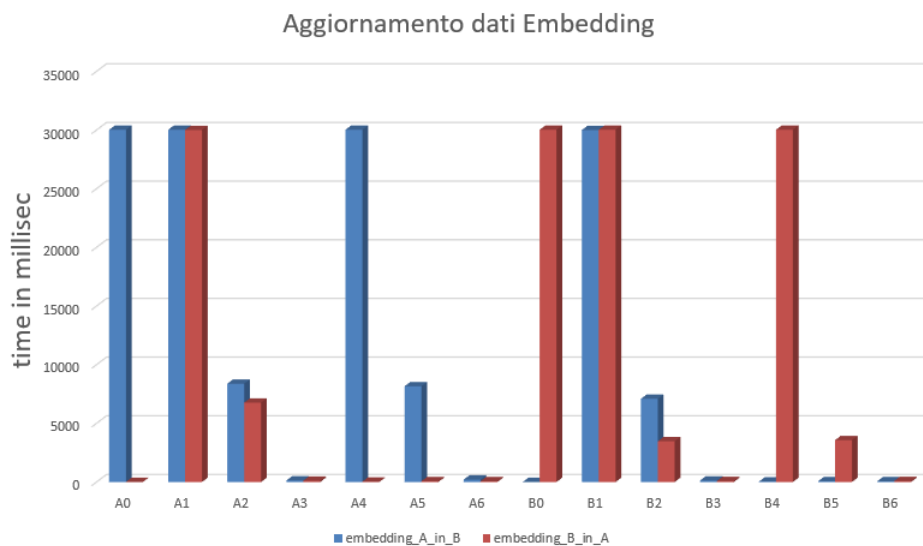


Figura 3.13: Grafico di aggiornamento per l' embedding di CouchDB

All'interno dell'embedding, figura 3.13, si nota invece quanto sia maggiormente oneroso modificare i documenti innestati all'interno, le prestazioni sono calate molto.

In figura 3.14 e tabella 3.6 sono visibili i tempi per tutti gli attributi di ogni documento.

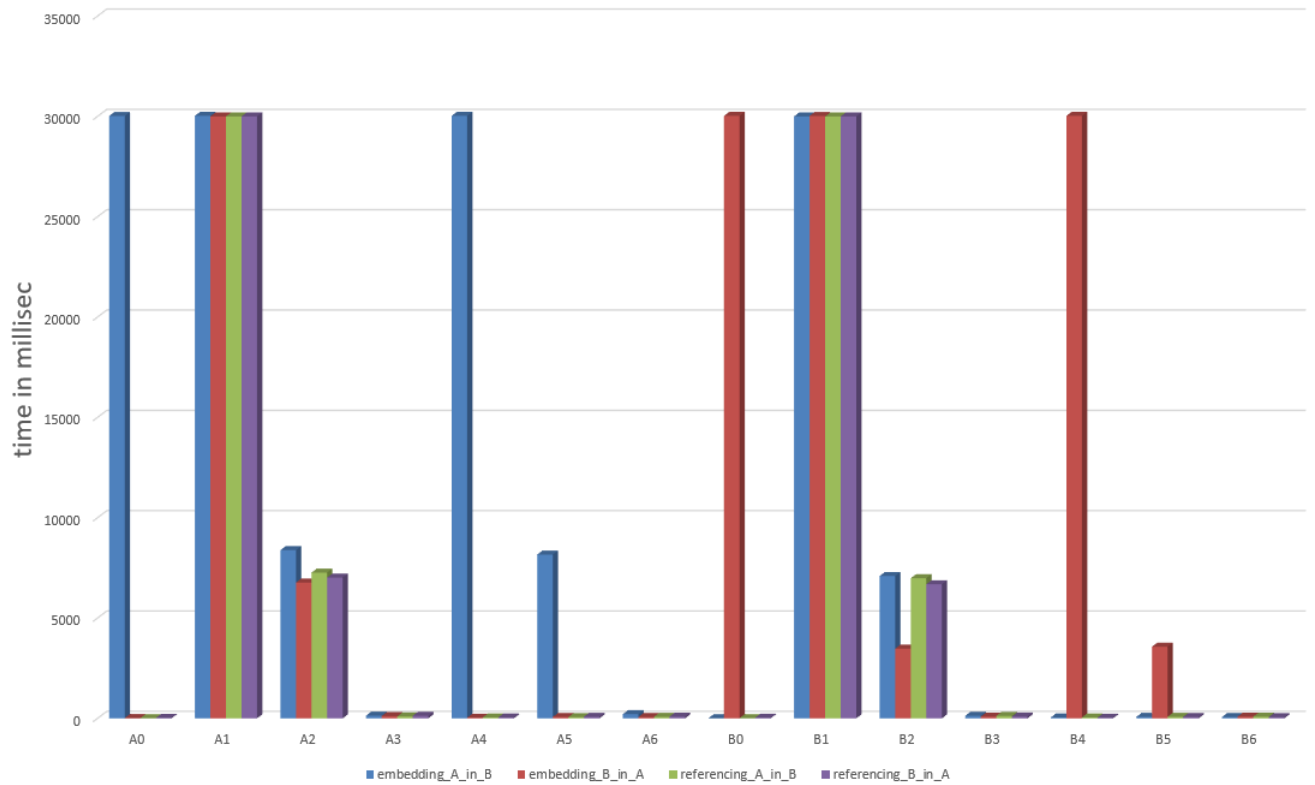


Figura 3.14: Grafico di aggiornamento per tutti gli attributi di CouchDB

Tabella 3.6: Tabella tempi aggiornamento di CouchDB

	embedding_A_in_B	embedding_B_in_A	referencing_A_in_B	referencing_B_in_A
A0	288001.2654	15.02020019	9.185131186	22.23928607
A1	68126.33102	24387.29044	26792.33555	30943.41705
A2	8382.086531	6768.962004	7265.057928	7015.500121
A3	135.2951096	113.4025369	94.85299168	125.0060012
A4	67001.68299	33.71748693	46.7626903	47.87131161
A5	8161.763236	71.12023961	63.09691267	77.43522967
A6	208.5213161	66.71289462	74.45368879	85.01299841
B0	7.280300154	158543.749	8.056679174	27.35192117
B1	253834.5184	126815.4719	264703.8242	285409.0585
B2	7094.00501	3472.816449	6983.299289	6681.451909
B3	127.9478412	86.30058583	123.0220173	88.43537725
B4	46.9661386	123617.562	46.20800574	33.00118351
B5	73.53258794	3571.528305	77.68148575	64.48137742
B6	67.80006986	86.06881668	80.34117896	58.65823389

3.3 Oracle



Figura 3.15: Oracle logo

3.3.1 Introduzione

Oracle [5] è una società che mette a disposizione numerosi prodotti, per la parte di confronto relazionale all'interno di questa tesi è stato usato oracle SQL developer, **figura 13**, un prodotto opensource che mette a disposizione un software per poter lavorare con SQL su database di Oracle attraverso il JDK (java development kit).

3.3.2 Server Locale

Oracle è un prodotto venduto soprattutto a grandi aziende, per utilizzarlo è necessario effettuare un'iscrizione e scaricare il loro software, la cosa più importante è riuscire a scaricare un server che venga correttamente eseguito in background sul proprio terminale.

Attraverso un software di oracle il processo viene completamente automatizzato creando una directory nella quale verranno creati i file di configurazione del server, una volta finita l'installazione dovrà essere presente tra i servizi un servizio particolare che si chiama OracleServicexxxx, in cui la parte finale definisce il nome della base di dati, automaticamente è ORCL ma può essere modificata, figura 3.16;

Nome	Descrizione	Stato	Tipo di avvio	Connessione
Modalità incorporata	Il servizio M...		Manuale (avv...	Sistema locale
Moduli di impostazione chiavi IPsec IKE e A...	Il servizio IK...		Manuale (avv...	Sistema locale
MongoDB Server (MongoDB)	MongoDB ...	In esecuzione	Automatico	Servizio di rete
Monitor del server dei fotogrammi di fotoca...	Monitora l'i...		Manuale (avv...	Sistema locale
Mozilla Maintenance Service	Mozilla Mai...		Manuale	Sistema locale
NPSMSvc_6cfed	<Impossibil...	In esecuzione	Manuale	Sistema locale
OneDrive Updater Service	Keeps your ...		Manuale (avv...	Sistema locale
OneSyncSvc_6cfed	Questo serv...	In esecuzione	Automatico (...)	Sistema locale
OpenSSH Authentication Agent	Agent to ho...		Disabilitato	Sistema locale
OpenSSH SSH Server	SSH protoc...		Manuale	Sistema locale
Ora cellulare	Questo serv...		Manuale (avv...	Servizio locale
Ora di Windows	Mantiene la...		Manuale (avv...	Servizio locale
OracleJobSchedulerORCL			Disabilitato	NT SERVICE\...
OracleJobSchedulerTIROCINIO			Disabilitato	NT SERVICE\...
OracleOraDB19Home1MTSRecoveryService			Automatico	NT SERVICE\...
OracleOraDB19Home1TNSListener			Automatico	NT SERVICE\...
OracleOraDB19Home2MTSRecoveryService		In esecuzione	Automatico	NT SERVICE\...
OracleOraDB19Home2TNSListener		In esecuzione	Automatico	NT SERVICE\...
OracleRemExecServiceV2			Manuale	Sistema locale
OracleServiceORCL		In esecuzione	Automatico	NT SERVICE\...
OracleServiceTIROCINIO			Automatico	NT SERVICE\...
OracleVssWriterORCL			Automatico	NT SERVICE\...
OracleVssWriterTIROCINIO			Automatico	NT SERVICE\...
Origin Client Service			Manuale	Sistema locale
Origin Web Helper Service			Automatico	Servizio locale
Ottimizza unità	Consente al...		Manuale	Sistema locale
Ottimizzazione recapito	Esegue attiv...		Automatico (...)	Servizio di rete
P9RdrService_6cfed	Abilita l'atti...		Manuale (avv...	Sistema locale
PenService_6cfed	Servizio pen...		Manuale (avv...	Sistema locale
PimIndexMaintenanceSvc_6cfed	Indicizza i d...	In esecuzione	Manuale	Sistema locale
Plug and Play	Consente al...	In esecuzione	Manuale	Sistema locale
PnkBstrA	PunkBuster ...	In esecuzione	Automatico	Sistema locale
Preparazione app	Consente di...		Manuale	Sistema locale
PrintWorkflowUserSvc_6cfed	Fornisce su...		Manuale (avv...	Sistema locale
Processo di installazione dei moduli di MG...	Caricata l'i...		Manuale	Sistema locale

Figura 3.16: Services di windows, oracle ORCL deve essere attivo

Al quale sarà possibile connettersi tramite l'applicativo sqldeveloper con la seguente configurazione di connessione in figura 3.17

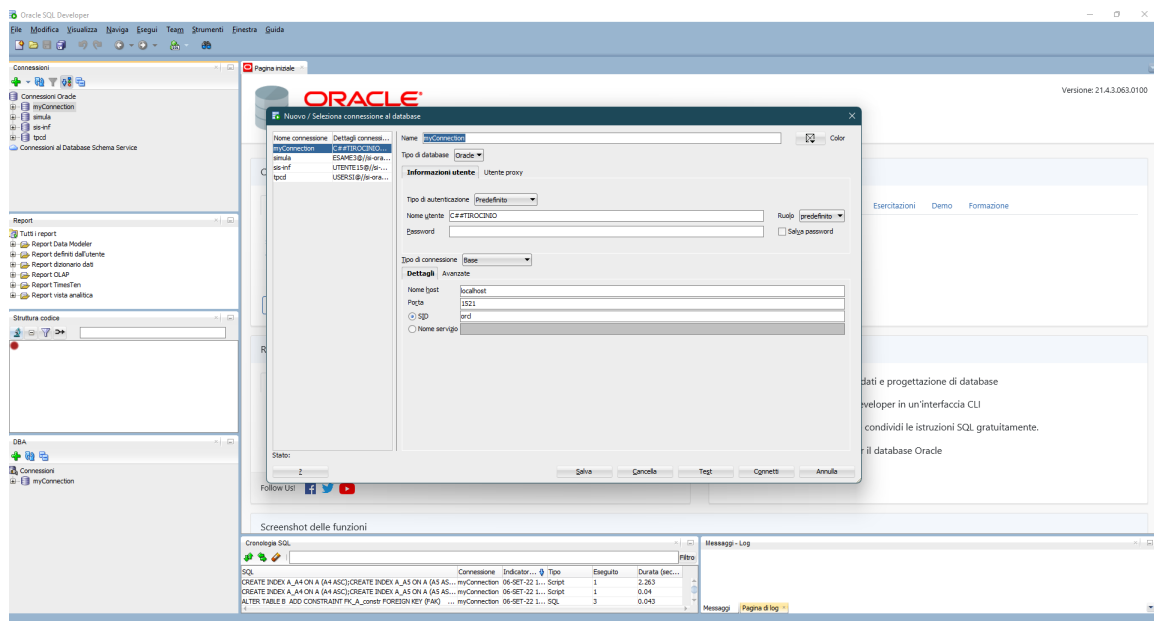


Figura 3.17: Pannello di connessione di sql developer per poter eseguire query sul nostro server

3.3.3 Organizzazione dei dati

Utilizzando un sistema relazionale ci si è dovuti limitare alla creazione di due collezioni distinte A (Post) e B(Commenti) sulle quali eseguire le query per poter ottenere una stima

dei costi su un modello relazionale che non utilizzasse embedding e referencing.

Sono stati creati file .csv dei dati facilmente importabili in un database Oracle, sui quali sono state effettuate alcune operazioni come la creazione di indici per specifici attributi e la limitazione della memoria cache del database locale.

Lo script di setup DB è riportato in appendice A.7

3.3.4 Inserimento (CRUD: Creation)

Oracle è un database relazionale, sul quale risulta semplice inserire una tupla all'interno di una tabella attraverso PL/SQL ma nel caso di una quantità cospicua di dati non è più possibile utilizzare quel metodo che funziona nei piccoli numeri. Il software non riesce a completare quei task così ci si è appoggiati alla funzionalità di importazione da file, in particolare si è adattato il dataset ad essere un file di tipo .csv che è stato facilmente inserito all'interno dell'istanza del database producendo le seguenti tempistiche in figura 3.18.

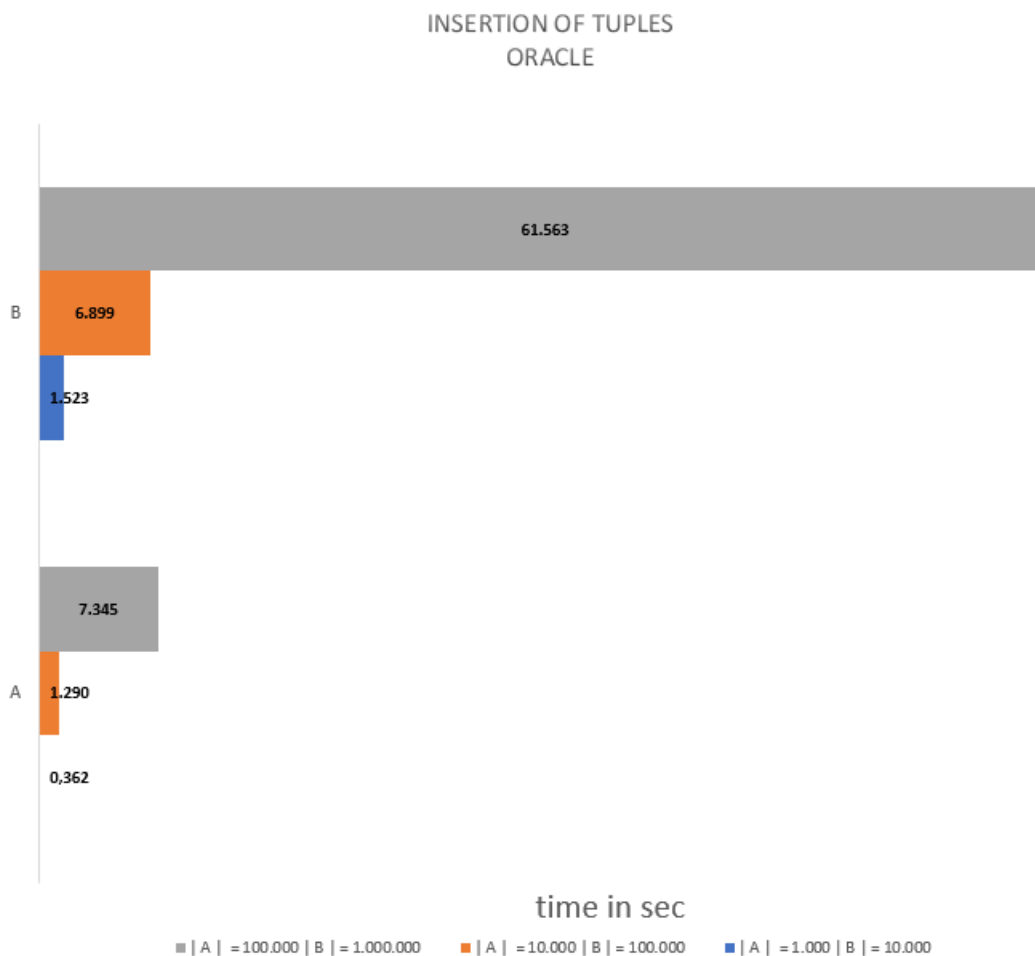


Figura 3.18: Statistiche inserimento dati su Oracle

3.3.5 Selezione (CRUD: Read)

Per poter eseguire le query di selezione è stato utilizzato uno script in python che utilizza il modulo `cx_Oracle` [6] per poter creare una connessione al database locale ed effettuare un numero arbitrario di query, ogni query effettuata riporta un piano di esecuzione visionabile nell'apposito file `.sql`. Dopo che la connessione viene confermata, si procede a creare un cursore con il quale si andranno ad eseguire query articolate come segue:

```
EXPLAIN PLAN FOR SELECT * FROM A WHERE A4 = 612;  
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Che provvedono a scrivere una riga all'interno del `PLAN_TABLE_OUTPUT` che poi verrà letta e salvata in un file, per poter avere una copia del piano di esecuzione di ciascuna transazione.

	A	B
A0	15	
A1	47	
A2	17	
A3	6	
A4	4	
A5	1	
A6	1	
B0		7
B1		295
B2		10
B3		5
B4		3
B5		5
B6		1
A0j	5	
A1j	15	
A2j	3	
A3j	4	
A4j	3	
A5j	2	
A6j	3	
B0j		4
B1j		524
B2j		10
B3j		937
B4j		2
B5j		5
B6j		797

3.3.6 Aggiornamento (CRUD: Update)

Le query di aggiornamento dei dati hanno prodotto i seguenti risultati

	A B
A0	2
A1	30
A2	23
A3	218
A4	1
A5	3
A6	220
B0	5
B1	3673
B2	660
B3	2079
B4	9
B5	16
B6	2864

Capitolo 4

Analisi dei risultati

4.1 Risultati modellazione

Per la manipolazione dei dati sono stati utilizzati fogli di calcolo excel, in particolare grazie al modulo `xlsxwriter` [7], ogni script dopo aver eseguito le query ha potuto riportare i risultati in file excel con colonne e righe preimpostate, con i tempi di esecuzioni su collezioni e attributi.

Lo script per creare file excel è riportato in appendice A.8

4.1.1 Operazione di Inserimento

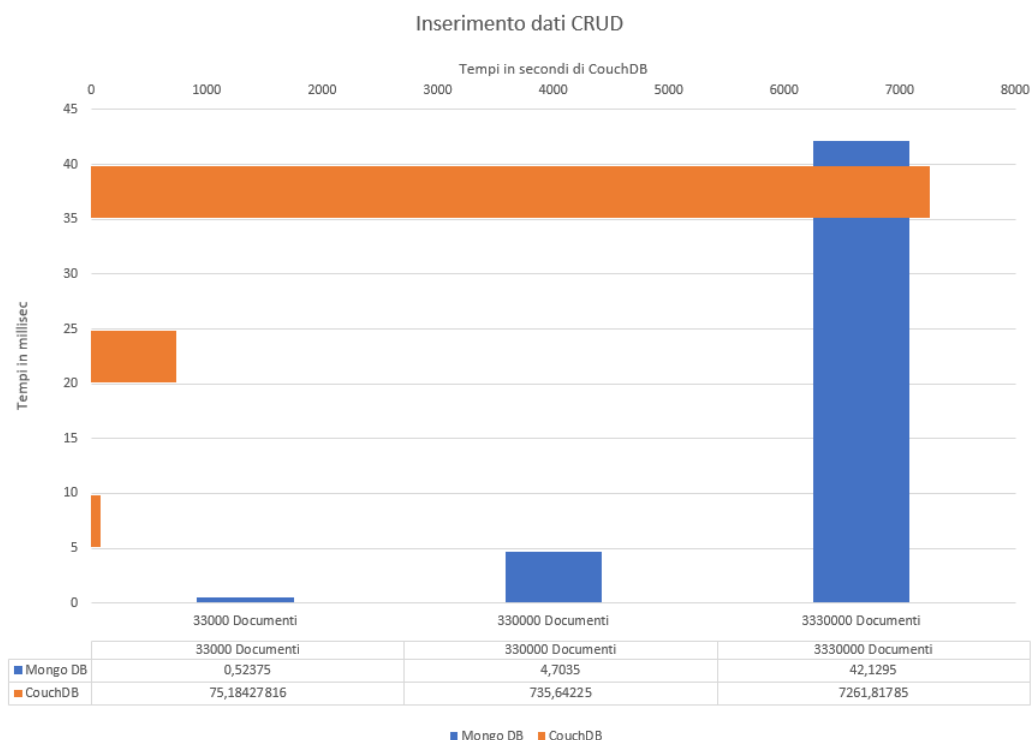


Figura 4.1: Grafico di inserimento per i valori dei database non relazionali

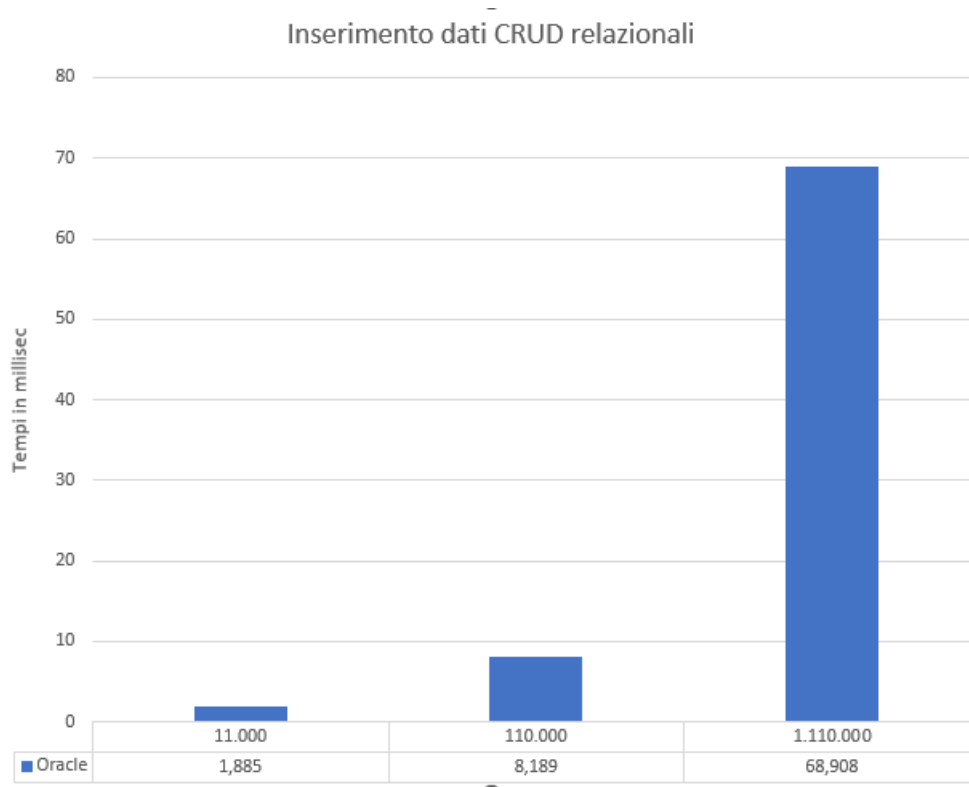


Figura 4.2: Grafico di inserimento per i valori dei database relazionali

I tempi di inserimento mettono subito in risalto quanto sia oneroso per CouchDB utilizzare il protocollo HTTP per creare un documento che a parità di dimensioni risulta molto più veloce nelle soluzioni relational e in MongoDB, i quali a loro volta presentano ulteriori differenze di tempistiche. Il miglior modo per poter popolare i database è quello di utilizzare l'applicativo mongoimport, sia su grandi quantità di dati che sulle più piccole mentre l'importazione di CSV ha risultati soddisfacenti ma non efficienti come nei database non relazionali che non utilizzano HTTP-REST.

Tabella 4.1: Tabella comparazione dimensione e tempo inserimento documenti in MongoDB

	emb_A_in_B	emb_B_in_A	ref_A_in_B	ref_B_in_A	dimensione
size	21 MB	12 M	12,1 MB	12,1 MB	33.000
					documenti
time	0,669	0,336	0,571	0,519	
size	208 MB	114 MB	116 MB	119 MB	330.000
					documenti
time	6,694	2,714	4,467	4,939	
size	2,01 GB	1.1 GB	1,11 GB	1,11 GB	3.330.000
					documenti
time	71,597	26,351	60,367	10,203	

4.1.2 Confronto con calcolo dei costi

La stima dei costi di accesso su MongoDB è più complessa rispetto a un database relazionale perché dovrebbe tener conto delle problematiche di gestione della cache e della replica dei dati. Per semplificare le stime e utilizzare formule per il calcolo dei costi proprie del mondo relazionale, abbiamo utilizzato alcune semplificazioni che riguardano essenzialmente l'uso di un solo nodo e la riduzione della memoria cache a disposizione del sistema. In questo modo è stato possibile applicare con un buon grado di approssimazione le formule di stima dei costi proprie dei database relazionali.

Si è partiti prendendo le dimensioni delle collezioni di dati e arrivando a trovare una lunghezza media per ogni collezione con la quale si è trovato il numero di pagine di disco occupate da ciascun modello.

Da qui è stato calcolato il costo di accesso ai dati per la selezione tramite gli attributi indicizzati e quelli non indicizzati, in particolare, dato l'uso di indici B-Tree è stata utilizzata la formula per indici Unclustered, appendice ??, mentre per gli attributi senza alcun indice costruito sopra la scansione sequenziale che legge la tabella, è stato usato un fattore alfa per ridurre i costi di lettura dato che non deve essere letta necessariamente tutta la collezione.

Per le selezioni che utilizzano il predicato di join, si è ipotizzato che MongoDB utilizzi l'algoritmo Nested Loop per l'esecuzione del join, le cui stime dei costi sono riportate in appendice B.5 e B.7.

In figura 4.3 sono mostrati i risultati ottenuti su MongoDB sommati per ogni modellazione ad un normale carico di lavoro comparati alle stime ottenute dalle formule.

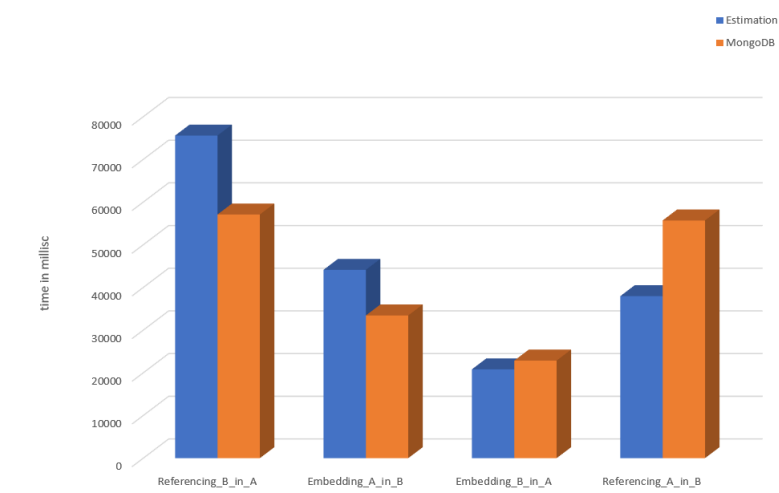


Figura 4.3: Comparazione dati per ogni collezione di MongoDB rispetto i costi stimati

4.1.3 Operazione di Selezione

Le operazioni di selezione si sono rivelate incredibilmente onerose con CouchDB mentre MongoDB è stato in grado di rimanere in linea con le stime effettuate ed in alcuni casi

impiega anche un tempo minore, come già citato non è possibile sapere quale piano d'accesso verrà utilizzato da MongoDB.

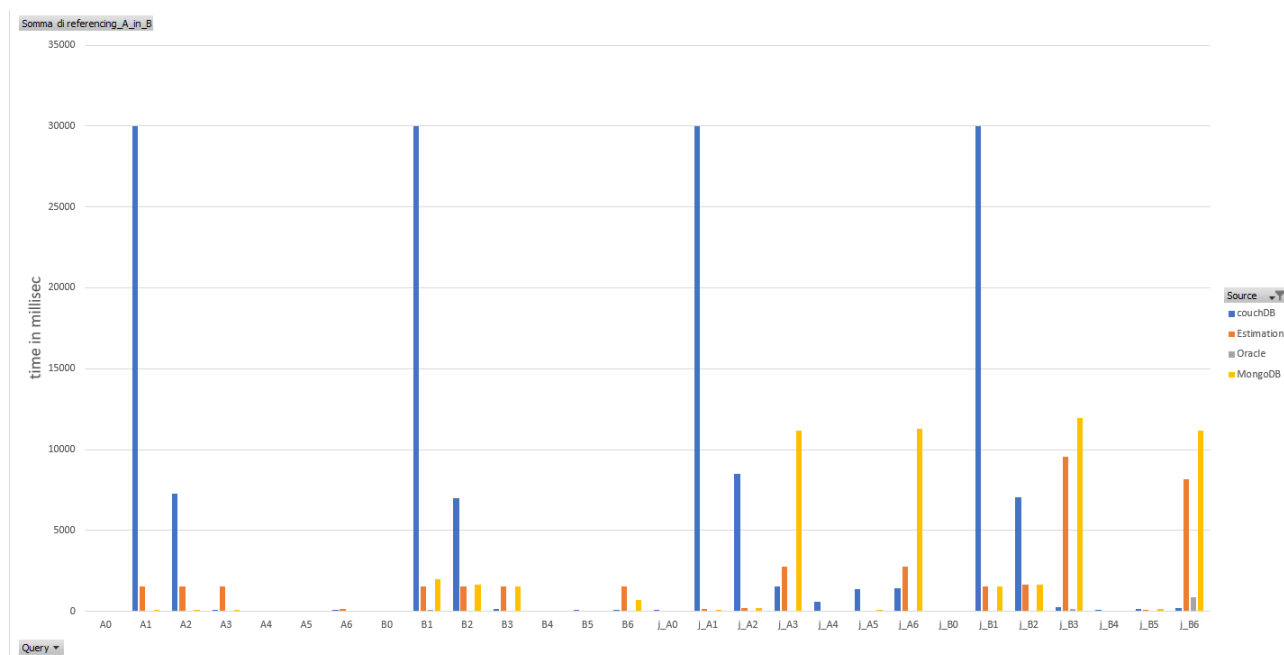


Figura 4.4: Confronto dati sulla collezione Referencing di A in B presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB

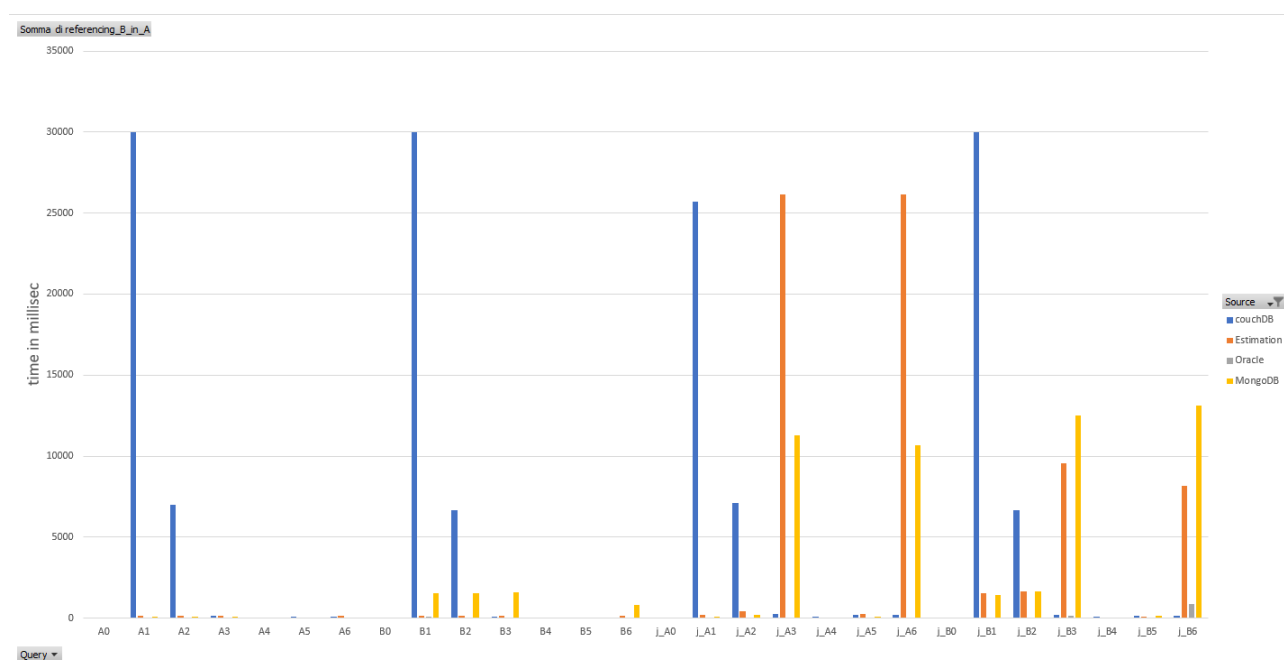


Figura 4.5: Confronto dati sulla collezione Referencing di B in A presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB

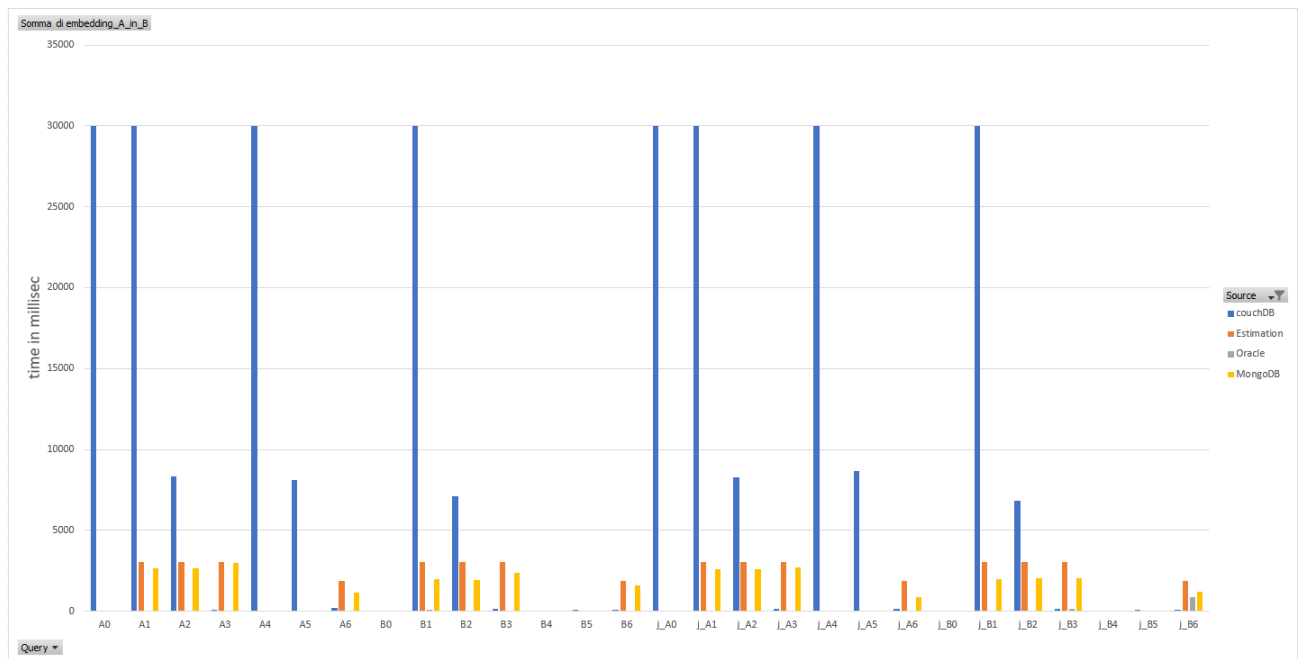


Figura 4.6: Confronto dati sulla collezione Embedding di A in B presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB

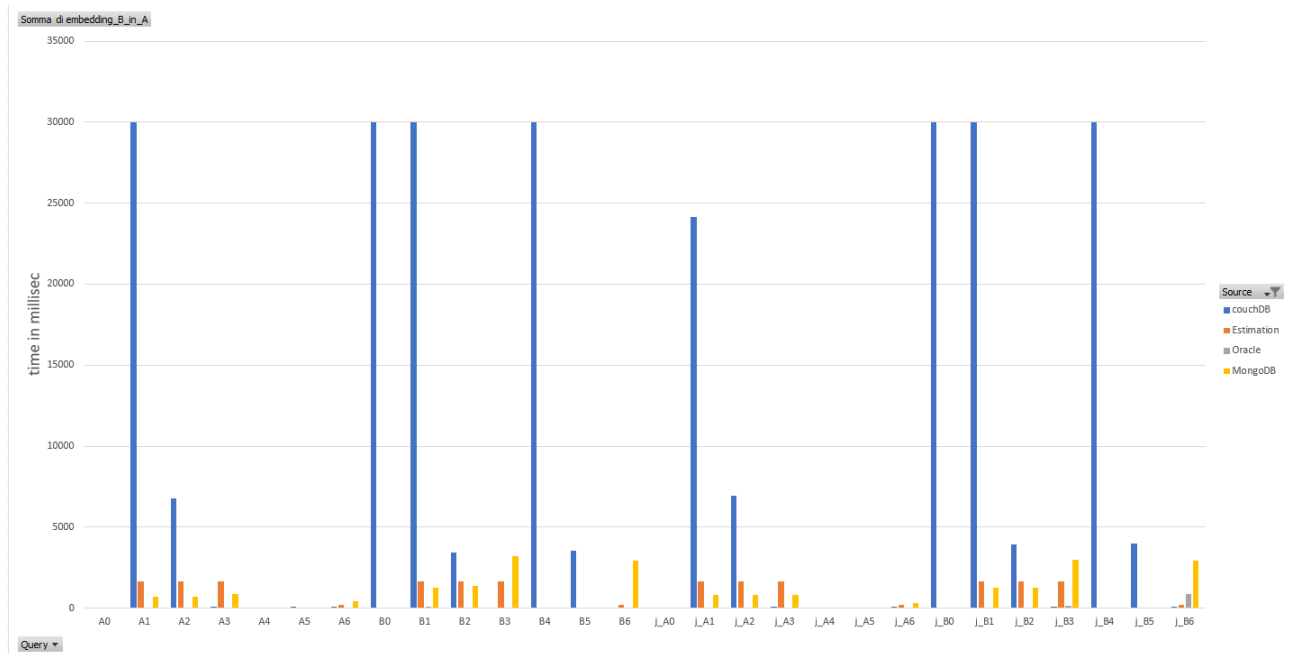


Figura 4.7: Confronto dati sulla collezione Embedding di B in A presi rispettivamente da CouchDB, stima dei costi, Oracle e MongoDB

4.2 Considerazioni finali

Le sperimentazioni e scelte effettuate all'interno di questo progetto di tesi sono state effettuate per poter verificare che alcune scelte, nell'ambito della creazione di basi di

dati non relazionali, siano effettivamente migliori le une rispetto alle altre ricercando dal generale al particolare.

I punti presi in analisi con le rispettive considerazioni seguono nei prossimi paragrafi....

4.2.1 Scelta del sistema di basi di dati non relazionale

La soluzione relazionale è sicuramente la più completa, in questa tesi ci siamo limitati ad osservare una relazione semplice mentre in situazioni reali spesso le cose non sono così facili e utilizzare modelli particolari può sicuramente incrementare le prestazioni ma anche la complessità del problema. Dopo aver considerato le statistiche rilevate nei risultati delle operazioni CRUD si può affermare senza ombra di dubbio che il miglior database non relazionale è MongoDB per molti motivi:

- non necessita di uno schema fisso da definire, è estremamente flessibile
- ogni query è stata facilmente eseguibile tramite comandi intuitivi e potenti (il comando aggregate)
- è molto efficiente, sia in fase di inserimento/update che in fase di ricerca

4.2.2 Scelta degli attributi da indicizzare tramite indice B-Tree

Il caso di studio presenta una piccola duplicazione dei dati per pura semplicità di esecuzione, il cui scopo è individuare la necessità di attributi indicizzati o meno.

Dopo aver raccolte le statistiche delle query di selezione è stato effettuato un lavoro sul carico di lavoro che ha prodotti i seguenti risultati nelle figure 4.8 4.9 4.10 in cui si comparano i grafici degli stessi attributi prima in versione normale e poi in versione indicizzata

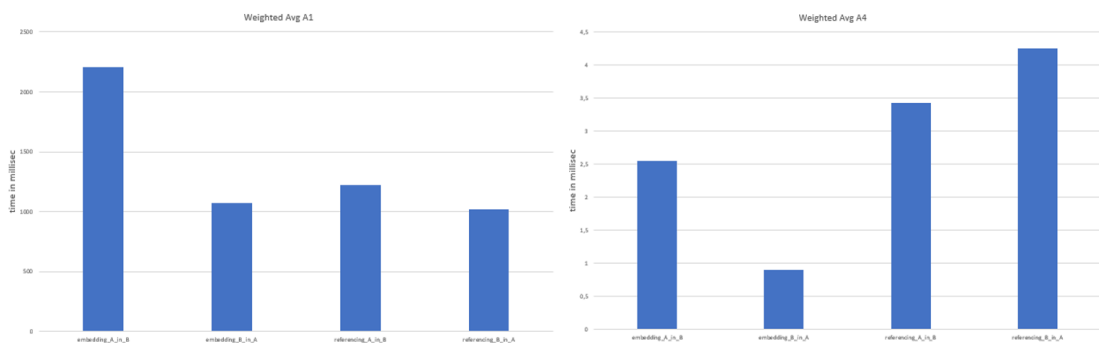


Figura 4.8: Confronto dati Attributo 1

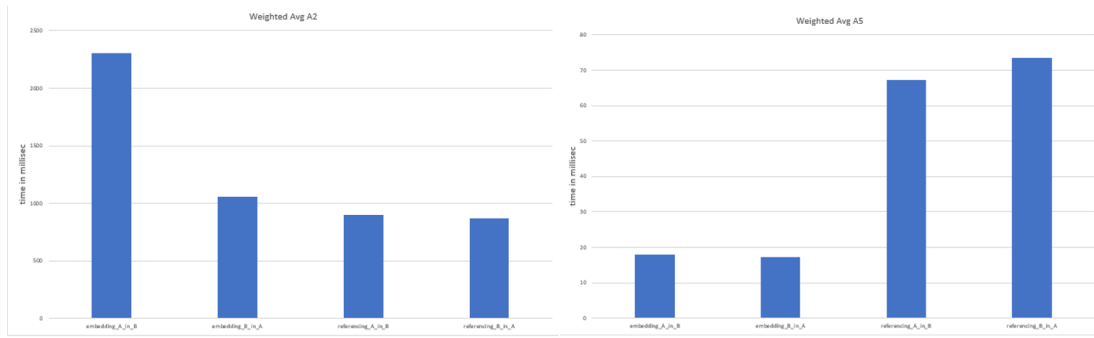


Figura 4.9: Confronto dati Attributo 2

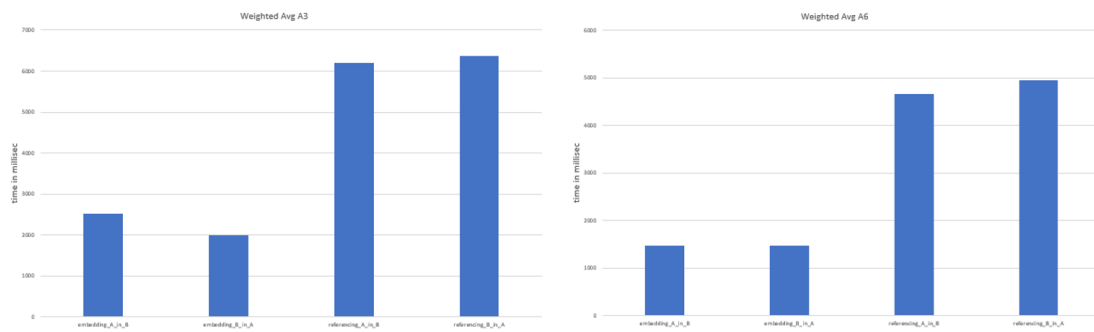


Figura 4.10: Confronto dati Attributo 3

La conclusione a cui si è giunti è che l'indicizzazione degli attributi 1 e 2 si rende necessaria nelle modellazioni di embedding, aumentando di molto le performance, in particolar modo nell'embedding di B in A che riesce ad essere maggiormente performante sia delle modellazioni di referencing che all'embedding di un singolo documento

Mentre l'attributo 3 ha dimostrato tempistiche pressoché simili, quindi si potrebbe valutare di utilizzare indicizzazione solo per i primi due attributi.

A riprova del fatto che in presenza di un fattore di selettività basso l'uso di un indice non è consigliabile si possono verificare le scelte dell'ottimizzatore di Oracle, che nelle query che coinvolgevano gli attributi A6/B6 (ossia la versione indicizzata di A3/B3), ha scelto di non utilizzare gli indici a favore della scansione sequenziale (FULL TABLE SCAN), quindi i risultati misurati hanno un riscontro con dati e scelte che vengono effettuati da un ottimizzatore commerciale.

Query di join

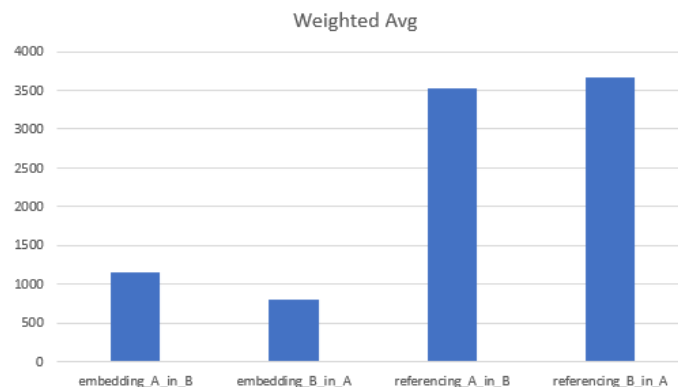


Figura 4.11: Confronto dati con carico di lavoro sulle query di join

Mentre spostando il focus sulle query di join si può notare che con un carico di lavoro incentrato maggiormente sulle query che utilizzano predicato di join 4.11 le soluzioni di embedding sono più vantaggiose del referencing. Assegnando un peso (frequenza) diverso alle query si può valutare la soglia oltre la quale, la quale la soluzione embedding è favorevole rispetto al referencing, figura 4.12 , il rapporto di questa soglia è all'incirca di 1 a 7, quindi se si prevede ad esempio di effettuare query di join con frequenza di maggiore di $1/7$ rispetto a selezioni su singola collezione, il modello più conveniente è quello di embedding mentre altrimenti è preferibile il referencing.

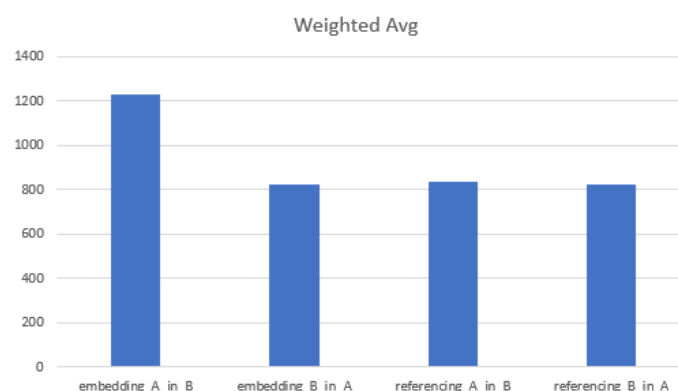


Figura 4.12: Confronto dati aumentando il carico di lavoro delle query no join

4.2.3 Scelta della modellazione documentale più performante

Per quanto riguarda le diverse strategie di modellazione dal grafico 4.13 le migliori modellazioni sono quelle di Embedding. Entrambe le soluzioni hanno il vantaggio di ridurre notevolmente i tempi di ricerca azzerando i costi di join e l'embedding di B in A riesce comunque a tenersi bassa anche nel caso di carichi di lavoro più intensi come a figura 4.12.

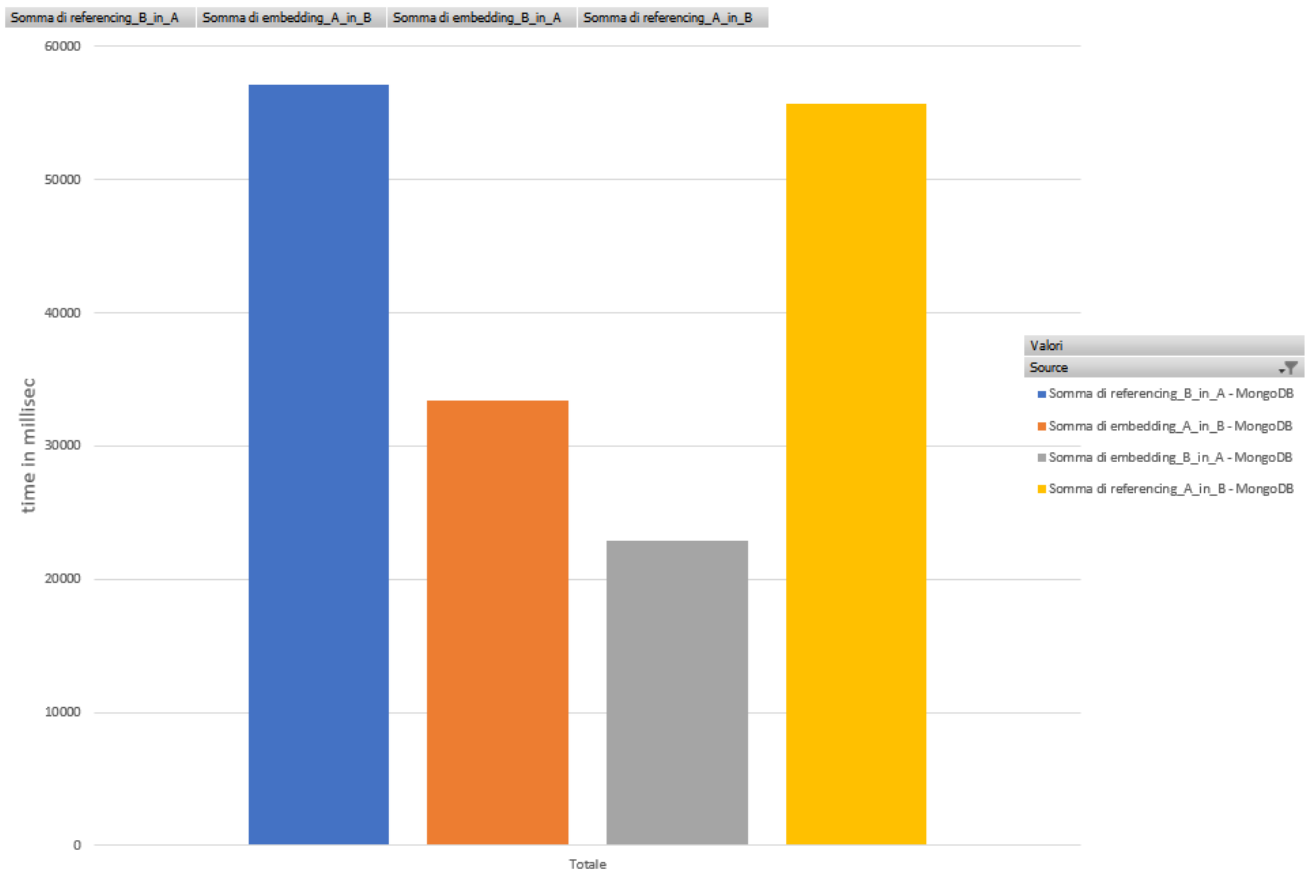


Figura 4.13: Confronto dati sulle collezioni in MongoDB

A parità di spazio occupato quindi sembrerebbe che i migliori metodi siano quello di effettuare l'embedding dell' entità che rappresenta il Post all'interno del Commento e l'utilizzo di sotto documenti di Commenti all'interno di un Post, si deve precisare che in questo dataset il rapporto tra post e commenti è di 1 a 10, quindi incapsulare un documento è performante se il numero di documenti è basso e non supera una certa soglia come citato sopra nella sezione 4.2.2

4.3 Problematiche riscontrate nell'elaborazione

I principali problemi riscontrati in questo progetto sono stati legati alle funzionalità di CouchDB

4.3.1 Assenza di join in CouchDb

Come precedentemente citato, CouchDB non consente l'utilizzo di predicati di join nella formulazione delle query, questo problema è stato risolto esplicitando la procedura che esegue il join lato applicazione: dopo aver ottenuto i risultati della prima selezione sulla relazione esterna sono stati salvati e attraverso un ciclo iterati per poter effettuare una

seconda ricerca sulla relazione interna tramite la corrispondenza della chiave comune alle due entità coinvolte all'interno dell'operazione.

4.3.2 Tempistiche CouchDB

Le statistiche legate ai tempi di esecuzione delle query si sono rivelati estremamente alte a causa di alcuni fattori:

- la scelta di mantenere il protocollo HTTP sembrerebbe essere legata alla sua modalità standard e universalmente conosciuta e per la sua facilità. MongoDB al contrario non presenta quasi nessun problema legato alla velocità di trasmissione, utilizza invece un protocollo TCP/IP
- CouchDB dopo aver ricevuto una richiesta HTTP, ad esempio una GET per l'inserimento di un nuovo documento non si limita semplicemente a trasferire uno stream di dati ma lo legge, utilizza un parser e poi serializza l'intero documento.

4.3.3 Indicizzazione CouchDB

CouchDB permette di costruire vari indici all'interno delle proprie collezioni, ma è risultato particolarmente difficile utilizzarne alcuni. Nel modello di dati embedding, ad esempio, il sotto-documento avrebbe dovuto avere indici diversi costruiti su ognuno dei propri attributi, ma nonostante ne fosse consentita la creazione, le query di selezione non erano in grado di utilizzarli effettuando solo delle scansioni sequenziali. La soluzione è stata quella di definire un solo indice che racchiudesse in se tutti gli attributi indicizzati.

Capitolo 5

Conclusioni

Gli obiettivi di questi tesi erano la modellazione logica e fisica di database non relazionali, ovvero l'analisi di diverse soluzioni di modellazione logica dei dati in database documentali e la scelta attributi sui quali costruire indici. Inoltre è stato operato il confronto di due DBMS documentali al fine di identificare quello più performante. Sono state utilizzati container all'interno dei quali è stato creato un ambiente privo di cache per evitare risultati che in parte dipendessero anche dal tipo di macchina utilizzato, è stato definito un caso di studio e un carico di lavoro sulle quali sono stati messi a confronto diversi DBMS e diverse soluzioni di modellazione logica e fisica. Il risultato di questo studio è che MongoDB, nonostante l'enorme numero di documenti inseriti, riesce a mantenere delle ottime performance rispetto a CouchDB. A causa delle API basate su protocollo HTTP, CouchDB è decisamente inefficiente trasportare, codificare e analizzare un enorme quantità di dati; al contrario MongoDB effettua una veloce conversione da BSON a JSON semplificando il processo. I tipi di modellazione testati hanno rivelato che con carichi di lavoro che fanno un uso minimo di query di join l'embedding ha tempi di selezione di molto inferiori al referencing, la differenza di tempi è stata colmata però al diminuire delle operazioni di join con le quali i tempi arrivano praticamente ad eguagliarsi. L'indicizzazione di attributi infine ha dimostrato che solo sugli attributi con selettività maggiore è utile la definizione di un indice, mentre se la selettività è dell'ordine di 1/10 la scansione sequenziale è il metodo d'accesso migliore: in questo senso i DBMS documentali si comportano allo stesso modo di quelli relazionali.

Concludendo con un numero minore di query di join le soluzioni di Referencing si sono dimostrate le più efficienti come si può vedere in figura 5.1.

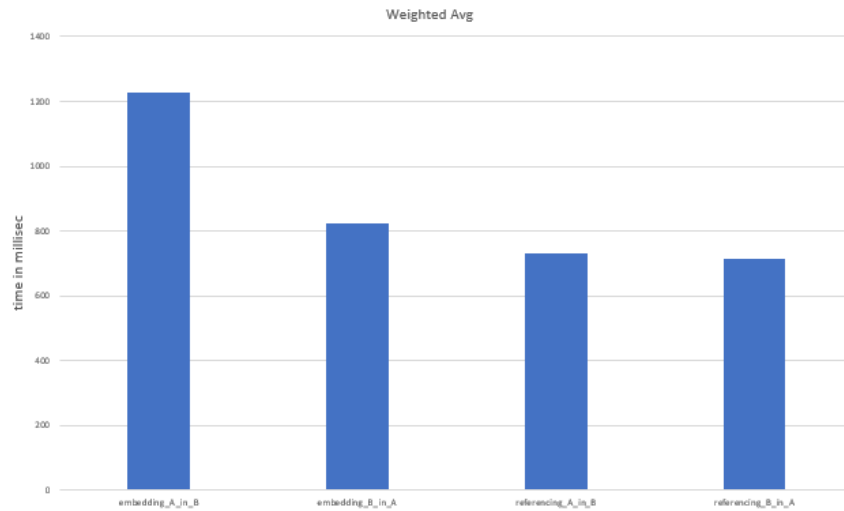


Figura 5.1: Grafico con carico di lavoro massimo su operazioni di selezioni senza join

Mentre con un carico di lavoro più proporzionato la modellazione migliore si è rivelata essere quella dell'embedding di B in A con il seguente schema riassuntivo.

Embedding di B in A conclusivo
<pre> { "_id": int, "AK": int, "A4": int, "A5": int, "A3": int, "A7": varchar(100), "B": [{"BK": int, "B4": int, "B5": int, "B3": int, "B7": varchar(100) }, . . {"BK": int, "B4": int, "B5": int, "B3": int, "B7": varchar(100) }] }</pre>
Embedding di B in A conclusivo

5.1 Sviluppi futuri

Il caso i studio considerato in questa tesi e il relativo carico di lavoro non esauriscono certamente le casistiche da considerare per definire una strategia di modellazione logica e fisica nei DBMS documentali. Ad esempio per la scelta degli indici è opportuno estendere il carico di lavoro considerato aggiungendo query che includono l'aggiornamento di attributi indicizzati, per poi vedere il comportamento degli indici al cambiamento degli attributi su cui sono costruiti. Altro limite del modello proposto riguarda il tipo di dati inseriti e di query effettuate: potrebbe essere interessante utilizzare dati che si avvicinano alla realtà e non solo attributi numerici, come ad esempio date per poter poi effettuare query più comuni, ma anche più complesse che utilizzino predicati di ordinamento o raggruppamento.

Anche le scelte effettuate in termini di progettazione logica non esauriscono il problema: oltre alle modellazioni testate in questa tesi, possono essere codificate tante soluzioni intermedie, come quelle proposte in [8], che includono modelli di documento ibridi che utilizzano sia l'embedding che il referencing. Questo tipo di documento potrebbe avere la base del modello referencing che al proprio interno contiene una piccola collezione di documenti embeddati secondo la logica del caso d'uso. Prendendo come esempio quello iniziale per questa tesi ossia il problema Post-Comments, la soluzione ibrida prevede di creare un documento per il Post che abbia un attributo Comments di tipo array, in modo da includere (embedding) un numero massimo di commenti (inseriti secondo una particolare logica di appartenenza come potrebbe ad esempio essere i commenti più recenti o i commenti che hanno un numero maggiore di likes o più risposte); tutti i commenti oltre al numero massimo dovrebbero essere linkati. Questa tipologia di soluzione ibrida, nota come Subset Pattern [8], incorporando solo i documenti che solitamente sono usati più frequentemente, ha il vantaggio di ridurre le dimensioni complessive del documento mantenendo al contempo un tempo di accesso più breve per la maggior parte dei dati utilizzati di frequente; mentre uno degli svantaggi è necessità di gestire i subset con operazioni di aggiornamento. Ovviamente l'uso di questa soluzione deve essere studiato in casi in cui la cardinalità della associazione tra post e commenti è piuttosto elevata (sicuramente maggiore del valore studiato in questa tesi).

Modello ibrido

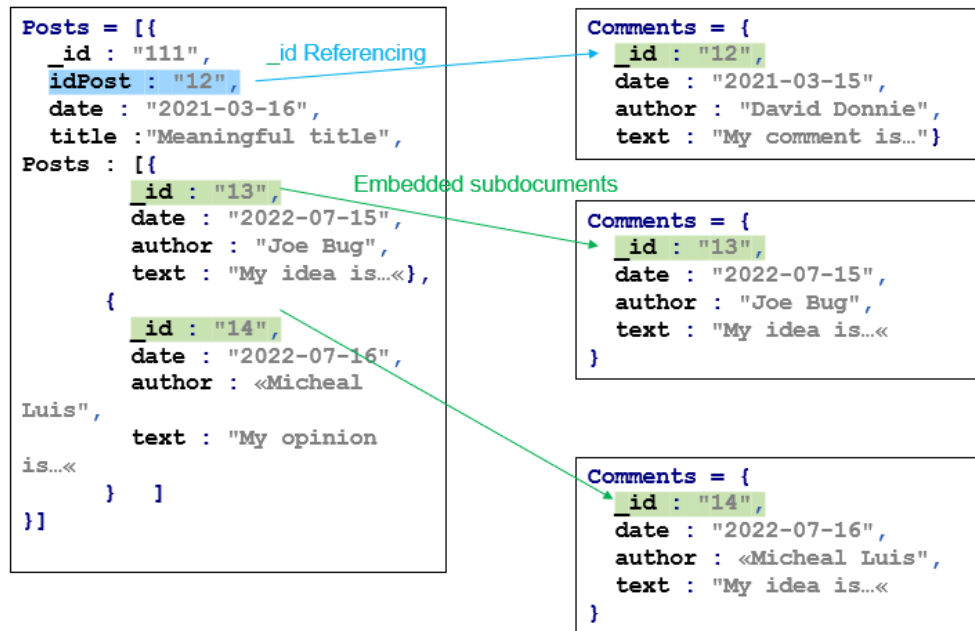


Figura 5.2: Schema di base per un eventuale modellazione ibrida

Minimo documento

Appendice A

Script

A.1 Script creazione Dataset

```
1  from ast import expr_context
2  import random
3  import json
4  from tqdm import tqdm
5
6  expA = 5
7
8  expB = 6
9
10 N_A = 10**expA
11
12 N_B = 10**expB
13
14 N = N_B / N_A
15
16 A = list(range(0, N_A))
17
18 B = list(range(0, N_B))
19
20 N_A1 = N_A/10
21
22 N_A2 = N_A/10**(round(expA/2))
23
24 N_A3 = N_A/10**(expA - 1)
25
26 N_B1 = N_B/10
```

```

27
28 N_B2 = N_B/10**(round(expB/2))
29
30 N_B3 = N_B/10**(expB - 1)
31
32 A1 = list(range(0, int(N_A1)))
33
34 A2 = list(range(0, int(N_A2)))
35
36 A3 = list(range(0, int(N_A3)))
37
38 B1 = list(range(0, int(N_B1)))
39
40 B2 = list(range(0, int(N_B2)))
41
42 B3 = list(range(0, int(N_B3)))
43
44 def populateList(rangeList):
45     return list(range(0, int(rangeList)))
46
47 def getAndDel(array):
48     if len(array) != 0:
49         element = random.choice(array)
50         ind = array.index(element)
51         return array.pop(ind)
52     else:
53         return "endOfList"
54
55
56 if __name__ == "__main__":
57     file_A = open("A.json", "w")
58     file_B = open("B.json", "w")
59     for i in tqdm(range(N_A)):
60         AK = getAndDel(A)
61         el_A1 = getAndDel(A1)
62         if el_A1 == "endOfList":
63             A1 = populateList(N_A1)
64             el_A1 = getAndDel(A1)
65         el_A2 = getAndDel(A2)

```

```

66         if el_A2 == "endOfList":
67             A2 = populateList(N_A2)
68             el_A2 = getAndDel(A2)
69         el_A3 = getAndDel(A3)
70         if el_A3 == "endOfList":
71             A3 = populateList(N_A3)
72             el_A3 = getAndDel(A3)
73         file_A.write(json.dumps({"AK" : AK, "A1" : el_A1, "A2" : el_A2,
74         "A3" : el_A3, "A4" : el_A1, "A5" : el_A2, "A6" : el_A3 ,
75         "A7" : "Lorem ipsum dolor sit amet....eget non justo quis." }) + "\n")
76         for y in range(N):
77             el_B1 = getAndDel(B1)
78             if el_B1 == "endOfList":
79                 B1 = populateList(N_B1)
80                 el_B1 = getAndDel(B1)
81             el_B2 = getAndDel(B2)
82             if el_B2 == "endOfList":
83                 B2 = populateList(N_B2)
84                 el_B2 = getAndDel(B2)
85             el_B3 = getAndDel(B3)
86             if el_B3 == "endOfList":
87                 B3 = populateList(N_B3)
88                 el_B3 = getAndDel(B3)
89             file_B.write(json.dumps({"BK" : getAndDel(B), "FAK" : AK, "B1" : el_B1, "
90             "B3" : el_B3, "B4" : el_B1, "B5" : el_B2, "B6" : el_B3 ,
91             "B7" : "Lorem ipsum dolor sit amet .... eget non justo quis." }) + "\n")

```

A.2 Script creazione immagine MongoDB

version: '3.8'

services:

mongodb:

image: mongo

container_name: mongodb

environment:

- MONGO_INITDB_ROOT_USERNAME=root
- MONGO_INITDB_ROOT_PASSWORD=pass12345
- MONGO_INITDB_DATABASE=test
- ME_CONFIG_BASICAUTH_USERNAME=admin

```

    - ME_CONFIG_BASICAUTH_PASSWORD=admin1234
volumes:
    - mongodb-data:/data/db
    - ../../dataJSON:/home/data
    - ./init-mongo.sh:/docker-entrypoint-initdb.d/init-mongo.sh
    - ./import.sh:/home/import.sh
    - ./result:/home/result
    - ./queries:/home/queries
networks:
    - mongodb_network
ports:
    - 27017:27017
healthcheck:
    test: echo 'db.runCommand("ping").ok' | mongo 127.0.0.1:27017/test -quiet
    interval: 30s
    timeout: 10s
    retries: 3
restart: unless-stopped
mem_limit: 1024m
cpu_count: 1
mongo-express:
    image: mongo-express:0.54.0 # 0.54.0 to see server status
    container_name: mongo-express
    environment:
        - ME_CONFIG_MONGODB_SERVER=mongodb
        - ME_CONFIG_MONGODB_ENABLE_ADMIN=true
        - ME_CONFIG_MONGODB_ADMINUSERNAME=root
        - ME_CONFIG_MONGODB_ADMINPASSWORD=pass12345
        - ME_CONFIG_BASICAUTH_USERNAME=admin
        - ME_CONFIG_BASICAUTH_PASSWORD=admin1234
    volumes:
        - mongodb-data
    depends_on:
        - mongodb
    networks:
        - mongodb_network
    ports:
        - 8081:8081
    healthcheck:

```

```

    test: wget -quiet -tries=3 -spider http://admin:admin1234@127.0.0.1:8081 || ex
    interval: 30s
    timeout: 10s
    retries: 3
    restart: unless-stopped
volumes:
  mongodb-data:
    name: mongodb-data
networks:
  mongodb_network:
    name: mongodb_network

```

A.3 Script creazione indici MongoDB

```
// indexes
```

```

db.getCollection('embedding_A_in_B').createIndex({'B4': 1});
db.getCollection('embedding_A_in_B').createIndex({'B5': 1});
db.getCollection('embedding_A_in_B').createIndex({'B6': 1});

```

```
// nested indexes
```

```

db.getCollection('embedding_A_in_B').createIndex({'A.AK': 1});
db.getCollection('embedding_A_in_B').createIndex({'A.A4': 1});
db.getCollection('embedding_A_in_B').createIndex({'A.A5': 1});
db.getCollection('embedding_A_in_B').createIndex({'A.A6': 1});

```

```
// indexes
```

```

db.getCollection('embedding_B_in_A').createIndex({'A4': 1});
db.getCollection('embedding_B_in_A').createIndex({'A5': 1});
db.getCollection('embedding_B_in_A').createIndex({'A6': 1});

```

```
// nested indexes
```

```

db.getCollection('embedding_B_in_A').createIndex({'B.BK': 1});
db.getCollection('embedding_B_in_A').createIndex({'B.B4': 1});
db.getCollection('embedding_B_in_A').createIndex({'B.B5': 1});
db.getCollection('embedding_B_in_A').createIndex({'B.B6': 1});

```

```
//FK
```

```

db.getCollection('referencing_A_in_B').createIndex({'AK': 1});

```

```

// indexes
db.getCollection('referencing_A_in_B').createIndex({'B4': 1});
db.getCollection('referencing_A_in_B').createIndex({'B5': 1});
db.getCollection('referencing_A_in_B').createIndex({'B6': 1});

//FK
db.getCollection('referencing_B_in_A').createIndex({'AK': 1});

// indexes
db.getCollection('referencing_B_in_A').createIndex({'A4': 1});
db.getCollection('referencing_B_in_A').createIndex({'A5': 1});
db.getCollection('referencing_B_in_A').createIndex({'A6': 1});

// array indexes
db.getCollection('embedding_A_in_B').createIndex({'A': 1});
db.getCollection('embedding_B_in_A').createIndex({'B': 1});

// A and B
db.getCollection('B').createIndex({'BK': 1});
db.getCollection('A').createIndex({'AK': 1});

db.getCollection('B').createIndex({'B4': 1});
db.getCollection('B').createIndex({'B5': 1});
db.getCollection('B').createIndex({'B6': 1});

db.getCollection('A').createIndex({'A4': 1});
db.getCollection('A').createIndex({'A5': 1});
db.getCollection('A').createIndex({'A6': 1});

```

A.4 Query update di MongoDB su python con pymongo

```

myclient = pymongo.MongoClient("mongodb://user:password@localhost:27017/")
mydb = myclient["tirocinio"]
res = mydb.command(
    'explain',
    {
        'update': 'embedding_B_in_A'
    }
)

```

```

        'updates' : [
            {'q' : {"A5": 56},
            'u' : {'$set' : {"A7": "modified document"
            }}}]
    },
    verbosity='executionStats'
)
if "stages" in list(res.keys()):
    time = (res["stages"][0]["$cursor"]["executionStats"]["executionTimeMillis"])
else:
    time = (res["executionStats"]["executionTimeMillis"])
# Updated failed
if res["executionStats"] == False:
    print(collection, ind)
    exit(0)
return time

```

A.5 Script creazione immagine CouchDB

```

version: '3'
services:
  couchserver:
    container_name: CouchDbServerMega
    image: couchdb:latest
    restart: always
    ports:
      - "5984:5984"
    mem_limit: 512m
    cpu_count: 4
    environment:
      - COUCHDB_USER=admin
      - COUCHDB_PASSWORD=admin
    volumes:
      - ./dbdata:/opt/couchdb/data

```

A.6 Script creazione indici per collezioni CouchDB

```

1
2 import requests

```



```
3 import json
4 import os
5
6 collections_a = [ 'referencing_b_in_a', 'embedding_b_in_a']
7 collections_b = ['embedding_a_in_b', 'referencing_a_in_b']
8
9 ind_a= ["A4", "A5", "A6"]
10 ind_b= ["B4", "B5", "B6"]
11
12
13 headers = {
14     'Content-Type': 'application/json'
15 }
16
17 # A
18 url = "http://admin:admin@127.0.0.1:5984/a/_index?partitioned=true"
19 for ind in ind_a:
20     payload = json.dumps({
21         "index": {
22             "fields": [
23                 ind
24             ]
25         },
26         "name": ind + "-index",
27         "type": "json"
28     })
29     response = requests.request("POST", url, headers=headers, data=payload)
30
31     payload = json.dumps({
32         "index": {
33             "fields": [
34                 "AK"
35             ]
36         },
37         "name": "AK" + "-index",
38         "type": "json"
39     })
40     response = requests.request("POST", url, headers=headers, data=payload)
41
```

```
42 # B
43 url = "http://admin:admin@127.0.0.1:5984/b/_index?partitioned=true"
44 for ind in ind_b:
45     payload = json.dumps({
46         "index": {
47             "fields": [
48                 ind
49             ]
50         },
51         "name": ind + "-index",
52         "type": "json"
53     })
54     response = requests.request("POST", url, headers=headers, data=payload)
55
56     payload = json.dumps({
57         "index": {
58             "fields": [
59                 "BK"
60             ]
61         },
62         "name": "BK" + "-index",
63         "type": "json"
64     })
65     response = requests.request("POST", url, headers=headers, data=payload)
66
67 for collection in collections_a:
68     url = "http://admin:admin@127.0.0.1:5984/" + collection + "/_index?partitioned=true"
69     for ind in ind_a:
70         payload = json.dumps({
71             "index": {
72                 "fields": [
73                     ind
74                 ]
75             },
76             "name": ind + "-index",
77             "type": "json"
78         })
79         response = requests.request("POST", url, headers=headers, data=payload)
80
```

```
81 for collection in collections_b:
82     url = "http://admin:admin@127.0.0.1:5984/" + collection + "/_index?partitioned=true"
83     for ind in ind_b:
84         payload = json.dumps({
85             "index": {
86                 "fields": [
87                     ind
88                 ]
89             },
90             "name": ind + "-index",
91             "type": "json"
92         })
93         response = requests.request("POST", url, headers=headers, data=payload)
94
95 url = "http://admin:admin@127.0.0.1:5984/referencing_a_in_b/_index?partitioned=true"
96 payload = json.dumps({
97     "index": {
98         "fields": [
99             "AK"
100     ]
101 },
102 "name": "AK" + "-index",
103 "type": "json"
104 })
105 response = requests.request("POST", url, headers=headers, data=payload)
106
107 # emb AB
108 url = "http://admin:admin@127.0.0.1:5984/embedding_a_in_b/_index?partitioned=true"
109 for ind in ind_a:
110     payload = json.dumps({
111         "index": {
112             "fields": [
113                 "A." + ind
114             ]
115         },
116         "name": "A." + ind + "-index",
117         "type": "json"
118     })
119     response = requests.request("POST", url, headers=headers, data=payload)
```

120

```
121 url = "http://admin:admin@127.0.0.1:5984/embedding_a_in_b/_index?partitioned=true"
```

```
122 for ind in ind_a:
```

```
123     payload = json.dumps({
```

```
124         "index": {
```

```
125             "fields": [
```

```
126                 "A.AK"
```

```
127             ]
```

```
128         },
```

```
129         "name": "A.AK-index",
```

```
130         "type": "json"
```

```
131     })
```

```
132     headers = {
```

```
133         'Content-Type': 'application/json'
```

```
134     }
```

```
135     response = requests.request("POST", url, headers=headers, data=payload)
```

```
136     print(response.text)
```

```
137 # emb BA
```

```
138 url = "http://admin:admin@127.0.0.1:5984/embedding_b_in_a/_index?partitioned=true"
```

```
139 for ind in ind_b:
```

```
140     payload = json.dumps({
```

```
141         "index": {
```

```
142             "fields": [
```

```
143                 "B." + ind
```

```
144             ]
```

```
145         },
```

```
146         "name": "B." + ind + "-index",
```

```
147         "type": "json"
```

```
148     })
```

```
149     response = requests.request("POST", url, headers=headers, data=payload)
```

150

```
151 url = "http://admin:admin@127.0.0.1:5984/embedding_b_in_a/_index?partitioned=true"
```

```
152 for ind in ind_b:
```

```
153     payload = json.dumps({
```

```
154         "index": {
```

```
155             "fields": [
```

```
156                 "B.BK"
```

```
157             ]
```

```
158         },
```

```
159     "name": "B.BK-index",
160     "type": "json"
161 })
162 headers = {
163     'Content-Type': 'application/json'
164 }
165 response = requests.request("POST", url, headers=headers, data=payload)
166
167 # ref BA
168 url = "http://admin:admin@127.0.0.1:5984/referencing_b_in_a/_index?partitioned=true"
169 payload = json.dumps({
170     "index": {
171         "fields": [
172             "B"
173         ]
174     },
175     "name": "B" + "-index",
176     "type": "json"
177 })
178 response = requests.request("POST", url, headers=headers, data=payload)
179
180 payload = json.dumps({
181     "index": {
182         "fields": [
183             "BK"
184         ]
185     },
186     "name": "BK" + "-index",
187     "type": "json"
188 })
189 headers = {
190     'Content-Type': 'application/json'
191 }
192 response = requests.request("POST", url, headers=headers, data=payload)
```

A.7 Script SQL per il setup del DB relazionale

- Creazione tabella A

```
CREATE TABLE "A"
( "AK" NUMBER(38,0) NOT NULL ENABLE,
  "A1" NUMBER(38,0),
  "A2" NUMBER(38,0),
  "A3" NUMBER(38,0),
  "A4" NUMBER(38,0),
  "A5" NUMBER(38,0),
  "A6" NUMBER(38,0),
  "A7" VARCHAR2(2048 BYTE),
  CONSTRAINT "PK" PRIMARY KEY ("AK"));
```

- Creazione tabella B

```
CREATE TABLE "B"
( "BK" NUMBER(38,0) NOT NULL ENABLE,
  "FAK" NUMBER(38,0),
  "B1" NUMBER(38,0),
  "B2" NUMBER(38,0),
  "B3" NUMBER(38,0),
  "B4" NUMBER(38,0),
  "B5" NUMBER(38,0),
  "B6" NUMBER(38,0),
  "B7" VARCHAR2(2048 BYTE),
  CONSTRAINT "PKB" PRIMARY KEY ("BK"));
```

- Primary key per A

```
alter table "A" add constraint PK primary key("AK")
```

- Primary key per B

```
alter table "B" add constraint PKB primary key("BK")
```

- Aggiungere il constraint per FAK

```
ALTER TABLE B
ADD CONSTRAINT FK_A_constr FOREIGN KEY (FAK)
  REFERENCES A(AK)
  ON DELETE CASCADE;
```

- Indici

```
CREATE UNIQUE INDEX A_AK ON A (AK ASC);
CREATE INDEX A_A4 ON A (A4 ASC);
CREATE INDEX A_A5 ON A (A5 ASC);
CREATE INDEX A_A6 ON A (A6 ASC);
CREATE UNIQUE INDEX B_BK ON B (BK ASC);
CREATE INDEX B_B4 ON B (B4 ASC);
CREATE INDEX B_B5 ON B (B5 ASC);
CREATE INDEX B_B6 ON B (B6 ASC);
CREATE INDEX B_FK ON B (FAK ASC);
```

- Ridurre la cache del client di sistema

```
ALTER SYSTEM SET CLIENT_RESULT_CACHE_SIZE = 128M SCOPE=SPFILE;
```

A.8 Script python con xlsxwriter

```
workbook = xlsxwriter.Workbook('OracleStat.xlsx')
worksheet = workbook.add_worksheet()
bold = workbook.add_format({'bold': True})
worksheet.write_string('B1', 'A', bold)
worksheet.write_string('C1', 'B', bold)
worksheet.write_string('A2', 'A0', bold)
worksheet.write_string('A3', 'A1', bold)
worksheet.write_string('A4', 'A2', bold)
worksheet.write_string('A5', 'A3', bold)
worksheet.write_string('A6', 'A4', bold)
worksheet.write_string('A7', 'A5', bold)
worksheet.write_string('A8', 'A6', bold)
worksheet.write_string('A9', 'B0', bold)
worksheet.write_string('A10', 'B1', bold)
worksheet.write_string('A11', 'B2', bold)
worksheet.write_string('A12', 'B3', bold)
worksheet.write_string('A13', 'B4', bold)
worksheet.write_string('A14', 'B5', bold)
```

```
worksheet.write_string('A15', 'B6', bold)
```

```
# inserire ad esempio un tempo in una cella particolare
```

```
worksheet.write(column, row , value)
```

```
# per completare e creare il file
```

```
workbook.close()
```


Appendice B

Calcolo dei costi

La parte teorica di questo progetto si basa sulla stima di costi di esecuzione delle operazioni sulle differenti opzioni di organizzazione dei documenti, per calcolare cio' abbiamo bisogno di alcune formule che andro qui ad elencare:

B.1 NL

$$NL = \left\lfloor \frac{NK * len(k) + NR * len(p)}{D * u} \right\rfloor$$

B.2 g

$$g = \left\lceil \frac{D - len(p)}{2 * (len(k) + len(p))} \right\rceil$$

B.3 h

Il calcolo per l'altezza del b-tree relativo agli indici degli attributi verrebbe normalmente calcolato attraverso questa formula, nel nostro caso di studio l'altezza e' stata approssimata a 3.

$$\left\lceil \log_{(2*g+1)} NL \right\rceil \leq h \leq \left\lceil \log_{(g+1)} \frac{NL}{2} \right\rceil$$

B.4 NP

Numero di pagine occupate sul disco da una tabella, nel caso non relazionale dalla collezione.

$$NP = \left\lfloor \frac{NR * len(t)}{D * u} \right\rfloor$$

B.5 Costo del nested loop

All'interno di un predicato di join in cui ho R relazione esterna e S relazione interna posso trovare il costo di questa operazione

- senza predicato di selezione

$$NP_R + NR_R * NP_S$$

- con predicato di selezione

$$NP_R + (sel(pred) * NR_R) * costo(S)$$

B.6 Indice clustered

$$Costo(S) = (h - 1) + \left\lfloor \frac{1}{NK} * NL \right\rfloor + \lfloor sel(pred) * NP \rfloor$$

B.7 Indice unclustered

$$Costo(S) = (h - 1) + \left\lfloor \frac{1}{NK} * NL \right\rfloor + 1 + \phi\left(\frac{NR}{NK}, NP\right)$$

Bibliografia

- [1] Wikipedia. «SQL Wikipedia». (), indirizzo: https://it.wikipedia.org/wiki/Structured_Query_Language.
- [2] I. MongoDB. «MongoDB la piattaforma di dati applicativi». (), indirizzo: <https://www.mongodb.com/it-it>.
- [3] I. MongoDB. «Web-based MongoDB admin interface written with Node.js, Express and Bootstrap3». (), indirizzo: <https://github.com/mongo-express/mongo-express>.
- [4] T. A. S. Foundation. «Couch DB apache». (), indirizzo: <https://couchdb.apache.org/>.
- [5] Oracle. «Oracle Italia | Applicazioni cloud e Cloud Platform». (), indirizzo: <https://www.oracle.com/it/index.html>.
- [6] A. Tuininga. «cx-Oracle». (), indirizzo: <https://pypi.org/project/cx-Oracle/>.
- [7] J. McNamara. «Creating Excel files with Python and XlsxWriter». (), indirizzo: <https://xlsxwriter.readthedocs.io/>.
- [8] A. Lumini. «MongoDB document-design». ().