Alma Mater Studiorum — Università di Bologna
Campus di Cesena

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea magistrale in Ingegneria e Scienze Informatiche

Stima dei costi database non relazionali

Calcolo e stima dei costi di elaborazione di diverse query su attributi indicizzati
Databse non relazionali in questione MongoDb e CouchDB

Presentata da:
Matteo Santoro
Matricola 0000881608

Relatore:
Prof. Alessandra Lumini

Correlatore:
Prof. Alessandra Lumini

III sessione di laurea
Anno Accademico 2022–2023

Parole chiave

Parola chiave 1

Parola chiave 2

Dedica

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, processare e conservare diversi tipi di dati, e' quindi fondamentale poter ottimizzare al meglio queste operazioni, all'interno della tesi che presentero' andro' ad analizzare e stimare alcuni possibili metodi e condurre un'analisi comparativa anche attraverso diversi sistemi di database. Io e la Prof.ssa Lumini abbiamo deciso di utlizzare due database Document-Based non relazionali, MongoDB e CouchDB ed un database di tipo relazionale, Oracle. La finalita' di questa tesi e' arrivare a valutare il miglior piano d'accesso possibile e la modellazione dei dati piu' efficente.

Indice

Introduzione

//TODO la e' non va bene, bisogna usare è

1 Differenze tra relational/non-relational DB

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse et ex vehicula, interdum mi eu, auctor augue. Suspendisse vel sagittis urna. In vitae ligula ipsum. Vivamus mattis neque efficitur, gravida purus facilisis, rhoncus felis. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nunc bibendum urna porta quam congue ornare. Fusce eget consequat libero. Donec varius justo vel libero malesuada, sit amet dignissim ex tincidunt. In pharetra vestibulum lacus quis laoreet. Donec vel laoreet ex, sed facilisis massa. Mauris commodo velit est. Maecenas non sem elementum, faucibus velit et, bibendum mi. Integer laoreet ex et eros accumsan, nec dapibus nisi vestibulum. Donec sed nunc quis mi gravida congue non vel tortor. Interdum et malesuada fames ac ante ipsum primis in faucibus.

Nullam non pharetra arcu. Donec eget velit elit. Pellentesque sed tortor sodales neque tristique rutrum. Vivamus et ex dolor. Vestibulum lacinia augue sit amet libero mattis pellentesque. In sed congue ante, quis tempus neque. Vestibulum semper eu sapien et vestibulum.

In ac erat ullamcorper, ultricies dolor sit amet, tempus neque. Pellentesque quam erat, ornare ac justo at, cursus luctus ex. Pellentesque at turpis blandit, elementum ex ac, fringilla nibh. Etiam et tincidunt lacus. Integer mattis mi sit amet faucibus rutrum. Sed at nisi commodo, ultricies purus a, tempus lacus. Mauris accumsan enim nisi, in tempor velit pharetra eu. Sed eu turpis et ante sagittis sodales. Duis a tellus id risus ultricies accumsan. Vestibulum bibendum in sapien sit amet rhoncus. Fusce aliquam, metus vel efficitur pulvinar, nibh lacus ultricies nisl, nec rhoncus elit ligula vitae risus. Mauris bibendum eget erat non rutrum. Curabitur in ligula eget lectus facilisis molestie sed sed neque. Vestibulum eu faucibus augue, a luctus augue. Nam lobortis massa non lorem

```
def hello_world():
    print("Hello floating world!")
```

Listing 1: Floating listing.

condimentum vehicula. Aliquam efficitur cursus neque, efficitur placerat libero tincidunt non. Nullam.

2 Il nostro caso di studio

Durante la progettazione si e' deciso di utilizzare una relazione per poter testare i diversi metodi di collezione dei dati. La relazione e' formata da 2 entita' A e B,in un esempio di un social network l'entita' A sono i Post e l'entita' B i Commenti sotto ogni post, collegate da una relazione 1..N -> 1...1, per ogni A esistono 10 B relativi, ogni Post contiene 10 Commenti.

La modellazione delle entita' e' stata fatta creando 2 triplette di attributi uguali, sulla seconda tripletta al contrario della prima sono stati costruiti degli indici su ogni attributo, questi attributi generici hanno una propria selettivita':

$$\left.\begin{aligned} sel(Att1, Att4) &= \frac{1}{10} \\ sel(Att2, Att5) &= \frac{1}{10.000} \\ sel(Att2, Att6) &= \frac{1}{100} \end{aligned}\right\}$$

Poi un attributo testaule, A7, di 100 bytes per poter inserire descrizione.

Il numero di Entita' A e' di 10^5 e di conseguenza ci sono 10^6 entita' B.

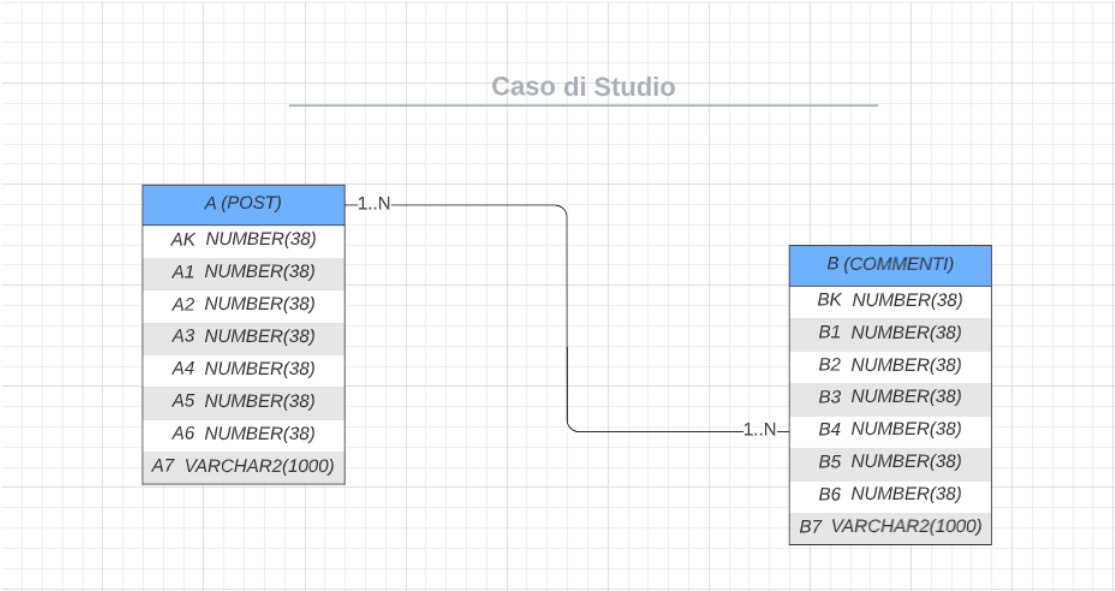
A (POST)


```
"_id": 6808,
"AK": 6808,
"A1": 3042,
"A2": 300,
"A3": 8,
"A4": 3042,
"A5": 300,
"A6": 8,
"A7": "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  Fusce lacinia eget arcu et maximus. Ut tempus est sit amet
  tortor commodo, sit amet facilisis mi rhoncus. Donec et elit
  venenatis, consequat tellus eu, tristique orci. Duis
  tristique sem ut nulla ullamcorper, a porta risus efficitur.
  Cras sed neque et nisl tincidunt vestibulum. Phasellus
  tristique tempor facilisis. Sed facilisis lectus eros, sed
  aliquet lacus elementum sed. Integer vel dictum mi.
  Maecenas pharetra tempus eros, efficitur mattis erat cursus
```

```
in. Nulla sit amet quam velit. Nullam tempus dictum lacus
id porttitor. Vestibulum facilisis pulvinar fermentum.
Ut elementum maximus feugiat. In at mollis leo, eu
facilisis magna. Vestibulum sed nisi ultricies, tincidunt
enim ac, fringilla ex. Phasellus pharetra mollis nisi
a fermentum. In nec faucibus nulla, eget molestie magna.
Vivamus in gravida ex. Aenean scelerisque gravida
ipsum, nec congue enim posuere sit amet. Donec vitae felis
id sem congue blandit eget non justo quis."
```

A (POST)

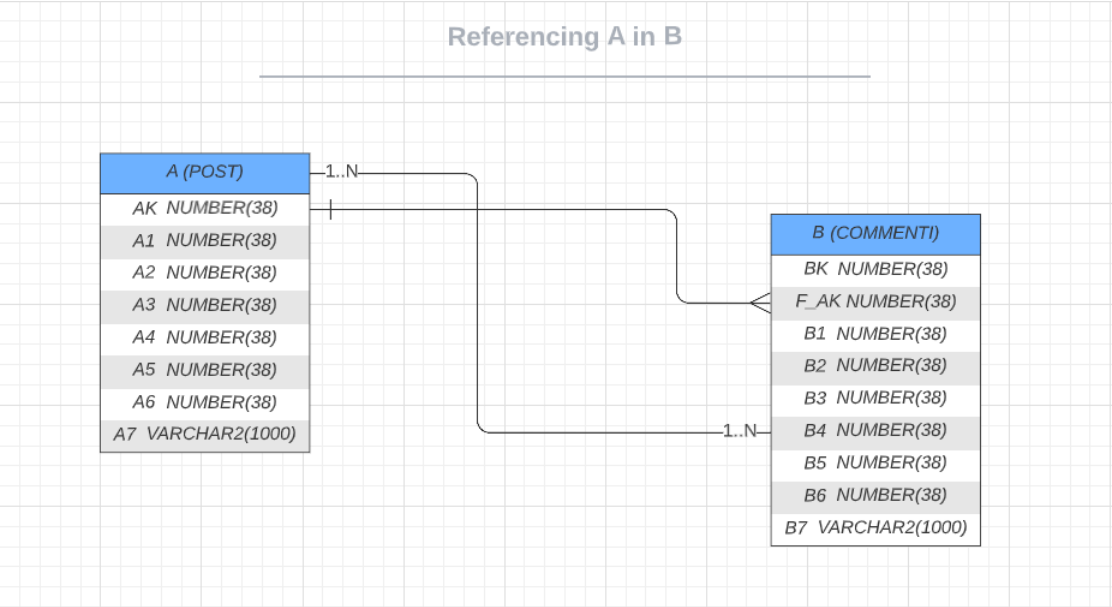
!!!!!!!!!! LE FOTO NON VANNO BENE, LE CHIAVI DEVONO ESSERE SEGNALATE !!!!!!!!!!!!!!!!!!!!!!!!!

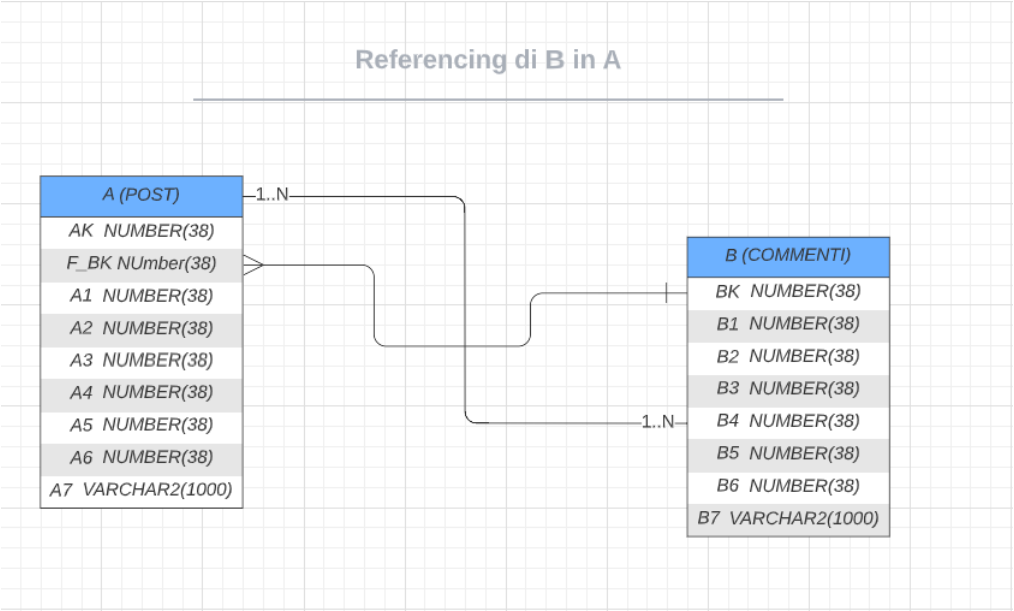


3 Modellazione dei dati

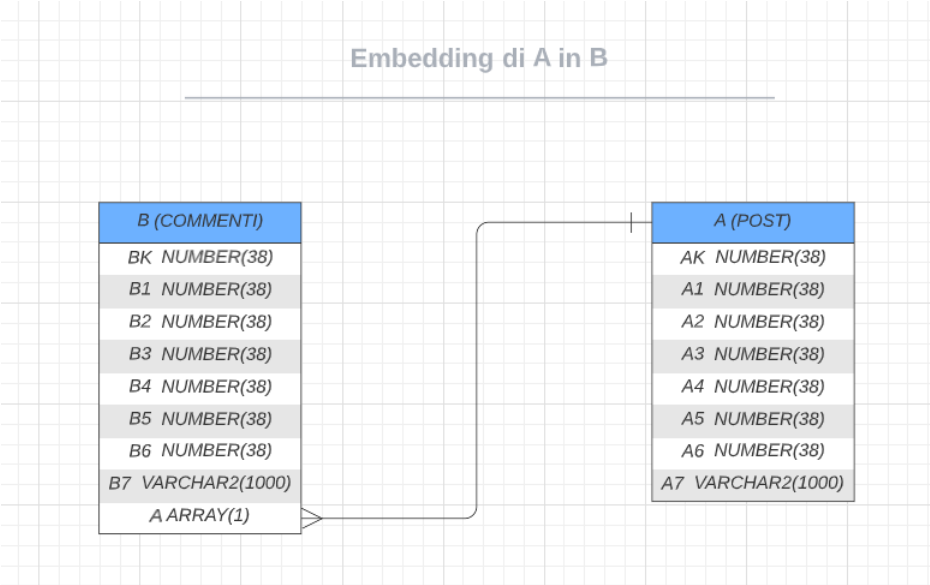
La modellazione dei dati ha un ruolo fondamentale nel calcolo dei costi di accesso e quindi costi di esecuzione delle query, le soluzioni dalle quali si e' deciso di di partire sono quelle dell'Embedding e del Referencing. Utilizzando un sistema non relazionale abbiamo necessita' di esprimere la relazione tra A e B in un modo differente, attraverso il referencing colleghiamo le un entita' inserendo un riferimento della seconda e viceversa, per poter poi eseguire ad esempio query di join o voler recuperare tutti i dati dell'entita' referenziata c'e' il bisogno di mantenere all'interno del database la collezione contenente i rimanenti attributi. Es. nell'eventuale soluzione del referencing di B all'interno di A, ci sono due collezioni ossia A con la foreign key relativa (BK) relativa a B e l'intera collezione B.

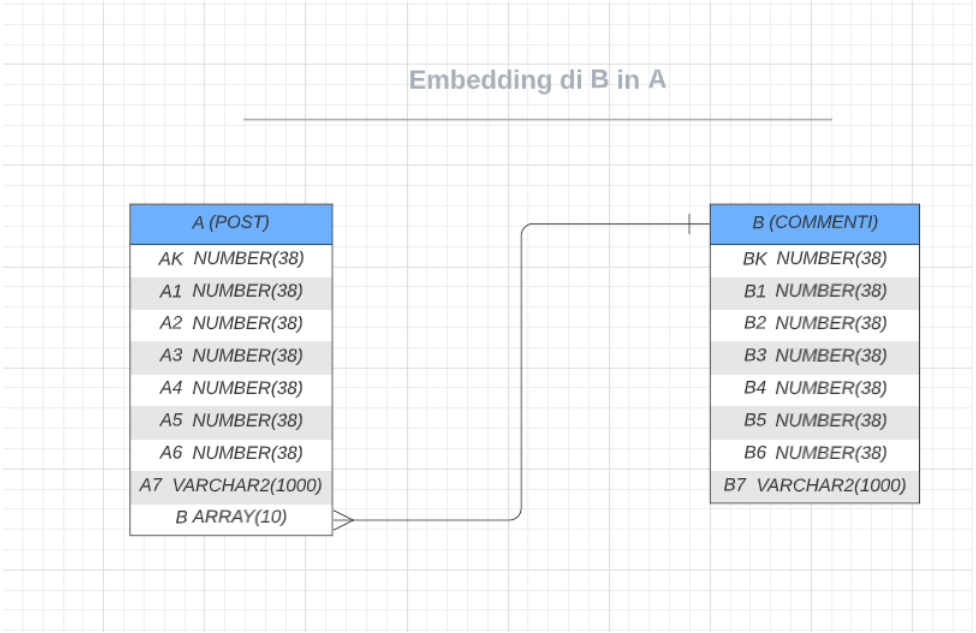
//TODO cambiare assolutamente queste foto





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





//TODO section che spiega le operanzioni CRUD

//TODO section che spiega le query che verranno effettuate sulle collezioni di dati (selezione e update)

4 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, di cui però non si terrà conto in questa tesi dato che i dati sono stati inseriti tramite strumenti come mongoimport per mongoDB o delle semplici richieste HTTP per couchDB e quindi i tempi sono più o meno simili.

//TODO inserire un grafico con i tempi non sarebbe male

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, sia 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'
```

Data la relazione tra A e B, sono state eseguite poi delle query di selezione di entrambi gli oggetti di una relazione dopo avere eseguito un join sulle chiavi primarie e leggendo un particolare valore degli attributi.

```
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'
```

I valori letti all'interno delle query sono valori randomizzati, ottenuti tramite le cardinalità delle entità, propri per ciascun attributo, in Python con ad esempio la seguente funzioni che prende in entrata l'indice sulla quale cercare e ne restituisce un numero valido.

```
expA = 5

expB = 6

N_A = 10**expA

N_B = 10**expB

N_A1 = N_A/10
```

```
N_A2 = N_A/10**(round(expA/2))

N_A3 = N_A/10**(expA - 1)

N_B1 = N_B/10

N_B2 = N_B/10**(round(expB/2))

N_B3 = N_B/10**(expB - 1)

 def get_random_indexed_int_A(ind):
  if "1" in ind or "4" in ind :
    return randint(0, N_A1 - 1)
  elif "2" in ind or "5" in ind:
    return randint(0, N_A2 - 1)
  elif "3" in ind or "6" in ind:
    return randint(0, N_A3 - 1)

def get_random_indexed_int_B(ind):
  if "1" in ind or "4" in ind :
    return randint(0, N_B1 - 1)
  elif "2" in ind or "5" in ind:
    return randint(0, N_B2 - 1)
  elif "3" in ind or "6" in ind:
    return randint(0, N_B3 - 1)
```

Le query di **update** sono state strutturate in modo abbastanza simile a quelle di letttura

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

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

//TODO inserire come ho creato i dataset /generazione File

//TODO valutare se queste section di inserimento sarebbero meglio all interno dei propri capitoli

5 Risultati inserimento dati

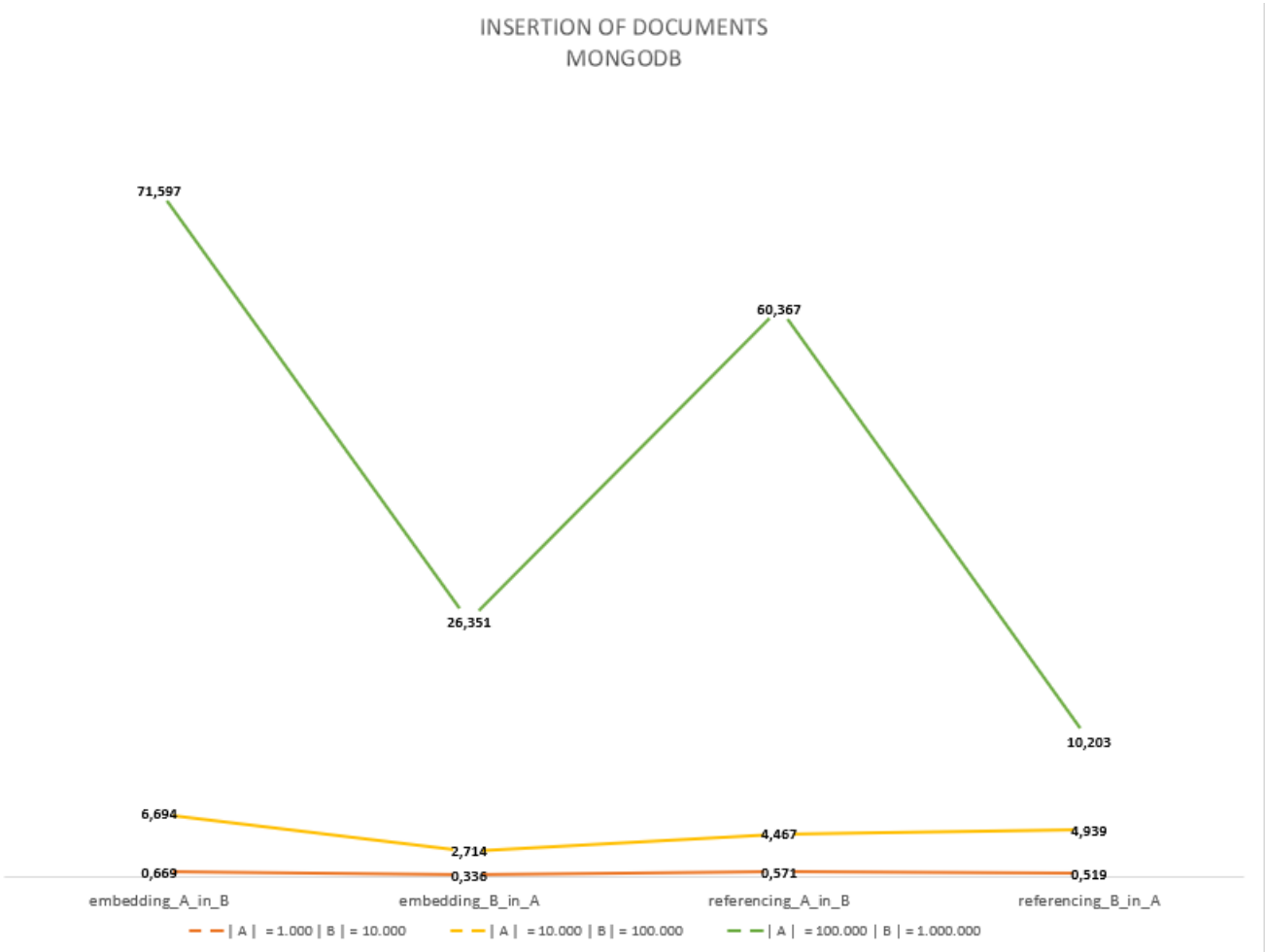
Dopo la creazione dei dataset delle entità A e B, di diverse dimensioni, sono stati verificati i tempi di inserimento per ogni software utilizzato, ognuno utilizza un differente metodo per la creazione di un documento.

5.1 MongoDB

L'applicativo di mongoDB su un sistema Linux contiene già al suo interno gli eseguibili *mongoimport* e *mongoexport* che sono stati ampiamente 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
```



5.2 CouchDB

CouchDB utilizza una POST cosi strutturata per inserire un nuovo documento

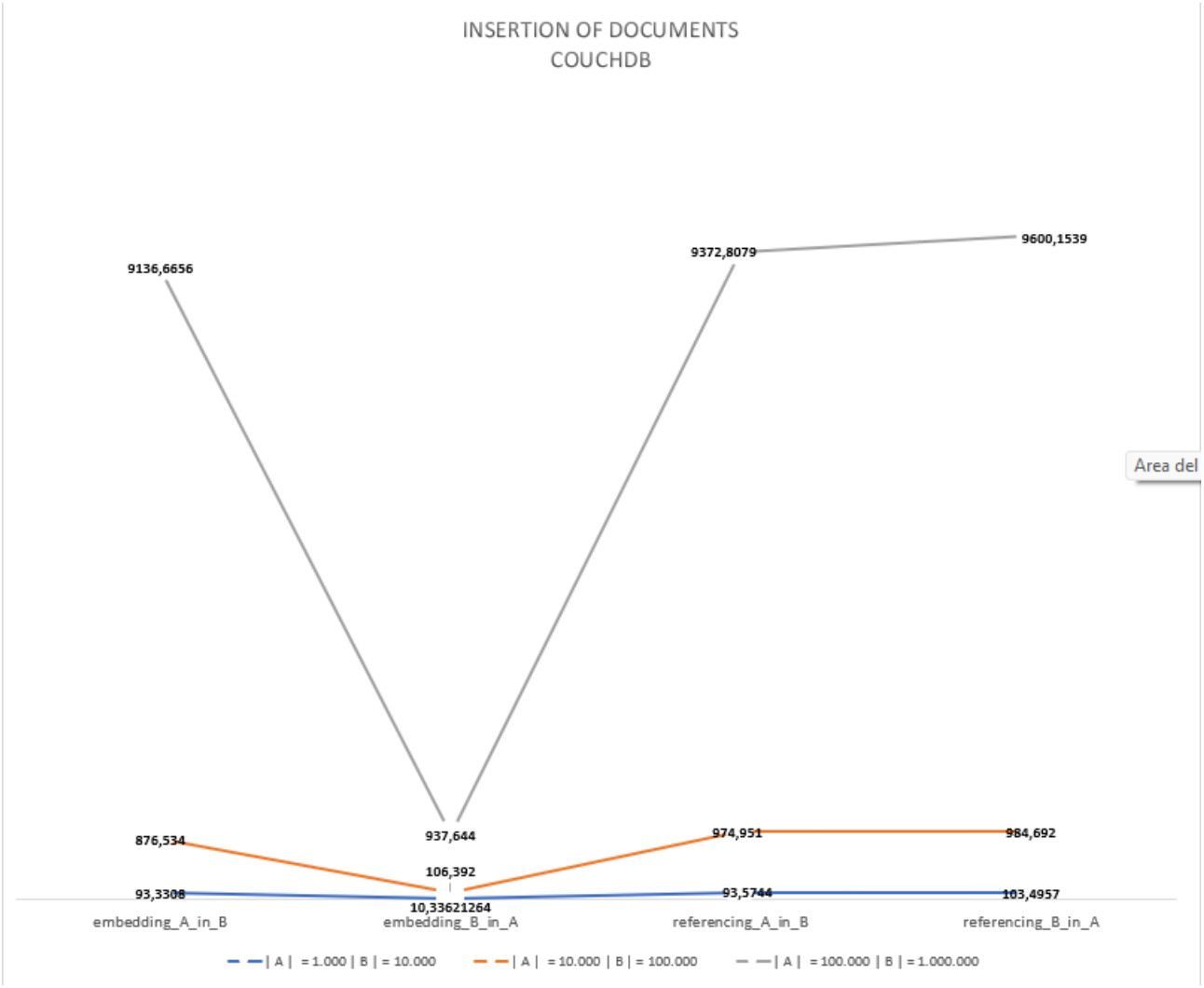
Request

```
POST /db HTTP/1.1
Accept: application/json
Content-Length: 81
Content-Type: application/json

{
    "_id": "FishStew",
    "servings": 4,
    "subtitle": "Delicious with fresh bread",
    "title": "Fish Stew"
}
```

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": "FishStew",
    "ok": true,
    "rev": "1-9c65296036141e575d32ba9c034dd3ee"
}
```

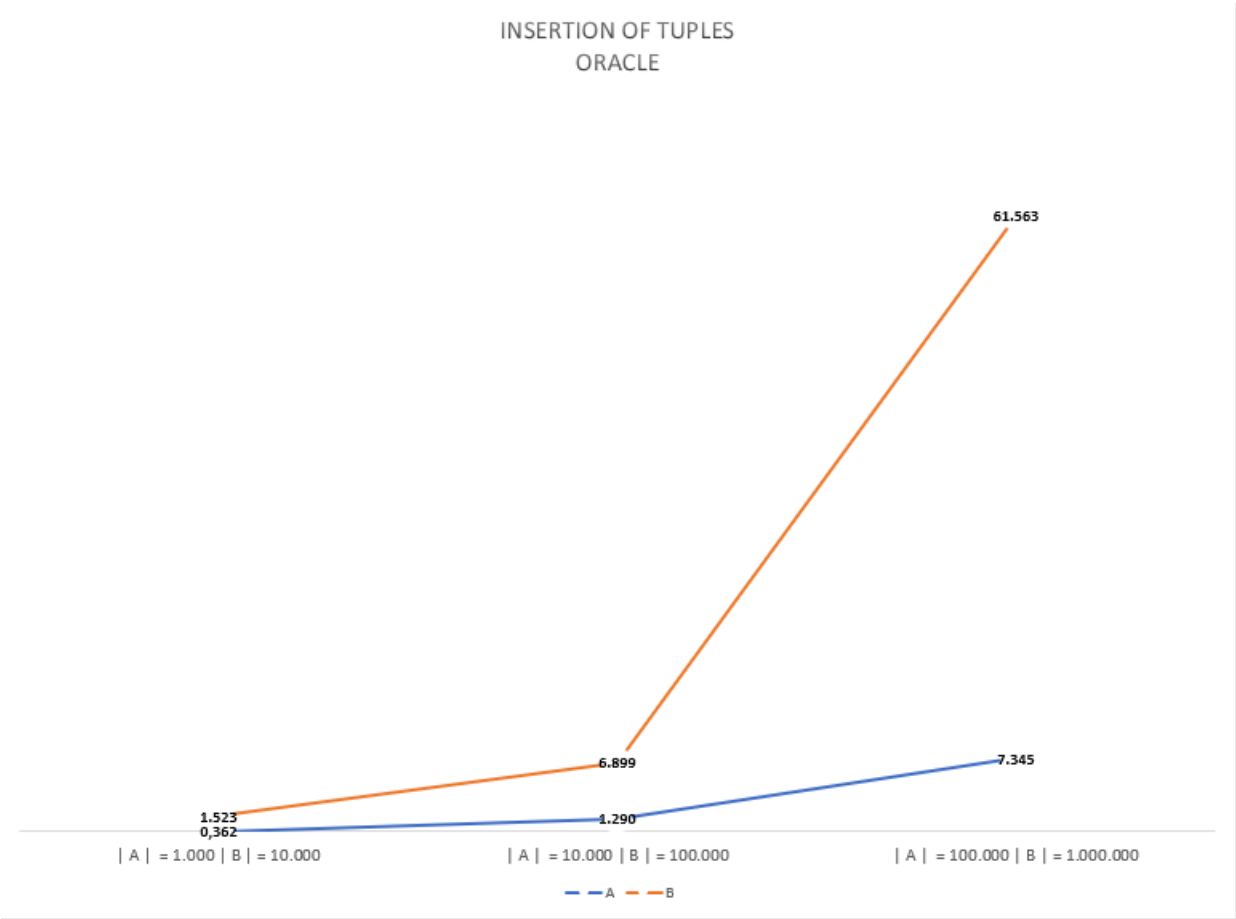
Quindi è stato scritto uno script in python che itera per ogni documento JSON del dataset ed esegue una POST all'indirizzo del database, specificando che il campo `_id`, che è chiave primaria di ogni documento, corrispondesse alla chiave dell'entità.



Purtroppo è risultato che questo è il metodo di gran lunga meno effeicente, 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 instanza, nel caso di multiple instanze le request sarebbero state ripartite sui nodi e i tempi sarebbero stati ridotti di molto.

5.3 Oracle

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 cospiqua di dati non è piu possibile utilizzare quel metodo che funziona nei piccoli numeri. Il software non riesce a completare quei task cosi 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.


INSERTION OF TUPLES
ORACLE
61.563
6.899
1.523
0,362
1.290
7.345
| A | = 1.000 | B | = 10.000
| A | = 10.000 | B | = 100.000
| A | = 100.000 | B | = 1.000.000
A
B

Costi

1 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:

1.1 NL

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

1.2 g

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

1.3 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.

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

1.4 NP

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

1.5 Costo del nested loop

In cui ho R relazione esterna e S relazione interna

- senza predicato di selezione

$$NP_R + NR_R * NP_S$$

- con predicato di selezione

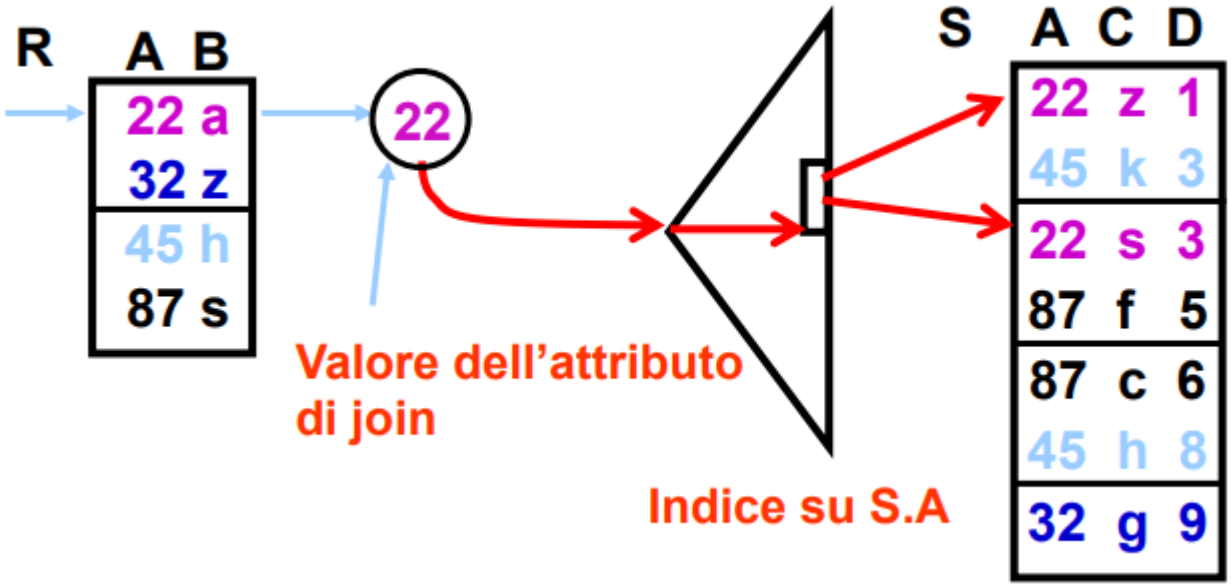
$$NP_R + (sel(pred) * NR_R) * costo(S)$$

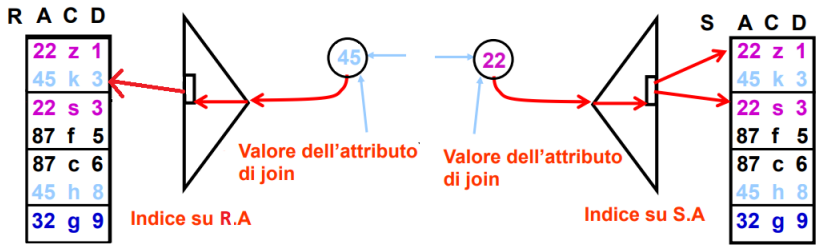
1.6 Indice clustered

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

1.7 Indice unclustered

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




R A C D
22 z 1
45 k 3
22 s 3
87 f 5
87 c 6
45 h 8
32 g 9
45
Valore dell'attributo di join
Indice su R.A
22
Valore dell'attributo di join
Indice su S.A
S A C D
22 z 1
45 k 3
22 s 3
87 f 5
87 c 6
45 h 8
32 g 9

MongoDb

1 Caratteristiche

MongoDb e' uno dei piu' diffusi database non relazionali orientato ai documenti, di tipo NoSQL, utilizza documenti in un formato JSON al posto delle tipiche tabelle dei sistemi relazionali. Piu' 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, 1º gennaio 1970)
- byte array (dati binari)
- booleani (true e false)
- NULL
- oggetto BSON
- array BSON
- espressioni regolari
- codice JavaScript

2 Organizzazione dei dati

2.1 Creazione collezioni

Per poter originare i set di dati sui quali lavoriamo si e' partiti da 2 enormi file JSON generati al paragrafo ??????????? //TODO riguarda qua // contenenti POST (A) e COMMENTI(B). Il paramentro $out posto dopo la query ha creato una collezione a tutti gli effetti sulla quale poter eseguire le query all'interno della rete del container.

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"
}
])
```

Referencing di A in B


```
"_id": 21748,
"BK": 21748,
"AK": 70394,
"B1": 85337,
"B2": 255,
"B3": 6,
"B4": 85337,
"B5": 255,
"B6": 6,
"B7": 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.'
  'Fusce lacinia eget arcu et maximus. Ut tempus est sit amet'
  'tortor commodo, sit amet facilisis mi rhoncus. Donec et elit'
  'venenatis, consequat tellus eu, tristique orci. Duis'
  'tristique sem ut nulla ullamcorper, a porta risus efficitur.'
```

```
'Cras sed neque et nisl tincidunt vestibulum. Phasellus'
'tristique tempor facilisis. Sed facilisis lectus eros, sed'
'aliquet lacus elementum sed. Integer vel dictum mi.'
'Maecenas pharetra tempus eros, efficitur mattis erat cursus'
'in. Nulla sit amet quam velit. Nullam tempus dictum lacus'
'id porttitor. Vestibulum facilisis pulvinar fermentum.'
'Ut elementum maximus feugiat. In at mollis leo, eu'
'facilisis magna. Vestibulum sed nisi ultricies, tincidunt'
'enim ac, fringilla ex. Phasellus pharetra mollis nisi'
'a fermentum. In nec faucibus nulla, eget molestie magna.'
'Vivamus in gravida ex. Aenean scelerisque gravida'
'ipsum, nec congue enim posuere sit amet. Donec vitae felis'
'id sem congue blandit eget non justo quis'
```

Referencing di A in B

Il referencing di B in A invece richiede un join, un' operazione piu' onerosa per la quale si e' dovuto passare il parametro allowDiskUse=true per poter utilizzare a pieno la memoria della macchina e quindi eseguire il join. Trovando di fatto le chiavi e aggiungendole all'array B che arrivera' a contenere 10 chivi 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}
)
```

Referencing di B in A

```
"_id": 8518,
"AK": 8518,
"A1": 6773,
"A2": 253,
"A3": 4,
"A4": 6773,
"A5": 253,
"A6": 4,
"A7": 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.'
  'Fusce lacinia eget arcu et maximus. Ut tempus est sit amet'
  'tortor commodo, sit amet facilisis mi rhoncus. Donec et elit'
  'venenatis, consequat tellus eu, tristique orci. Duis'
  'tristique sem ut nulla ullamcorper, a porta risus efficitur.'
  'Cras sed neque et nisl tincidunt vestibulum. Phasellus'
  'tristique tempor facilisis. Sed facilisis lectus eros, sed'
  'aliquet lacus elementum sed. Integer vel dictum mi.'
  'Maecenas pharetra tempus eros, efficitur mattis erat cursus'
  'in. Nulla sit amet quam velit. Nullam tempus dictum lacus'
  'id porttitor. Vestibulum facilisis pulvinar fermentum.'
  'Ut elementum maximus feugiat. In at mollis leo, eu'
```

```
  'facilisis magna. Vestibulum sed nisi ultricies, tincidunt'
  'enim ac, fringilla ex. Phasellus pharetra mollis nisi'
  'a fermentum. In nec faucibus nulla, eget molestie magna.'
  'Vivamus in gravida ex. Aenean scelerisque gravida'
  'ipsum, nec congue enim posuere sit amet. Donec vitae felis'
  'id sem congue blandit eget non justo quis',
"B": [
  182143,
  450648,
  120258,
  151670,
  314940,
  281717,
  335311,
  802016,
  313524,
  537
]
```

Referencing di B in A

Embedding

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

```
"_id": 963593,
"B1": 77221,
"B2": 86,
"B3": 1,
"B4": 77221,
"B5": 86,
"B6": 1,
"B7": [ ..... ],
"A": [
  {
    "AK": 70394,
    "A1": 7960,
    "A2": 269,
    "A3": 8,
    "A4": 7960,
```

```
    "A5": 269,
    "A6": 8,
    "A7": [ ..... ]
  }
]
```

Embedding di B in A

Embedding di B in A

```
  db.B.aggregate(
[
 {
    $group: {
      _id: {"AK" : "$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}
```

Embedding di B in A


```
"_id": 48,
"AK": 48,
"A1": 691,
"A2": 154,
"A3": 0,
"A4": 691,
"A5": 154,
"A6": 0,
"A7": [ ..... ],
"B": [
  {
    "BK": 844575,
    "B1": 94813,
    "B2": 140,
    "B3": 2,
    "B4": 94813,
    "B5": 140,
    "B6": 2,
    "B7": [ ..... ]
  },
  {
    "BK": 115006,
    "B1": 36,
    "B2": 483,
    "B3": 8,
    "B4": 36,
    "B5": 483,
    "B6": 8,
    "B7": [ ..... ]
```

```
  },
  .
  .
  .
  .
  {
    "BK": 834226,
    "B1": 77768,
    "B2": 707,
    "B3": 6,
    "B4": 77768,
    "B5": 707,
    "B6": 6,
    "B7": [ ..... ]
  }
]
```

Embedding di B in A

2.2 Indici

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

MongoDB utilizza, se non esplicitamente indicato, come strutture dati per gli indici dei B-tree. Dato che e' stato scelto di adoperare l'embedding di interi documenti sono stati indicizzati anche gli attributi dei documenti innestati sui quali saranno esegeuite delle query, anche sulle foreign del referencing sono stati costruiti indici.

```
// 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});
```

3 Container

//TODO

4 Risultati

	embedding_A_in_B	embedding_B_in_A	referencing_A_in_B	referencing_B_in_A
A0	2576	1.3	0.5	0.3
A1	2512.3	631.3	95.2	100.9
A2	2547.8	632.4	89.6	106.4
A3	2923.5	724.5	97.6	110.3
A4	6.3	0.3	0.4	114.8
A5	10	2.4	1.9	103.8
A6	1143.3	413	64.9	104.7
B0	1.3	1044.7	165.8	1.4
B1	1879.9	1028.5	1491.3	1470.3
B2	1899.4	1099.7	1537.2	1518.8
B3	2327	2988.2	1661.3	1578.9
B4	3.7	2.2	1.2	1.2
B5	16	36.5	19.6	12
B6	1482.4	2902.5	754	736.9
A0j	2586.9	0.8	2.5	0.7
A1j	2519.4	736.3	108.8	123.2
A2j	2641.1	821.8	212	222.9
A3j	2668.5	733.9	11241.7	10990.1
A4j	1	0.2	12.2	9.6
A5j	2.7	0.6	104.2	110.6
A6j	804.4	305.7	10942.5	10779.3
B0j	0.8	1090.1	0.3	0
B1j	1981.2	1054.8	1415.1	1399.1
B2j	1951.8	1084	1605.2	1453.1
B3j	2045.4	2964	11818.2	3840
B4j	0.1	0.5	1.2	0.2
B5j	35	61.7	126.6	43.3
B6j	1215.4	2894.6	10936	3041.2

Oracle

1 Introduzione

Oracle e' una societa' che mette a disposizione numerosi prodotti, per la parte di confronto relazionale e' stato usato oracle SQL developer, un software opensource che mette a disposizione un software per poter lavorare con SQL su database di Oracle attraverso il JDK (java development kit).

2 Raccolta dati

3 Organizzazione dei dati

Utilizzando un sistema relazionale ci si e' dovuti limitare alla crezione di due collezioni distinte A (Post) e B(Commenti) sulle quali eseguire le query di selezione per poter ottenere una stima dei costi su un modello relazionale che non utlizasse embedding e referencing.

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

```
- Primary key per A

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

- Primary key per B

alter table "SYS"."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 BITMAP INDEX A_A4_BITMAP ON A (A4 ASC);
CREATE BITMAP INDEX A_A5_BITMAP ON A (A5 ASC);
CREATE BITMAP INDEX A_A6_BITMAP ON A (A6 ASC);
CREATE BITMAP INDEX B_B4_BITMAP ON B (B4 ASC);
CREATE BITMAP INDEX B_B5_BITMAP ON B (B5 ASC);
CREATE BITMAP INDEX B_B6_BITMAP ON B (B6 ASC);
CREATE BITMAP INDEX B_FK_BITMAP ON B (FAK ASC);

- Ridurre la cache del client di sistema

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

Per poter eseguire le query di selezione e' stato utilizzato uno script in python che utilizza il modulo `cx_Oracle` 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 A6 = 612;
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

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

	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

CouchDb

1 Caratteristiche

E' un sistema di gestione di basi di dati non relazionali. E' stato creato soprattutto per il mondo Web, progettato per poter gestire sia importanti quantita' di richieste attraverso la rete sia per poter immagazzinare i dati in dispositivi mobili ed essere veloce. Utilizza una HTTP/JSON API per poter gestire le richieste ed inviare i dati, quindi fa ampio utilizzo di POST, PUT, DELETE e cosi via. E' diventato un progetto Apache nel 2008, versione OpenSource di CouchServer.

Per poter 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'e' la mancanza dell'operatore di lookup ossia di join, infatti i tempi che presentero' successivamente sulle query di join sono sostanzialmente la somma di query di selezione sulle varie collezioni.

2 Popolamento

Per poter popolare i database contenti i diversi tipi di collezione si e' utilizzato le collezioni di documenti precedentemente create attraverso MongoDB e per ogni riga, ossia documento, attraverso una post all'indirizzo locale del server di CouchDB si e' potuta eseguire una query.

CouchDB richiede che il campo `_id` sia una stringa, quindi attraverso il comando sed in uno script in bash si e' potuto riutilizzare le collezione esportate da MongoDB, dato che il join non e' tra gli operatori di Mango, per generare nuovamente le collezioni.

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

```
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)
```

Listing 2: Caricamento di un documento CouchDB

3 Indici

Gli indici all'interno di ogni collezione sono stati creati tramite uno script in python con un procedimento simile, per poter ottenere collezioni uguali in tutto e per tutto a quelle di MongoDB. Mango e' un linguaggio di query dichiarativo JSON per CouchDB. Gli indici di Mango con tipo json, sono costruiti usando viste MapReduce attraverso il seguente script in python:

```
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    },
```

```
  "name": ind + "-index",
  "type": "json"
})
response = requests.request("POST", url, headers=headers, data=payload)

payload = json.dumps({
"index": {
  "fields": [
    "BK"
  ]
},
"name": "BK" + "-index",
"type": "json"
})
response = requests.request("POST", url, headers=headers, data=payload)

for collection in collections_a:
  url = "http://admin:admin@127.0.0.1:5984/" + collection + "/_index?partitioned=true
  for ind  in ind_a:
      payload = json.dumps({
        "index": {
          "fields": [
            ind
          ]
        },
        "name": ind + "-index",
        "type": "json"
      })
      response = requests.request("POST", url, headers=headers, data=payload)

for collection in collections_b:
  url = "http://admin:admin@127.0.0.1:5984/" + collection + "/_index?partitioned=true
  for ind  in ind_b:
      payload = json.dumps({
        "index": {
          "fields": [
            ind
          ]
        },
```

```
        "name": ind + "-index",
        "type": "json"
      })
      response = requests.request("POST", url, headers=headers, data=payload)

url = "http://admin:admin@127.0.0.1:5984/referencing_a_in_b/_index?partitioned=true"
payload = json.dumps({
"index": {
  "fields": [
    "AK"
  ]
},
"name": "AK" + "-index",
"type": "json"
})
response = requests.request("POST", url, headers=headers, data=payload)

# emb AB
url = "http://admin:admin@127.0.0.1:5984/embedding_a_in_b/_index?partitioned=true"
for ind in ind_a:
payload = json.dumps({
  "index": {
    "fields": [
      "A." + ind
    ]
  },
  "name": "A." + ind + "-index",
  "type": "json"
})
response = requests.request("POST", url, headers=headers, data=payload)

# emb BA
url = "http://admin:admin@127.0.0.1:5984/embedding_b_in_a/_index?partitioned=true"
for ind in ind_b:
payload = json.dumps({
  "index": {
    "fields": [
      "B." + ind
    ]
```

```
  },
  "name": "B." + ind + "-index",
  "type": "json"
})
response = requests.request("POST", url, headers=headers, data=payload)

# ref BA
url = "http://admin:admin@127.0.0.1:5984/referencing_b_in_a/_index?partitioned=true"
payload = json.dumps({
"index": {
  "fields": [
    "B"
  ]
},
"name": "B" + "-index",
"type": "json"
})
response = requests.request("POST", url, headers=headers, data=payload)
```

4 Container

Per poter utilizzare CouchDB, come per MongoDB, e' stato deciso di adoperare un container docker nel quale poter muovere comodamente i dati e organizzare un interfaccia sulla quale poter lavorare.

Tramite uno script la memoria e' stata limitata e tramite un interfaccia web-based chiamata Project Fauxton, è stato possibile interagire con le collezioni di dati per poter testare query e amministrare generalmente il database.

//TODO inserire la foto delle collezioni con indici e A,B

```
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
```

5 Risultati

	embedding_A_in_B	embedding_B_in_A	referencing_A_in_B	referencing_B_in_A
A1	75189.332	34471.335	140895.697	51247.2
A2	8349.807	7989.056	15699.93	9581.866
A3	101.425	98.243	181.265	115.518
A4	71597.284	31.813	82.354	51.148
A5	8631.563	54.654	128.021	134.083
A6	110.197	62.023	144.409	72.947
B1	285188.803	28669.218	527133.738	343064.969
B2	6998.93	532.826	14976.854	9221.521
B3	91.907	14.276	147.44	119.905
B4	39.654	30737.048	73.722	56.231
B5	62.485	494.581	105.319	83.123
B6	57.733	16.16	123.626	122.586
A1j	71474.39	28136.817	63191.924	48265.973
A2j	8346.599	7764.238	19266.786	9213.109
A3j	106.766	103.708	3447.027	269.924
A4j	82670.276	18.806	1084.652	83.669
A5j	8500.238	48.353	2927.926	213.943
A6j	99.526	61.878	2840.764	160.212
B1j	262027.396	27951.846	481069.029	313197.745
B2j	6980.902	526.624	17093.095	9438.85
B3j	89.303	15.736	381.211	288.027
B4j	27.341	28554.843	117.376	95.116
B5j	56.489	529.836	280.386	281.755
B6j	49.414	13.703	284.724	223.645

//TODO update per couchDB

Minimo documento
Appendice

Ringraziamenti