# 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:**

Grasso Francesco Crescenzo M63001244

Palermo Emilio M63001178

Vitrano Arianna M63001171


ANNO ACCADEMICO 2021/2022

Secondo Semestre

# Indice

# Capitolo 1

# Introduction

L'obiettivo del primo Homework del corso di Big Data Engineering è quello di ricavare informazioni utili attraverso l'analisi di dati provenienti dal sito Yelp, una società che pubblica recensioni di crowdsourcing sulle aziende e gestisce un servizio di prenotazione di ristoranti online chiamato Yelp. Il set di dati fornitoci da Yelp è un sottoinsieme di attività, recensioni e dati degli utenti che abbiamo utilizzato per scopi accademici nonché educativi. Il lavoro svolto dal nostro Team Work è in particolare incentrato sull'utilizzo di MongoDB, un database NoSQL document-oriented ma non solo. Nei seguenti capitoli andremo ad esporre:

- una caratterizzazione dei dati, fondamentale per comprendere la tipologia di dati analizzati ed il dominio in cui abbiamo lavorato
- una descrizione del modello dei dati e relativa giustificazione di tool e tecnologie utilizzate per svolgere l'Homework
- una descrizione della procedura che ci ha portato alla creazione del database, importazione dei dataset su MongoDB, analisi e processazione dei dati. Successivamente alla presentazione delle pipeline per le varie query nelle quali vengono realizzate tecniche di Preprocessing, Sentimental Analysis o Join, vengono illustrati anche task e possibili riscontri pratici di ciascuna query

# Capitolo 2

# Data Characterization

Fra tutti dati presenti in Yelp, la nostra analisi è avvenuta considerando due tipi di categorie di dati: Business e Tips.

## 2.1 Business

Il file considerato nell'elaborazione è *yelp_academic_dataset_business.json* ed è grande 118.86 MB. Esso contiene un numero di elementi pari a 150.3 mila.

Questa categoria contiene informazioni relative alle imprese, compresi i dati sull'ubicazione, attributi e categorie. In particolare contiene:

- business_id: campo di tipo string di lunghezza fissa, 22 caratteri, che va a mappare in maniera univoca quelli che sono i documenti di tipo business
- name: campo di tipo string che indica il nome dell'attività
- address: campo di tipo string che indica l'indirizzo dell'attività
- city: campo di tipo string che indica la città in cui è situata l'attività
- state: campo di tipo string che rappresenta l'acronimo dello stato dove l'attività risiede
- postal code: campo di tipo string che indica il codice postale dell'attività
- latitude, longitude: campi di tipo float e indicano la posizione esatta dell'attività
- stars: campo di tipo float che indica il numero di stelle dell'attività, nonché la media delle valutazioni lasciate dagli utenti
- review_count: campo di tipo integer che indica il numero di recensioni scritte dagli utenti associati all'attività

- is_open: campo di tipo integer che rappresenta l'apertura o la chiusura dell'attività
- attributes: attributi aziendali a valori true/false come ad esempio BusinessParking con relative informazioni, BusinessAcceptCreditCard che indica la possibilità di pagare con carta di credito
- categories: array di stringhe che indica la categoria di business e ci ha consentito di ricercare la tipologia dei locali hours: oggetto costituito da una lista key-value in cui la chiave indica il giorno e il valore l'orario di apertura

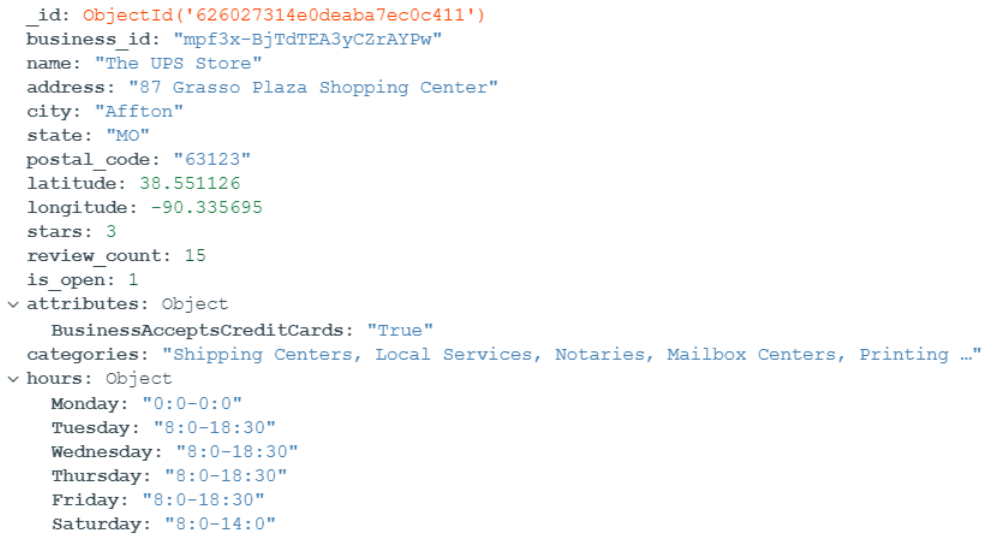


Figura 2.1: View

## 2.2 Tips

Il file considerato nell'elaborazione è *yelp_academic_dataset_tp.json* ed è grande 180.6 MB. Esso contiene un numero di elementi pari a 908.9 mila.

Questa categoria contiene informazioni sui suggerimenti lasciati dagli utenti. Quest'ultimi sono più brevi delle recensioni e tendono a dare un feedback immediato. In particolare contiene:

- Business_id: campo di tipo string di lunghezza fissa, 22 caratteri, che va a mappare in maniera univoca quelli che sono i documenti di tipo business
- user_id: campo di tipo string di lunghezza fissa, 22 caratteri, che va a mappare in maniera gli utenti autori dei Tips
- text: campo di tipo string che indica il commento sull'attività specificata dal campo business_id scritto dall'utente specificato dallo user_id

- date: campo di tipo string il cui formato è il seguente ”YYYY-MM-DD hh:mm:ss” e specifica data e ora di quando il Tip è stato scritto dall'utente
- compliment_count: campi di tipo integer che indica il numero di apprezzamenti associati al Tip

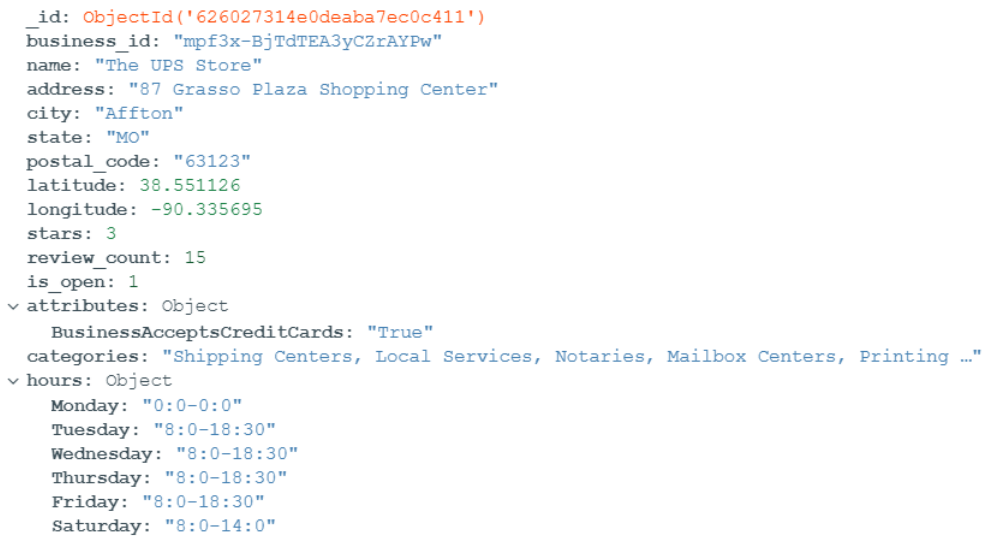


Figura 2.2: View

# Capitolo 3

# Data Model

## 3.1 Model in MongoDB

Il database utilizzato per l'elaborazione dell'homework è stato chiamato **Yelp** ed è stato creato in Mongo DB. Trattandosi di una tipologia di database document-oriented, risulta essere strutturato in collezioni e documenti. Esso è costituito, in particolare, da due collezioni di centinaia di migliaia di documenti le quali sono state chiamate **Business** e **Tip**. Di seguito ne viene riportata una rappresentazione ad alto livello.

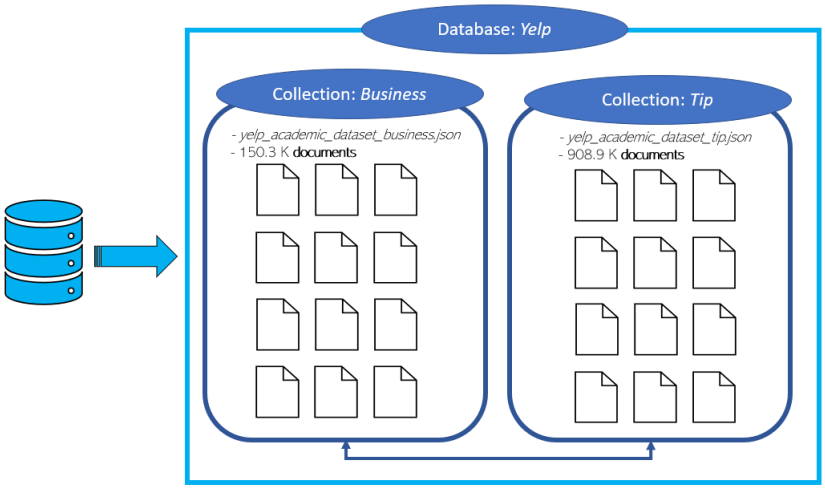


Figura 3.1: Data Model

## 3.2 Design choices: MongoDB

Per il nostro Homework abbiamo optato l'utilizzo di un database aggregate-oriented, in particolare di tipo documentale, Mongo DB. Il motivo di tale scelta risiede nella tipologia di analisi che

abbiamo condotto e nella natura dei dati che abbiamo voluto analizzare. Per quanto concerne la prima motivazione, le interrogazioni effettuate al nostro database hanno necessitato, per lo più, della valutazione attenta e accurata dei campi di ogni singola entità del dataset anziché del riconoscimento e della valutazione delle relazioni tra le differenti entità. Il nostro obiettivo è stato il recupero di informazioni utili sulla base di Preprocessing di dati, ricerche per campi, intervalli e regular expression. Di conseguenza, si è ritenuto opportuno sfruttare i vantaggi di un database document-oriented adatto a queste tipologie di elaborazioni, tra i quali ci è stato di grande aiuto il vantaggio di poter effettuare queries basate sui campi del documento. La seconda motivazione riguarda la natura dei dati che abbiamo voluto analizzare. In particolare, i due dataset racchiudono dati rispettivamente associati alle attività e ai suggerimenti scritti dagli utenti. L'unica relazione che connette i due dataset riguarda il fatto che ogni Tip sia riferito ad una particolare attività. Questa relazione sebbene sia stata sfruttata in una delle query in cui è stata realizzata l'operazione di join rappresenta una minima parte del lavoro finale che si compone di più pipeline. Il lavoro di elaborazione e di ricerca sui singoli dataset è stato di gran lunga più corposo e complesso rispetto al lavoro effettuato per poter mettere in relazione le due entità, attività e recensioni. Un possibile riscontro di pattern all'interno di un'eventuale struttura a grafo caratterizzato da archi e nodi sarebbe stato non consono al nostro obiettivo per cui la scelta è ricaduta sull'utilizzo di Mongo DB.

Constatato che un database aggregate-oriented presenta un modello dati molto differente da quello di un database graph-oriented, generalmente c'è anche un'altra caratteristica da tenere in considerazione. Un database orientati agli aggregati è più probabile che venga eseguito in maniera distribuita tra i cluster, mentre un Graph database è più probabile che venga eseguito su un singolo server. Ciò avviene a causa delle innumerevoli relazioni esistenti tra le diverse entità che rendono più complessa l'operazione di distribuzione dell'informazione. Conseguentemente potremmo dedurre che l'utilizzo di un database document-oriented ci consentirebbe di sfruttare a pieno il principio di scalabilità orizzontale in caso ci fosse il bisogno migliorare le prestazioni in termini di capacità o velocità.

# Capitolo 4

# Analysis and Processing

## 4.1 Use of Jupyter Lab for MongoDB

L'accesso a Mongo DB è stato effettuato sia tramite *Mongo DB Compass*, strumento interattivo per eseguire query, ottimizzare e analizzare i dati tramite pipeline, sia tramite *JupyterLab*, ambiente di sviluppo interattivo basato sul Web per notebook, codice e dati. L'utilizzo di JupyterLab ci ha consentito di sfruttare le potenzialità del linguaggio di programmazione Python e di tutte le sue librerie. La sua interfaccia flessibile ha reso possibile implementare in maniera efficiente tutti i passaggi utili per la realizzazione dei requisiti dell'Homework: la connessione al database, il caricamento dei dataset yelp_academic_dataset_business e yelp_academic_dataset_tip in formato *json* e la processazione dei dati tramite query.

Il **prerequisito** necessario per operare tramite JupyterLab è stato installare il package pymongo (*pip install pymongo*) dal terminale. PyMongo è una distribuzione Python contenente strumenti per lavorare con MongoDB da Python.

Il **primo** passo è stato importare i file contenenti i dati di nostro interesse da una cartella salvata in locale ad un vettore di elementi con estensione json.

```
import json
path = "C:/Users/Francesco/Desktop/Università/03 Magistrale/2° anno - 2° semestre/Big
     Data Engineering/Homework/Materiale/"
business = []
tip = []
with open(path + 'yelp_academic_dataset_business.json', 'rb') as f:
    for jsonObj in f:
        business_dict = json.loads(jsonObj)
        business.append(business_dict)
with open(path + 'yelp_academic_dataset_tip.json', 'rb') as f:
    for jsonObj in f:
        tip_dict = json.loads(jsonObj)
        tip.append(tip_dict)
```

Il **secondo** passo è stato instaurare una connessione dalla API offerta da Jupyter verso il nostro database interno ad un cluster "condiviso" chiamato *Cluster0*. Di fatti successivamente alla registrazione necessaria per poter usufruire dei servizi forniti da Mongo, abbiamo creato il cluster in maniera gratuita tramite *Mongo DB Atlas*, un servizio *MongoDB* ospitato nel cloud da piattaforme come AWS, Azure e Google Cloud. Sebbene la versione gratuita *Shared Cluster* fosse soggetta a limitazioni, è stata comunque sufficiente per le nostre esigenze.

```
import pymongo
# Connection
client = pymongo.MongoClient("mongodb+srv://francisfat:****@cluster0.c8bwx.mongodb.
    net/yelp?retryWrites=true&w=majority")
```

Il **terzo** passo è stata la creazione del database, chiamato *Yelp*, e la creazione delle collezioni business e tip inserendo in quest'ultime tutti i documenti che avevo precedentemente importato da locale. Una precisazione da fare è che i database e le collezioni vengono create solo quando i dati vengono caricati in essi.

```
# Select the db by its name
db = client['yelp']

# Created or Switched to collection
collection = db['tips']
res = collection.insert_many(tip)
collection = db['business']
res = collection.insert_many(business)
# bulk insertion of the data
res = collection.insert_many(tip)
```

Per quanto concerne l'elaborazione delle *Aggregations* abbiamo sfruttato Mongo DB Compass, la GUI per Mongo DB in versione desktop, nonostante Jupyter offrisse gli strumenti adeguati per farlo. La sua interfaccia user-friendly ed intuitiva ha reso molto facile creare delle pipeline che se avessimo dovuto scrivere in codice sarebbe stato decisamente più complesso.

## 4.2 Query and Final Task

### 4.2.1 Query About Best Restaurant in South USA with some Specific Features

La query uno lavora su documenti di tipo business e presenta come output ristoranti aperti, situati nel sud degli Stati Uniti, con più di quattro stelle, con un numero di review maggiore a mille, che accettino Carte di Credito, che forniscano free WiFi e che presentino un parcheggio per biciclette. Il filtraggio è stato ottenuto mediante $match. Per ottenere i soli ristoranti del sud abbiamo imposto un valore di latitudine uguale o inferiore a 37,7576793°, la latitudine

di San Francisco. L'operatore $regex, utilizzato nello stage di filtraggio, è stato fondamentale nel ricercare la parola chiave "restaurant" nel campo "categories" di ogni documento, seguito dalla lettera "i" che ci ha consentito di rendere la ricerca insensibile al maiuscolo/minuscolo. Nello stage di proiezione $project abbiamo usufruito dell'operatore $multiply per moltiplicare numero di feedback e numero di stelle di ogni locale ottenendo così l'affidabilità del locale. Infine, abbiamo ordinato la lista dei risultati per affidabilità mediante uno stage $sort.

```
[{
    $match: {
        stars: {$gt: 4},
        latitude: { $lte: 37.7576793 },
        is_open: 1,
        'attributes.BusinessAcceptsCreditCards': 'True',
        'attributes.WiFi': 'u\'free\'',
        'attributes.BikeParking': 'True',
        categories: {  $regex: RegExp('restaurant', i) },
        review_count: { $gt: 1000 }
    }
}, {
    $project: {
        _id: 1,
        name: 1,
        stars: 1,
        address: 1,
        city: 1,
        categories: 1,
        Reliability: { $multiply: [ '$stars','$review_count']}
    }
}, {
    $sort: {
        Reliability: -1
    }
}]
```

Task. La creazione di un campo chiamato affidabilità discende da una considerazione semplice ma non banale: potrebbero esistere locali con un numero di stelle molto alto ma con un numero di feedback estremamente basso, e viceversa, potrebbero esistere locali con un numero di feedback molto alto ma con un numero di stelle molto basso. L'affidabilità di un locale non può essere associata esclusivamente al numero di stelle o al numero di feedback. Essa presenta una dipendenza diretta da entrambi i fattori. Questo è il motivo per il quale abbiamo stabilito il criterio per individuare fra tutti i ristoranti di qualità medio alta (4 stelle) quelle che avessero anche un numero elevato di recensioni. Quindi attraverso il prodotto dei due valori abbiamo ottenuto l'affidabilità. Sicuramente a parità di stelle il fatto di avere un numero di recensioni elevato è indice dell'affidabilità del dato sulle stelle.

Come ultimo passo abbiamo creato una View di quanto ottenuto. Il seguente è il primo dei risultati.

```
_id: ObjectId('626027314e0deaba7ec0cd5c')
name: "Prep & Pastry"
address: "2660 N Campbell Ave"
city: "Tucson"
stars: 4.5
categories: "Restaurants, Cocktail Bars, Bars, Comfort Food, American (New), Sandwi…"
Reliability: 9567
```

```
_id: ObjectId('626027314e0deaba7ec0cbc3')
name: "District Donuts Sliders Brew"
address: "2209 Magazine St"
city: "New Orleans"
stars: 4.5
categories: "Food, Donuts, Burgers, American (Traditional), Coffee & Tea, Restauran…"
Reliability: 9279
```

Figura 4.1: View

### 4.2.2 Query About Cities With More Sushi Restaurant

La query due lavora sui documenti di tipo business e permette di consultare le città con il maggior numero di ristoranti di sushi. In seguito al filtraggio con lo stage $match utilizzato per ricercare esclusivamente i ristoranti sushi, è stato necessario raggruppare tutti i ristoranti per città mediante lo stage $group utilizzando come ID di raggruppamento il nome delle città. In tale fase è stato realizzato sia il conteggio dei ristoranti tramite l'operatore aritmetico $count che la media delle stelle dei ristoranti per ogni città tramite $avg. Il passo successivo di ci è stato utile per arrotondare, tramite $round, la media delle stelle ottenuta nel passo precedente. Infine, abbiamo riordinato le città per numero di ristoranti di sushi in senso decrescente in modo da avere come primi elementi le città con il maggior numero di ristoranti di sushi.

```
1 [{
2     $match: {
3         categories: { $regex: RegExp('sushi', i)}
4     }
5 }, {
6     $group: {
7         _id: '$city',
```

```
        State: {$first: '$state'},
        'Total Sushi Restaurants': {$count: {}},
        'Average Star Restaurants': { $avg: '$stars'}
    }
}, {
    $project: {
        _id: 1,
        State: 1,
        'Total Sushi Restaurants': 1,
        'Rounded Average Star Restaurants': { $round: ['$Average Star Restaurants',1]
     }
    }
}, {
    $sort: {
        'Total Sushi Restaurants': -1
    }
}]
```

Task. Dai risultati appena ottenuti potremmo dedurre che in città come Philadelphia, Tampa e Indianapolis il Sushi è diventato particolarmente popolare per cui potremmo considerare queste città più propense a nuove esperienze culinarie e quindi, in termini generali, ad una contaminazione culturale. Questo potrebbe essere un fattore importante per nuove iniziative di bussiness.

Come ultimo passo abbiamo creato una View di quanto ottenuto.

```
_id: "Philadelphia"
State: "PA"
Total Sushi Restaurants: 185
Rounded Average Star Restaurants: 3.7
```

```
_id: "Tampa"
State: "FL"
Total Sushi Restaurants: 118
Rounded Average Star Restaurants: 3.8
```

```
_id: "Nashville"
State: "TN"
Total Sushi Restaurants: 86
Rounded Average Star Restaurants: 3.7
```

Figura 4.2: View

### 4.2.3 Query About User Activity Over The Years

La query tre lavora sui documenti di tipo tip e consente di monitorare l'attività degli utenti nel corso degli anni. Per attività degli utenti nel tempo si intende il numero di feedback pervenuti

in una determinata finestra temporale. Per fare ciò è stata necessario un'operazione di Preprocessing del campo "date" poiché quest'ultimo era stato memorizzato come una stringa. Ciò di cui necessitavamo era splittare il campo "date" in due diversi campi data, ora e successivamente suddividere il campo data in anno, mese e giorno. Abbiamo realizzato quest'operazione di split in [data, ora] e split in [anno, mese, giorno] rispettivamente tramite due stage ed altri due stage consecutivi di tipo *projecte*set.

```
[{
    $project: {
        timestamp: { $split: [ '$date',' ' ] }
    }
}, {
    $set: {
        date: { $arrayElemAt: ['$timestamp', 0 ] },
        hour: { $arrayElemAt: ['$timestamp', 1 ] }
    }
}, {
    $project: {
        date: { $split: ['$date','-'] }
}, {
    $set: {
        year:  { $arrayElemAt: ['$date', 0 ] },
        month: { $arrayElemAt: ['$date', 1] },
        day:   { $arrayElemAt: ['$date', 2] }
    }
}]
```

In seguito, è bastato raggruppare i documenti per anno, conteggiarli e riordinarli in senso decrescente.

```
[{
    $group: {
        _id: '$year',
        'Number of Tips': { $count: {} }
    }
}, {
    $sort: {
        'Number of Tips': -1
    }
}]
```

Task. In questo modo si è voluto dimostrare che il picco di recensioni da parte degli utenti è avvenuto tra il 2011 ed il 2014 evidenziando lo sviluppo tecnologico che si è avuto in quegli anni e di conseguenza la crescita della popolarità di Yelp. Mentre si può notare una decrescita del numero di recensioni pubblicate sulla piattaforma Yelp dal 2014 in poi, dovuta probabilmente allo sviluppo di altre piattaforme più utilizzate. I risultati mostrano la

massima attività nell'anno 2012. In aggiunta, abbiamo inserito una semplice visualizzazione di quanto ottenuto dalla query. Questo potrebbe essere un ottimo motivo per rinnovare la piattaforma Yelp.

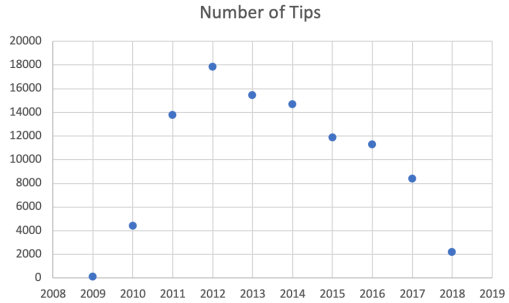


Figura 4.3: Number of Tips for Year

Come ultimo passo abbiamo creato una View di quanto ottenuto.

```
_id: "2012"
Number of Tips: 110459

_id: "2014"
Number of Tips: 109160

_id: "2013"
Number of Tips: 107563
```

Figura 4.4: View

### 4.2.4 Query About Dirtiest Places For Users

La query quattro lavora sia sui documenti di tipo business che di tipo tip e consente di riconoscere i posti più sporchi e squallidi registrati su Yelp usando come metro di giudizio i pareri degli utenti. Per recuperare le opinioni negative degli utenti è stata implementata un'operazione di Light Sentimental Analysis sulle recensioni contenute nel dataset tip. Sono state ricercate parole specifiche all'interno del testo scritto da ogni utente.

```
[{
    $match: {
        $or: [
            { text: RegExp('dirly') },
            { text: RegExp('bad smell') },
            { text: RegExp('dingy') },
            { text: RegExp('filthy') },
            { text: RegExp('nasty') }
```

```
        ]
    }
}]
```

Le informazioni sui luoghi memorizzate nel dataset business sono state recuperate mediante una operazione di join effettuata mediante lo stage $lookup. Quest'operatore ha associato ad ogni recensione le informazioni del locale a cui la recensione faceva riferimento sfruttando l'unicità del campo business_id contenuto in ogni documento di tipo tip.

```
[{
    $lookup: {
        from: 'business',
        localField: 'business_id',
        foreignField: 'business_id',
        as: 'business'
    }
}]
```

Tramite lo stage $replaceRoot abbiamo accorpato ai campi del documento tip i campi del documento business precedentemente esistenti al livello inferiore, sotto forma di campi di un oggetto. Successivamente, la proiezione ha rimosso i campi non di nostro interesse velocizzando le successive fasi. Inoltre, essendo la pipeline lunga e complessa, abbiamo semplificato il lavoro computazionale limitando le elaborazioni successive ad un numero di documenti inferiore tramite lo stage $limit. Infine, abbiamo raggruppato per business_id ed ordinato per numero di feedback negativi degli utenti.

```
[{
    $replaceRoot: { newRoot: { $mergeObjects: [ { $arrayElemAt: [ '$business', 0 ]},'
    $$ROOT' ] }}
}, {
    $project: {
        business_id: 1,
        name: 1,
        city: 1
    }
}, {
    $limit:
        500
}, {
    $group: {
        _id: '$business_id',
        'Place Name': { $first: '$name' },
        'Number of Disgusted Users': { $count: {} }
    }
}, {
    $sort: {
```

```
        'Number of Disgusted Users': -1
    }
}]
```

Task. La query ha dei riscontri pratici in quanto potrebbe consigliare agli utenti quali luoghi evitare a causa di odori sgradevoli, mancate pulizie e altro. Inoltre, questa tipologia di query racchiude diverse tecniche fondamentali per la gestione dei dati, tra cui Join e Sentimental Analysis. Un'osservazione da fare è che la Sentimental Analysis è stata realizzata a titolo esemplificativo. Una Sentimental Analysis più efficace dovrebbe essere più ricca di vocaboli da ricercare ed essere attenta ad evitare eventuali ambiguità, ad esempio il termine "dirty" potrebbe comparire nella frase "The place is not dirty" creando, così, un risultato equivoco. Come ultimo passo abbiamo creato una View di quanto ottenuto.


Figura 4.5: View

Una volta terminata la procedura e creata una view per ogni query otteniamo ciò che segue:

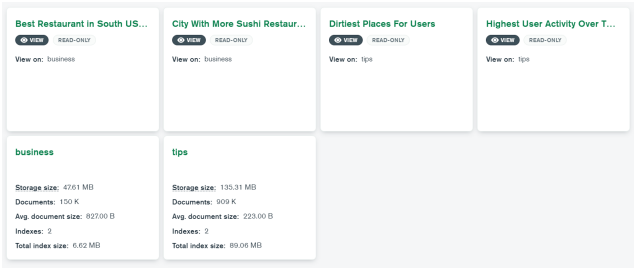


Figura 4.6: Views

## 4.3 Use of Indexing

Si è deciso di implementare aggregati ed indici: due elementi importanti per migliorare l'efficienza d'utilizzo del nostro database. Le operazioni di aggregazione elaborano i record di dati e

restituiscono i risultati calcolati, esse raggruppano i valori di più documenti e possono eseguire una serie di operazioni sui dati raggruppati per restituire poi un unico risultato. MongoDB offre tre modi per eseguire l'aggregazione: la pipeline di aggregazione, il metodo MapReduce, metodi di aggregazione a scopo singolo. Le aggregazioni sono state effettuate utilizzando la prima metodologia grazie alla semplicità d'utilizzo offerta dalla GUI di MongoDb Compass: i documenti entrano in una pipeline a più fasi che trasforma i documenti in un risultato aggregato.

Gli indici supportano l'esecuzione efficiente delle query in MongoDB senza i quali, dovrebbe eseguire una scansione della raccolta, ovvero esaminare ogni documento presente in una collezione, per selezionare quei documenti che corrispondono all'istruzione della query. Se esiste un indice appropriato per una query, MongoDB può utilizzare l'indice per limitare il numero di documenti che deve ispezionare. Gli indici, sono strutture di dati speciali che memorizzano una piccola parte del set di dati organizzati in modo da poter effettuare ricerche nella collection in maniera più semplificata ma soprattutto più efficiente: memorizza i riferimenti ad un campo specifico o ad un insieme di campi, ordinato in base al valore del campo. Per dimostrare l'efficacia degli indici implementati, sono riportati degli esempi di query dimostrativi prima e dopo la creazione degli stessi. In particolare, per verificare le performance di una query effettuata su una determinata collezione, è possibile utilizzare la sezione denominata "Explain Plain" nel quale vengono riportati dettagli riguardanti il numero di documenti restituiti, il numero di indici scansionati (opzionale, perché se non esiste un indice verrà restituito 0), il numero totale di documenti scansionati ed il tempo impiegato per restituire il risultato.

### 4.3.1 Example

A titolo esemplificativo abbiamo riportato l'utilizzo dell'applicazione di un indice ad una query molto semplice. L'utilizzo di un indice, per dataset molto ampi, può essere molto utile per migliorare le performance in termini di tempo di risposta non solo della ricerca ma anche dell'inserimento, dell'aggiornamento o della cancellazione su documenti di una collection. Il motivo di tale Speed Up risiede nel fatto che un indice è una collezione di riferimenti ai documenti ordinati sulla base dei campi che specifichiamo durante la sua creazione. La velocità di query incrementa grazie a questa lista di rifermenti appena creata. Sebbene su Mongo DB sia possibile applicare indici testuali, multi-key, o composti, ovvero basati su più campi, in questo caso conveniva utilizzare un indice a campo singolo utilizzato quando la ricerca più frequente in una raccolta si basa su un campo specifico. Il campo di cui stiamo parlando è *city*.

Inoltre, abbiamo riportato le differenze tra le prestazioni di una query senza e con l'utilizzo di un indice. La prima differenza che salta all'occhio è sicuramente il tempo d'esecuzione ridotto almeno della metà. La seconda differenza risiede nel numero di documenti esaminati; il caso


Figura 4.7: Index

senza indice analizza circa 150 mila documenti rispetto ai 14 mila documenti analizzati nel caso di utilizzo di indice.

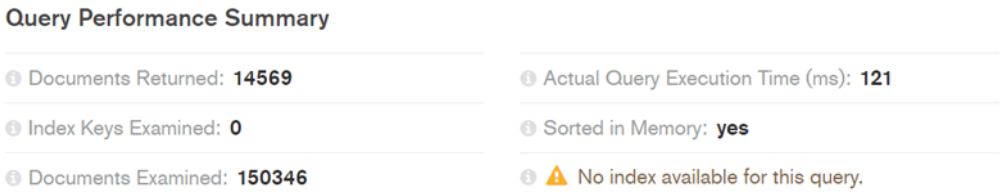


Figura 4.8: Query Performance without index

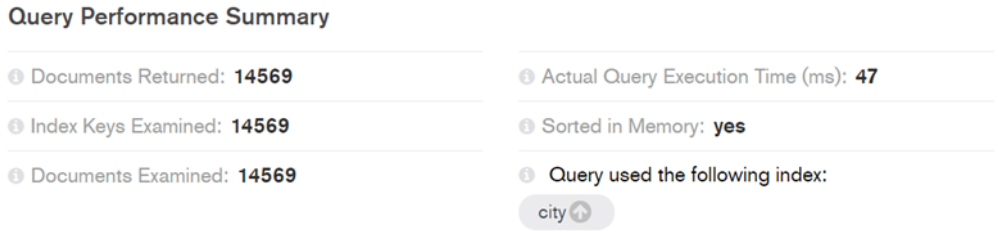


Figura 4.9: Query Performance with index