UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Ingegneria Elettrica e delle tecnologie dell'informazione

Primo Homework del corso di

BIG DATA ENGINEERING

Professore:
Giancarlo Sperlì

Candidati:
Benfenati Domenico M63001165
Carandente Vincenzo M63001229

ANNO ACCADEMICO 2021/2022
Secondo Semestre

Indice

Capitolo 1

Introduzione

1.1 Business Understanding

Tale elaborato ha l'obiettivo di sviluppare un database NoSQL per memorizzare un insieme di dati relativi alle recensioni effettuate dagli utenti riguardo alcuni locali e ristoranti provenienti dalla **Yelp**, una S.P.A. americana con sede a San Francisco, in California, che pubblica recensioni di crowdsourcing sulle attività commerciali.
Lo scopo dell'elaborato è quello di definire ed implementare un modello dei dati in un database NoSQL, mettendo in evidenza il motivo della scelta del modello e della tecnologia. Inoltre, per valutare i vantaggi che ne derivano sono state realizzate diverse query sul database.
Per la memorizzazione dei dati si è scelto di utilizzare **MongoDB**, un database NoSQL orientato ai documenti. Il database è stato installato in cloud, sfruttando il piano *Atlas* di MongoDB, ed è stato gestito attraverso un'interfaccia grafica proprietaria, **MongoDB Compass**, la quale permette, tra le altre cose, di creare un database, definire delle collection di documenti, realizzare query e operazioni CRUD.

1.2 Data Characterization

Nella sezione corrente si vogliono descrivere con maggiore precisione i dati a disposizione, i quali possono essere divisi in differenti categorie: **Business**, **Review**, **Users**, **Checkin**, **Tips**. È bene sottolineare che, dovendo implementare un database documentale, i dati utilizzati per popolarlo provengono da file in formato *.json*.

1.2.1 Business

La prima categoria di dati consiste in 150346 istanze, e include tutte le informazioni relative agli esercizi commerciali che sono inclusi all'interno delle classifiche Yelp.

In particolare, il file considerato nell'elaborazione è *yelp_academic_dataset_business.json* e contiene informazioni riguardo i vari esecizi commerciali quali posizione, città, valutazione in numero di stelle, numero di recensioni ricevute, e altre informazioni riguardo gli "attributi" del locale.

1.2.2 Review

La seconda categoria di dati contiene circa 6990280 istanze, e include tutte le informazioni relative alle recensioni sulla piattaforma Yelp. In particolare il file è *yelp_academic_dataset_review.json* e contiene l'identificativo, l'id dell'utente e dell'esercizio commerciale, la data, il numero di stelle attribuite, il testo della recensione e dei interi che specificano il numero di voti utili, divertenti e hot che altri utenti hanno attributo a tale recensione. Data la numerosità dei dati, si è scelto di non considerare tale categoria ai fini dell'analisi.

1.2.3 User

La terza categoria di dati conta circa 1987897 documenti, e include tutte le informazioni relative agli utenti presenti all'intenro della piattaforma Yelp.
In particolare il file è *yelp_academic_dataset_user.json* e contiene l'identificativo dell'utente, il nome, la data d'iscrizione, il numero di recensioni da esso effettuate, una lista di amici ad esso collegati, e una serie di valori interi indicanti il conteggio di differenti tipologie di complimenti ricevuti dall'utente. Data la numerosità dei dati, si è scelto di non considerare tale categoria ai fini dell'analisi.

1.2.4 Checkin

La quarta categoria di dati aggrega circa 131930 istanze, e include tutte le informazioni relative ai check-in degli utenti all'interno degli esercizi commerciali.
In particolare il file è *yelp_academic_dataset_checkin.json* e contiene l'identificativo del locale e una lista di date corrispondenti ai checkin effettuati in quell'esercizio. Tale categoria non è risultata utile ai fini di un analisi, e si è quindi scelto di non considerare tali dati.

1.2.5 Tip

La quinta categoria di dati conta 908915 elementi, e include tutte le informazioni relative ai suggerimenti inseriti dagli utenti all'interno della piattaforma Yelp.
In particolare il file considerato nell'elaborazione è *yelp_academic_dataset_tip.json* e contiene informazioni circa l'utente che ha inserito il suggerimento, il locale a cui è destinato, e la data di inserimento, nonchè il testo in chiaro inserito dall'utente.

Capitolo 2

Data Model

Nel capitolo corrente sono riportate le operazioni di manipolazione effettuate sui dati per la definizione di un modello dei dati che permetta una semplice fruizione da parte degli utenti finali, sulla base di quelle che si ritengono essere le operazioni più frequenti.

2.1 MongoDB

Osservando i dati a disposizione si è scelto di utilizzare il database NoSQL MongoDB, visto che la struttura degli stessi si presta infatti alla memorizzazione in un database documentale. Tutte le informazioni associate ad un singolo locale (indirizzo, nome, etc.) infatti possono essere modellate in un singolo documento. Lo stesso vale per le informazioni relative agli utenti che hanno effettuato le recensioni, o ai relativi consigli. É evidente che esistono relazioni tra recensione e utente, così come per il locale, ma tale legame non è stato ritenuto importante nel processo di interrogazione di tale database da parte di un utente, per il quale si ritiene più utile una vista di informazioni aggregate riguardo la proprio ricerca.
In aggiunta, l'uso di tale database è conveniente nel caso in cui i dati hanno una struttura dinamica o presentano un gran numero di campi opzionali, come accade nel dataset considerato per alcuni attributi dei vari documenti.

2.2 Creazione del Database

Attraverso il tool Compass è possibile instanziare un **client** MongoDB, connettersi al server in cloud, e creare un nuovo **database**. Si creano quindi due **collection** all'interno del database, che conterranno rispettivamente i documenti relativi ai locali recensiti (Business), e i consigli degli utenti sui locali da essi recensiti (Tips). Una volta create le collection, il database può essere popolato inserendo i dati contenuti nei file .json descritti precedentemente.
Nonostante la natura *schema-less* di un database NoSQL, si può comunque notare una struttura

ben precisa tra i documenti inseriti all'interno della collection. Si riporta un esempio di documento per entrambe le collection.

Business


```
_id: ObjectId('6259c1d87f16273d78fdcee1')
business_id: "1SWheh84yJXfytovILXOAQ"
name: "Arizona Biltmore Golf Club"
address: "2818 E Camino Acequia Drive"
city: "Phoenix"
state: "AZ"
postal_code: "85016"
latitude: 33.5221425
longitude: -112.0184807
stars: 3
review_count: 5
is_open: 0
attributes: Object
  GoodForKids: "False"
categories: "Golf, Active Life"
hours: null
```


Tips


```
_id: ObjectId('6259d91c7f16273d7800bf49')
user_id: "UPw5DWs_b-e2JRBS-t37Ag"
business_id: "VaKXUpmWTTWDKbpJ3aQdMw"
text: "Great for watching games, ufc, and whatever else tickles yer fancy"
date: "2014-03-27 03:51:24"
compliment_count: 0
```


2.3 Preprocessing

Prima di procedere, si converte il formato delle date presenti nel documento della collection **Tips** dal tipo `string` al tipo `date`. In questo modo, si facilitano le interrogazioni effettuate da parte dell'utente. Per effettuare la conversione si utilizza l'aggregazione **$set**, che permette di aggiornare più documenti presenti in una determinata collection. Di seguito il codice della conversione.

```
{$set: {
    "date": {$dateFromString:{dateString: "$date"}}
    }
}
```

Altra operazione di preprocessing da effettuare è quella per il parsing di alcune feature dell'attributo *attributes* della collection **Business**, definite come stringhe ma indicanti un oggetto di tipo JSON. La conversione non è stata però possibile in quanto il piano Atlas Cloud free non permette l'utilizzo della funzionalità **$function**, ma per completezza se ne riporta ugualmente la sintassi applicata ad un campo dell'attributo.

```
{$set: {
    $function: {
        body: function(jsonString){
            return JSON.parse(jsonString)
        },
        args: ["$attributes.GoodForMeal"],
        lang: "js"
        }
    }
}
```

Infine, si è scelto di scompattare l'attributo *hours*, e per ogni giorno della settimana sono stati definiti orario di apertura e di chiusura con un formalismo semplificato. Purtroppo non è stato possibile convertire tali orari nel tipo `date` vista la scorretta formattazione delle ore, indicate nel formato *one-digit* e non tramite *two-digit*. Si riporta un esempio di tale conversione, per gli orari del sabato.

```
{$match: {
    "hours.Saturday": {$exists: "True"}
    }
},
{$project: {
    "OrariSabato": { $split: ["$hours.Saturday", "-"] }
    }
},
{$set: {
    "AperturaSabato": { $arrayElemAt: ["$OrariSabato", 0] },
    "ChiusuraSabato": { $arrayElemAt: ["$OrariSabato", 1] }
    }
}
```

Output di esempio

```
_id: ObjectId('6259c1d87f16273d78fdcee3')
name: "Musashi Japanese Restaurant"
city: "Charlotte"
› OrariSabato: Array
AperturaSabato: "17:30"
ChiusuraSabato: "22:0"
```

2.4 Creazione delle Views

Sulla base delle interrogazioni più frequenti che si ritiene vengano effettuate sui dati sono state definite diverse view sulle due collection. Una view non è altro che il risultato readonly di una

aggregation pipeline eseguita su una collection. Le aggregation pipeline sono strumenti offerti da MongoDB che permettono di eseguire una serie di operazioni aggregate in grado di trasformare i documenti presenti in una collection. Le viste definite permettono quindi di accedere facilmente ai risultati delle aggregazioni più frequenti, senza richiedere che l'utente definisca la pipeline.

2.4.1 Vista per città

Si è scelto di creare una prima vista per la visualizzazione dei locali all'interno di una città, aggregando il numero di locali presenti, tra totale e numero di locali aperti, il numero di recensioni effettuate per i locali di quella città, e la media di stelle dei ristoranti di quella città. Di seguito il codice e un esempio di output.

```
{$project: {
    city: 1,
    state: 1,
    categories: 1,
    stars: 1,
    review_count: 1,
    is_open: 1
    }
},
{$group: {
    _id: "$city",
    "Totale-Locali": {$count: {}},
    "di-cui-Aperti": {$sum: '$is_open'},
    "Totale-Recensioni": {$sum: "$review_count"},
    "Numero-Stelle-Medio": {$avg: "$stars"}
    }
}
```

Output d'esempio

```
_id: "Bridgeville"
Totale-Locali: 191
di-cui-Aperti: 165
Totale-Recensioni: 3482
Numero-Stelle-Medio: 3.4214659685863875
```

Grazie a tale vista è possibile rapidamente, tramite una query, visualizzare quale sia la città con i ristoranti migliori, in termini di numero medio di stelle e numero totale di recensioni fatte. Di seguito la query suddetta con l'output finale.

```
{$sort: {
    "di-cui-Aperti": -1,
```

```
    "Numero-Stelle-Medio": -1,
    "Totale-Recensioni": -1
    }
},
{$group: {
    _id: "$allElementsTrue",
    "Miglior-Città": {$first: "$_id"}
    }
}
```

Output d'esempio

```
_id: null
Miglior-Città: "Las Vegas"
```

2.4.2 Vista per Join tra Documenti

Successivamente, per permettere la fruizione della collection Tips, si è scelto di effettuare un unione tra i documenti in tale collezione e quelli all'interno della collection Business; tale unione è resa possibile dal richiamo all'id del locale presente in ogni documento della collection Tips.

É stata quindi effettuata la join tra tali documenti, sfruttanto il comando **lookup** e creata una vista dei documenti risultanti dall'unione. Tali comandi vengono riportati di seguito, corredati di un esempio di output.

Come si nota dai comandi, è stato necessario convertire il campo importato in un oggetto, in quanto, tramite il comando *lookup* si ottiene un array che contiene un elemento per ogni documento importato. Nel caso di tale join il documento inglobato è unico, e l'array si compone di un singolo elemento, vista l'univocità del campo **business_id**.

```
{$lookup: {
    from: "Business",
    localField: "business_id",
    foreignField: "business_id",
    as: "business"
    }
},
{$set: {
    business: {$arrayElemAt: ["$business",0]}
    }
}
```

Output d'esempio


```
_id: ObjectId('6259d91c7f16273d7800bf4d')
user_id: "LUlKtaM3nXd-E4N4uOk_fQ"
business_id: "AkL6Ous6A1atZejfZXn1Bg"
text: "Molly is definately taking a picture with Santa lols"
date: 2012-10-06T00:19:27.000+00:00
compliment_count: 0
business: Object
  _id: ObjectId('6259c1da7f16273d78fdd8b2')
  business_id: "AkL6Ous6A1atZejfZXn1Bg"
  name: "Petagogy"
  address: "5880 Ellsworth Ave"
  city: "Pittsburgh"
  state: "PA"
  postal_code: "15232"
  latitude: 40.4576629
  longitude: -79.9280108
  stars: 4.5
  review_count: 30
  is_open: 1
  attributes: null
  categories: "Pet Stores, Pets"
  hours: Object
```


2.5 Query effettuate

2.5.1 Miglior ristorante aperto per ogni città

Si è scelto di selezionare il miglior ristorante per ogni città, effettuando un raggruppamento per città, cercandone i ristoranti ed identificandone il migliore facendo riferimento sul numero di stelle di tale ristorante. Di seguito la pipeline e un output esempificativo.

É doveroso far notare che il comando **$limit** è stato inserito unicamente per evitare il mancato completamento della pipeline dovuto ad un errore che segnala l'elevato numero di byte da elaborare. Tale pipeline è correttamente funzionante anche escludento tale limitazione.

```
{$match:{
    $and:[
        {city: {$exists: true}},
        {city: {$ne: ""}},
        {categories: /Restaurants/},
        {is_open: 1}
        ]
    }
},
{$sort:{
    city: 1,
    stars: -1
    }
},
{
    $limit: 2000
},
```

```
{$group:{
    _id: "$city",
    "Totale Ristoranti": {$count: {}},
    "Nome Locale": {$first: "$name"},
    "Indirizzo": {$first: "$address"},
    "Stelle Miglior Ristorante": {$first: "$stars"},
    "Review Locale": {$first:"$review_count"}
    }
}
```

Output d'esempio

```
_id: "Bainbridge Township"
Totale Ristoranti: 1
Nome Locale: "Panera Bread"
Indirizzo: "8480 E Washington St"
Stelle Miglior Ristorante: 2.5
Review Locale: 26
```

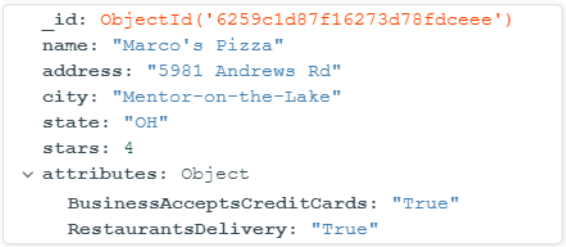
2.5.2 Locali con Delivery e Carta di Credito

Si è scelto di effettuare una query di ricerca anche per attributi del locale, come può essere ad esempio quella di ricercare quale esercizio accetta pagamenti con carta e permette di effettuare consegne a domicilio. É importante far notare che essendo la consegna attributo unicamente di ristoranti, non è necessario discretizzare i locali per categoria. Di seguito il codice e un esempio di output.

```
{$match: {
    $and:[
        {"attributes.RestaurantsDelivery":{$exists:true}},
        {"attributes.BusinessAcceptsCreditCards": {$exists:true}},
        {$expr:{
            $eq:[
                "$attributes.BusinessAcceptsCreditCards",
                "$attributes.RestaurantsDelivery"
                ]
                }
            }
        ]
    }
}
```

Output d'esempio

Per una corretta visualizzazione all'interno del documento, si è scelto di effettuare una proiezione delle sole caratteristiche chiave dell'output ottenuto.

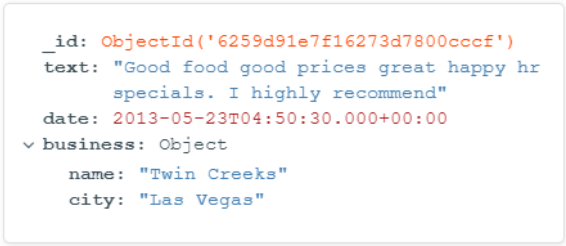


2.5.3 Light Feedback Analysis sui Tips per Locale

Si è pensato di analizzare i feedback lasciati dagli utenti in maniera blanda, controllando se essi contenessero alcune parole chiave relative ad un gradimento del locale sui cui gli utenti hanno espresso un parere, applicando la query alla vista che aggrega le due collection Tips e Business. Di seguito la pipeline e un output di esempio.

```
{$match:{
    $and: [ {text: /great/},{text: /happy/},{text: /good/} ]
    }
},
{$project:{
    "business.name": 1,
    "business.city": 1,
    date: 1,
    text: 1
    }
}
```

Output d'esempio



2.6 Creazione degli Indici

Dal momento che si prevede un elevato uso di query sulla collection *Business* che coinvolgono filtri e aggregazioni sul campo **city** e sull campo **name**, si è ritenuto opportuno definire un indice su tali campi. Questo consente di migliorare le prestazioni di query di questo tipo, in termini di tempo di esecuzione.

Si è quindi deciso di valutare l'incremento prestazionale di una semplice query esemplificativa, come quella in basso.

```
1 {city: "Phoenix"}
```

I risultati delle prestazioni ottenute sono riportati in figura.

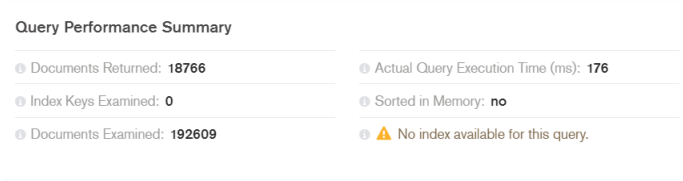


Figura 2.1: Valutazione query senza indice

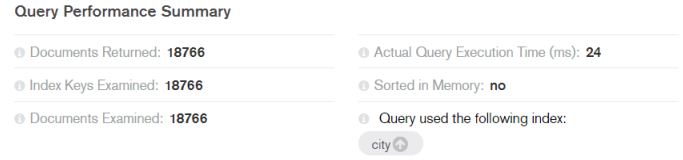


Figura 2.2: Valutazione query con indice

Tempi decisamente più discrepanti tra uso e non di un indice sulle query sono però apprezzabili eseguendo query computazionalmente più complesse.