



Università degli Studi di Salerno



Dipartimento di Ingegneria dell'Informazione ed Elettrica e  
Matematica Applicata

Corso di Laurea Magistrale in Ingegneria Informatica  
Curriculum di Enterprise Software Architectures

## **Data Management Systems**

Project Work

### **Twitter Sentiment Analysis**

Grazia Ferrara **0622701901**

**Anno accademico 2023-2024**

## Sommario

<b>1. Descrizione della realtà di interesse</b>	<b>3</b>
1.1. Analisi della realtà di interesse	3
<b>2. Analisi delle specifiche</b>	<b>5</b>
2.1. Glossario dei termini	5
2.2. Strutturazione dei requisiti in frasi	5
2.2.1. Frasi di carattere generale	5
2.2.2. Frasi relative a Tweet	6
2.2.3. Frasi relative a Trend	6
2.2.4. Frasi relative a Utente	6
2.2.5. Frasi relative a Sentiment Analysis	6
2.2.6. Frasi relative a tipi specifici di Analisi	6
2.3. Identificazione delle operazioni principali	6
<b>3. Progettazione Concettuale</b>	<b>8</b>
3.1. Schema Concettuale	8
3.1.1. Note sullo schema E-R	8
3.2. Dizionario dei Dati	8
3.3. Regole Aziendali	9
<b>4. Progettazione Logica</b>	<b>10</b>
4.1. Analisi dello Schema Concettuale	10
4.1.1. Analisi delle Prestazioni	10
4.2. Scelta degli identificatori principali	10
4.3. Scelta degli indici	11
4.4. Schema logico	11
4.5. Documentazione dello schema logico	12
<b>5. Script Creazione e Popolamento Database</b>	<b>12</b>
5.1. Creazione dello schema	13
5.2. Popolamento del database	16
5.3. Operazioni di inserimento	17
5.4. Operazioni di aggiornamento	20
5.5. Operazioni di rimozione	21
<b>6. Query</b>	<b>26</b>
6.1. Aggregation: Average sentiment per Trend	26
6.2. Aggregation: Sentiment percentages	27
6.3. Aggregation: Trend diffusion degree	29
6.4. Aggregation: User coherence score	31
6.5. Aggregation: User's sentiment percentages	32
6.6. Aggregation: Engagement metrics computation	34
6.7. Aggregation: Discussions' detection	35
6.8. Altre query	36
6.8.1. Utente da più tempo su Twitter	36
6.8.2. Tweet più condivisi	37
6.8.3. Trends più popolari	37
6.8.4. Utenti più popolari	38

6.8.5. Ottieni i tweet per trends	38
6.8.6. Ottieni i tweet per gli utenti	39

# 1. Descrizione della realtà di interesse

Titolo: **Twitter Sentiment Analysis**

*Un tweet è un messaggio di testo avente una lunghezza non superiore a 4000 caratteri, inviato a un sito Internet tramite instant messenger, e-mail o cellulare con lo scopo di comunicare informazioni in tempo reale. I tweet consentono agli utenti di mettere in piedi delle vere e proprie discussioni in merito ad un determinato argomento, legandosi a quest'ultimo per mezzo di un hashtag o semplicemente sulla base di parole ricorrenti. Hashtag e parole ricorrenti concorrono infatti alla creazione di un trend. I trend sono mostrati nella home page di Twitter e sono facilmente accessibili agli utenti in modo tale che possano avere un'idea degli argomenti più in voga del momento non appena effettuano l'accesso. Gli utenti possono vedere i tweet di altri utenti, se pubblici o se presenti nella loro rete di amicizie, mettere like e condividerli nella propria bacheca ("re-postarli" in gergo). Utilizzando i tweet che sono legati ad un determinato topic, si può effettuare la sentiment analysis e vedere qual è l'opinione espressa dalle persone tramite i tweet su un determinato argomento (positiva, negativa, neutra), utilizzando qualche tool che, preso in input il testo, restituisce il sentiment. È di interesse lo studio del sentiment in vari ambiti applicativi.*

*Si intende progettare e realizzare una base di dati non relazionale (non avendo i dati a disposizione uno schema fisso e ben definito) per la memorizzazione delle informazioni necessarie ad effettuare la sentiment analysis sui tweet legati a un certo topic, considerando i seguenti elementi:*

- 1. Con riferimento ai tweet sono essenziali le seguenti informazioni (basilari ma non esaustive): username dell'utente che ha pubblicato il tweet, testo del tweet, topic del tweet, il sentiment ad esso associato.*
- 2. Con riferimento agli utenti sono essenziali le seguenti informazioni (basilari ma non esaustive): username dell'utente, numero di persone seguite, numero di persone che lo seguono e se è verificato.*
- 3. Con riferimento ai trend sono essenziali le seguenti informazioni (basilari ma non esaustive): nome del trend, numero di post pubblicati per quel trend e il luogo e la data in cui esso è andato virale.*
- 4. È di interesse conoscere il sentiment delle persone per un dato trend.*
- 5. È di interesse misurare l'impatto sociale di un trend in termini di persone presumibilmente raggiunte all'interno della rete sociale di Twitter.*
- 6. È di interesse misurare il grado di affidabilità dell'utente nell'espressione di un certo sentiment per un dato trend.*
- 7. È di interesse studiare la sentiment history di un utente.*
- 8. È di interesse calcolare le metriche di engagement.*

## 1.1. Analisi della realtà di interesse

Dalla descrizione fornita, è possibile identificare una serie di elementi chiave relativi alla realtà di interesse:

- 1. Tweet:** Sono brevi messaggi di testo con una lunghezza massima di 4000 caratteri inviati attraverso diverse piattaforme, come instant messenger, e-mail o dispositivi mobili. Questi messaggi servono a comunicare informazioni in tempo reale.

2. **Trend:** Sono utilizzati per collegare i tweet a determinati argomenti e contribuiscono alla creazione dei trend. I trend rappresentano gli argomenti più popolari e sono facilmente accessibili agli utenti nella home page di Twitter.
3. **Sentiment analysis:** Viene utilizzata per valutare l'opinione espressa nelle discussioni online relative a un determinato argomento. Questa analisi può determinare se il sentimento è positivo, negativo, neutro o altro, attraverso l'uso di strumenti specifici.
4. **Informazioni chiave necessarie:**
  - Per i tweet: username dell'utente, testo del tweet, topic del tweet, sentiment associato.
  - Per gli utenti: username, numero di seguaci, numero di persone seguite, verifica del profilo.
  - Per i trend: nome del trend, numero di post associati, luogo e data di diffusione.
5. **Obiettivi specifici:**
  - Conoscere il numero di persone che esprimono un determinato tipo di sentiment su un topic.
  - Misurare l'impatto di un tweet con un dato sentiment sulla rete, in base alla rilevanza sociale dell'utente che l'ha pubblicato.
  - Identificare utenti che hanno pubblicato più tweet su un topic con sentiment discordanti che potrebbero influire negativamente sull'analisi.
  - Studiare come evolve l'espressione del sentiment di un utente nel tempo.
  - Ottenere le metriche di engagement: condivisioni, citazioni, like, ...

In sintesi, il progetto mira a condurre un'analisi approfondita dei sentiment legati ai trending topics, considerando la rilevanza sociale degli utenti, la coerenza dei sentiment e l'effettivo impatto delle discussioni online. Questi elementi contribuiranno a una comprensione più completa di come le discussioni influenzino l'opinione pubblica e saranno fondamentali per valutare l'efficacia delle campagne di comunicazione e il coinvolgimento degli utenti nelle discussioni online.

## 2. Analisi delle specifiche

### 2.1. Glossario dei termini

	Termine	Descrizione	Sinonimi	Collegamenti
1	Tweet	Messaggio o post breve pubblicato su Twitter, contenente testo, immagini o video.	Post, messaggio, aggiornamento di stato	Utente, Trend
2	Trend	Temi o discussioni più rilevanti e attuali all'interno della comunità online in un dato momento.	Tendenza, moda, argomento popolare	Tweet
3	Utente	Individuo che utilizza una piattaforma di social media, come Twitter. Può pubblicare tweet, seguire altri utenti ed essere seguito da altri.	Account, profilo, persona registrata	Tweet
4	Sentiment	Opinione associata a un determinato argomento, tweet o discussione online.	Opinione	Tweet, Utente
5	Sentiment Analysis	Processo di analisi del testo mirato a determinare l'opinione associata a un contenuto online.	Analisi dell'opinione	Tweet, Utente, Trend
6	Social influence	Effetto che un utente o un contenuto online ha sulla sua cerchia di seguaci e sull'opinione pubblica.	Potere di persuasione	Utente, Sentiment Analysis
7	Agreement	Misura in cui gli utenti online esprimono sentimenti simili o discordanti su un determinato argomento.	Concordanza, accordo, consenso, armonia	Sentiment analysis, Utente

Tabella 1. Glossario dei Termini

### 2.2. Strutturazione dei requisiti in frasi

#### 2.2.1. Frasi di carattere generale

*Si intende progettare e realizzare una base di dati non relazionale (non avendo i dati a disposizione uno schema fisso e ben definito) per la memorizzazione delle informazioni necessarie ad effettuare la sentiment analysis sui tweet legati a un certo topic.*

#### 2.2.2. Frasi relative a Tweet

*Un tweet è un messaggio di testo avente una lunghezza non superiore a 4000 caratteri, inviato a un sito Internet tramite instant messenger, e-mail o cellulare con lo scopo di comunicare*

informazioni in tempo reale. I tweet consentono agli utenti di mettere in piedi delle vere e proprie discussioni in merito ad un determinato argomento, legandosi a quest'ultimo per mezzo di un hashtag o semplicemente sulla base di parole ricorrenti.

Con riferimento ai tweet sono essenziali le seguenti informazioni (basilari ma non esaustive): username dell'utente che ha pubblicato il tweet, testo del tweet, topic del tweet, , il sentiment ad esso associato.

### **2.2.3. Frasi relative a Trend**

Hashtag e parole ricorrenti concorrono infatti alla creazione di un trend. I trend sono mostrati nella home page di Twitter e sono facilmente accessibili agli utenti in modo tale che possano avere un'idea degli argomenti più in voga del momento non appena effettuano l'accesso.

Con riferimento ai trend sono essenziali le seguenti informazioni (basilari ma non esaustive): nome del trend, numero di post pubblicati per quel trend e il luogo e la data in cui esso è andato virale.

### **2.2.4. Frasi relative a Utente**

Gli utenti possono vedere i tweet di altri utenti, se pubblici o se presenti nella loro rete di amicizie, mettere like e condividerli nella propria bacheca ("re-postarli" in gergo).

Con riferimento agli utenti sono essenziali le seguenti informazioni (basilari ma non esaustive): username dell'utente, numero di persone seguite, numero di persone che lo seguono e se è verificato.

### **2.2.5. Frasi relative a Sentiment Analysis**

È di interesse lo studio del sentiment in vari ambiti applicativi. Utilizzando i tweet che sono legati ad un determinato topic, si può effettuare la sentiment analysis e vedere qual è l'opinione espressa dalle persone tramite i tweet su un determinato argomento (positiva, negativa, neutra), utilizzando qualche tool che, preso in input il testo, restituisce il sentiment.

### **2.2.6. Frasi relative a tipi specifici di Analisi**

È di interesse conoscere il sentiment delle persone per un dato trend.

È di interesse misurare l'impatto sociale di un trend in termini di persone presumibilmente raggiunte all'interno della rete sociale di Twitter.

È di interesse misurare il grado di affidabilità dell'utente nell'espressione di un certo sentiment per un dato trend.

È di interesse studiare la sentiment history di un utente.

È di interesse calcolare le metriche di engagement.

## **2.3. Identificazione delle operazioni principali**

Di seguito vengono riportate solo le operazioni più complesse

**Operazione 1:** per ogni trend, selezionare tutti i tweet associati al trend e mostrare il sentiment ottenuto come media dei sentiment dei tweet selezionati (20 volte al giorno)

**Operazione 2:** per ogni trend, selezionare tutti i tweet che vi afferiscono e, per ogni categoria di sentiment che il tweet può assumere, stampare la percentuale di tweet che hanno ottenuto quel determinato sentiment (20 volte al giorno)

**Operazione 3:** dato un certo trend, identificare tutti gli utenti che hanno pubblicato tweet che vi afferiscono e sulla base del loro numero di followers identificare quante persone sono state all'incirca raggiunte dal trend per ognuno di quei tweet, come somma del numero di followers (20 volte al giorno)

**Operazione 4:** *per ogni utente, raggruppare i tweet che ha scritto per topic e, per ogni cluster, assegnare all'utente un punteggio di coerenza, fare la media dei punteggi ottenuti (1 volta a settimana)*

**Operazione 5:** *dato un utente, prendi i tweet che ha scritto e calcola le percentuali di tweet con sentiment positivo, negativo e neutro (1 volta a settimana)*

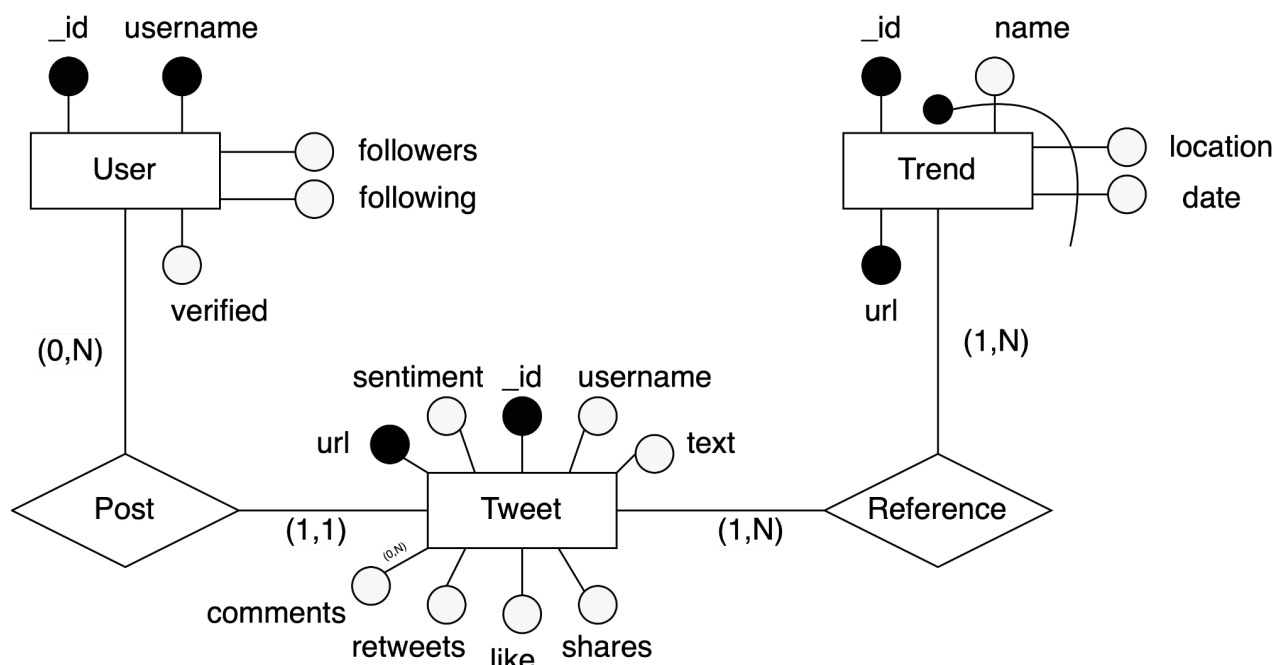
**Operazione 6:** *per ogni trend calcolare il numero medio di like, condivisioni, retweet e commenti che i suoi post hanno avuto (1 volta al mese)*

**Operazione 7:** *dato un trend, per ogni tweet ad esso associato, verificare se i suoi commenti hanno dato origine a una discussione identificando eventuali sentiment discordanti (600 volte al mese)*



### 3. Progettazione Concettuale

#### 3.1. Schema Concettuale



##### 3.1.1. Note sullo schema E-R

Per la progettazione è stata utilizzata una strategia di tipo top-down. In particolare si sono prima individuate le entità e le associazioni e poi sono stati aggiunti gli attributi. Lo schema ER qui riportato non è esaustivo (ad esempio per le entità non sono stati inseriti tutti i possibili attributi), proprio perché essendo in un contesto web ed avendo a che fare con una base di dati non relazionale, non si ha un insieme finito e limitato di dati di interesse.

#### 3.2. Dizionario dei Dati

Entità	Descrizione	Attributi	Identificatore
Tweet	Messaggio o post breve pubblicato su Twitter, contenente testo, immagini o video.	<u>_id</u> , <u>url</u> , <u>username</u> , <u>text</u> , <u>comments</u> (multivalore e opzionale), <u>retweets</u> , <u>like</u> , <u>shares</u> , <u>sentiment</u>	<u>_id</u> , <u>url</u>
Trend	Tema rilevante in una comunità in un determinato momento.	<u>_id</u> , <u>url</u> , <u>name</u> , <u>location</u> , <u>date</u>	<u>_id</u> , <u>url</u> , ( <u>name</u> , <u>location</u> , <u>date</u> )
User	Utente social che ha la facoltà di pubblicare tweet ed	<u>_id</u> , <u>username</u> , <u>followers</u> , <u>following</u> , <u>verified</u>	<u>_id</u> , <u>username</u>

	interagire con i post altrui.		
--	-------------------------------	--	--

Tabella 2. Dizionario dei dati – Entità

Relazioni	Descrizione	Entità Coinvolte	Attributi
Post	Pubblicazione di un tweet da parte di un utente.	User, Tweet	-
Reference	Appartenenza dell'argomento di un tweet ad un trend.	Trend, Tweet	-

Tabella 3. Dizionario dei dati - Relazioni

### 3.3. Regole Aziendali

Regole di Vincolo
<p><b>(RV1)</b> L'attributo <i>verified</i> di <i>User</i> può assumere i valori <i>true</i> o <i>false</i>.</p> <p><b>(RV2)</b> L'attributo <i>following</i> di <i>User</i> deve essere sempre maggiore o uguale di zero.</p> <p><b>(RV3)</b> L'attributo <i>followers</i> di <i>User</i> deve essere sempre maggiore o uguale di zero.</p> <p><b>(RV4)</b> L'attributo <i>date</i> di <i>Trend</i> deve avere una data non successiva alla data odierna.</p>

Tabella 4. Regole di vincolo

## 4. Progettazione Logica

### 4.1. Analisi dello Schema Concettuale

#### 4.1.1. Analisi delle Prestazioni

##### 4.1.1.1. *Tavola dei volumi*

Concetto	Tipo	Volume
Trends	E	100
Tweets	E	10000
Users	E	3000
Post	R	10000
Reference	R	10000

Tabella 6. Tavola dei volumi

##### 4.1.1.2. *Tavola delle operazioni*

Operazione	Tipo	Frequenza
Operazione 1: <i>Sentiment medio di un trend</i>	B	20 al giorno
Operazione 2: <i>Sentiment percentuali di un trend</i>	B	20 al giorno
Operazione 3: <i>Grado di diffusione di un trend</i>	I	20 al giorno
Operazione 4: <i>Grado di affidabilità di un utente su un trend</i>	B	1 volta alla settimana
Operazione 5: <i>Sentiment percentuali di un utente</i>	I	1 volta alla settimana
Operazione 6: <i>Calcolo delle metriche di engagement</i>	B	1 volta al mese
Operazione 7: <i>Rilevazione di discussioni</i>	I	600 volte al mese

Tabella 7. Tavola delle operazioni

### 4.2. Scelta degli identificatori principali

Per identificare il Tweet si è scelto di utilizzare, tra gli attributi presenti, il suo url, e lo stesso vale per il Trend. Il Trend però, può anche essere identificato tramite il suo nome, la data ed il luogo in cui si è diffuso. L'utente può essere invece identificato univocamente tramite il suo username. Considerando la base di dati che verrà utilizzata successivamente, possiamo aspettarci però che ognuno di essi abbia un `_id` univoco (ObjectId), che verrà usato per semplicità come identificatore principale in ciascuno dei casi.

### 4.3. Scelta degli indici

Gli indici in MongoDB supportano l'esecuzione di query in maniera efficiente, in modo da evitare di scansionare tutti i documenti di un database per restituire dei risultati. Dal momento in cui sono anche costosi perché devono essere aggiornati ad ogni operazione di scrittura, è bene metterli solo su campi che vengono usati molto di frequente per effettuare delle ricerche. Si è pensato che sarebbe una buona idea quella di indicizzare gli **username** degli utenti e creare un ulteriore indice, stavolta *composto*, su **data**, **luogo** e **nome** di un trend.

### 4.4. Schema logico

Poiché in MongoDB le operazioni di write vengono fatte in maniera atomica, per la rappresentazione dei commenti dei tweet, si è deciso di usare un modello di dati denormalizzato, attraverso **documenti embedded**. Come suggerisce la documentazione di MongoDB, questo tipo di rappresentazione è adatta a rappresentare relazioni di 'contains' e in questo modo, usando un'unica operazione atomica, è possibile effettuare l'aggiornamento sia di un attributo del tweet, che dei tweet presenti nell'array **comments** dei commenti. Ad esempio, è possibile rimuovere un tweet dai commenti ad aggiornare il numero delle risposte nel tweet principale con un'unica operazione di write. Inoltre, tale disposizione dei documenti torna utile per velocizzare le operazioni di lettura quando dal tweet 'parent' si ha bisogno di accedere ai tweet 'children'.

Per modellare la relazione uno a molti che sussiste tra User e Tweet, si è deciso di usare un modello di dati normalizzato, usando dei **references** tra i documenti. Tale scelta è dovuta al fatto che la disposizione embedded dell'utente all'interno del tweet avrebbe provocato una forte duplicazione dei dati dello stesso, dal momento in cui è plausibile che l'utente possa scrivere più tweet per un determinato topic, oppure scrivere un tweet e rispondere ai commenti sotto il suo post. Pertanto i documenti della collezione User, avranno al loro interno un array di reference **tweets** ai documenti della collezione Tweet ed i documenti della collezione Tweet avranno un campo **user\_id** che conterrà il reference al documento che rappresenta l'utente che ha scritto quel tweet nella collezione User.

Per modellare la relazione molti a molti che sussiste tra Trend e Tweet, si è deciso di usare un modello di dati normalizzato, usando dei **references** tra i documenti. Trattandosi di una relazione di questo tipo, non è conveniente usare la rappresentazione tramite documenti embedded che provocherebbe una ingente duplicazione di tutti i dati. Pertanto i documenti della collezione Trend avranno un array di riferimenti **tweets** ai Tweet che sono stati pubblicati in merito a quello specifico trend, mentre i documenti della collezione Tweet avranno un array di riferimenti **trends** ai trend a cui afferiscono.

---

Dunque

- i **Tweet** saranno modellati nel seguente modo

```
{
  _id: <ObjectId>,
  url: <String>,
  username: <String>,
```

```

    user_id : <ObjectId>,
    text: <String>,
    comments: [{tweetDocument},...,{tweetDocument}],
    retweets: <Int32>,
    like: <Int32>,
    shares: <Int32>,
    sentiment: <Double>,
    trends: [<ObjectId>,...,<ObjectId>]
  }

```

- i **Trend** saranno modellati nel seguente modo

```

{
  _id: <ObjectId>,
  url: <String>,
  name: <String>,
  location: <String>,
  date: <Timestamp>,
  tweets: [<ObjectId>,...,<ObjectId>]
}

```

- gli **User** saranno modellati nel seguente modo

```

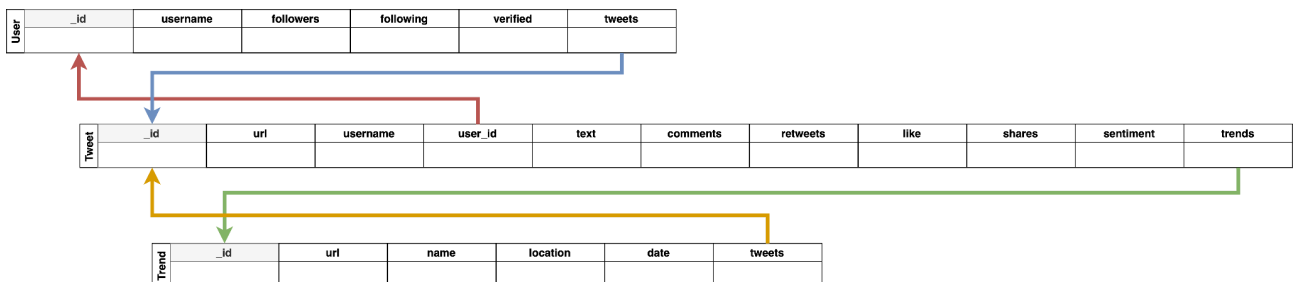
{
  _id: <ObjectId>,
  username: <String>,
  followers: <Int32>,
  following: <Int32>,
  verified: <Boolean>,
  tweets: [<ObjectId>,...,<ObjectId>]
}

```

Eventuali ulteriori attributi nei documenti sono contemplati per la natura stessa della base dati.

## 4.5. Documentazione dello schema logico

Di seguito si riporta una rappresentazione grafica dello schema logico.



## 5. Script Creazione e Popolamento Database

### 5.1. Creazione dello schema

**NOTA:** Nello schema precedente gli id sono stati presentati come ObjectId, tuttavia nella base dati sono ObjectId salvati come stringa per via della modalità di collezionamento dei dati da Twitter che li ha trascritti in quel modo.

*Per la creazione delle collezioni del database si è deciso di utilizzare un JSON schema. Si tratta di un vocabolario che consente di annotare e validare documenti JSON. La creazione dello schema e degli indici è riportata nel file createSchema.js.*

```
db = connect("localhost:27017")

// create database Twitter
db = db.getSiblingDB('Twitter')

// drop all the collections
db.dropDatabase()

// create the collections

// create collection 'Users'
db.createCollection('Users', {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      title: "Users object validation",
      required: ["_id", "username", "verified", "following", "followers"],
      properties: {
        username: {
          bsonType: "string",
          description: "Username of the user. Required.",
        },
        verified: {
          bsonType: "bool",
          description: "Whether the user is verified or not. Required.",
        },
        following: {
          bsonType: "int",
          description: "Number of users the user is following. Required. It must  
be greater than or equal to 0.",
          minimum: 0,
        },
        followers: {
```

```
        bsonType: "int",
        description: "Number of users following the user. Required. It must be
greater than or equal to 0.",
        minimum: 0,
    }
}
}
}
}))
print("Users' schema created.")

// create collection 'Trends'
db.createCollection('Trends', {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["_id", "url", "name", "location"],
            properties: {
                url: {
                    bsonType: "string",
                    description: "URL of the trend. Required.",
                },
                name: {
                    bsonType: "string",
                    description: "Name of the trend. Required.",
                },
                location: {
                    bsonType: "string",
                    description: "Location of the trend. Required.",
                }
            }
        }
    }
})

print("Trends' schema created.")

// create collection 'Tweets'
db.createCollection('Tweets', {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            required: ["_id", "url", "username", "text", "retweets", "likes", "shares",
"sentiment"],
            properties: {
                url: {
```

```
        bsonType: "string",
        description: "URL of the tweet. Required.",
    },
    username: {
        bsonType: "string",
        description: "Username of the user who tweeted. Required.",
    },
    text: {
        bsonType: "string",
        description: "Text of the tweet. Required.",
    },
    retweets: {
        bsonType: "int",
        description: "Number of retweets. Required. It must be greater than or
equal to 0.",
        minimum: 0,
    },
    likes: {
        bsonType: "int",
        description: "Number of likes. Required. It must be greater than or
equal to 0.",
        minimum: 0,
    },
    shares: {
        bsonType: "int",
        description: "Number of shares. Required. It must be greater than or
equal to 0.",
        minimum: 0,
    },
    sentiment: {
        bsonType: "double",
        description: "Sentiment of the tweet. Required.",
    }
}

})

print("Tweets' schema created.")
```

*Si è deciso di creare poi due indici per velocizzare le operazioni frequenti su data, nome e luogo di un trend e sull'username di un utente.*

```
// create index for 'Trends' on 'name', 'date', 'location' and rename to
name_date_location
db.Trends.createIndex({ name: 1, date: 1, location: 1 }, {unique : true, name :
"name_date_location_index"})
```



```
print("Index created for Trends.")

// create index for 'Users' on 'username'
db.Users.createIndex({ username: 1 }, {unique : true, name : "username_index"})
print("Index created for Users.")
```

*Per gli altri eventuali campi non definiti nello schema, non è presente una validazione perché si è ritenuto potessero anche avere formati diversi.*

## 5.2. Popolamento del database

Per il popolamento del database si è deciso di utilizzare uno script python (*loadData.py*) che si occupa di prendere i dati da file in formato JSON e caricarli nelle collezioni opportune. Per la sua esecuzione è richiesta la previa installazione di *pymongo*.

```
try:
    import pymongo
except ImportError:
    print("pymongo is not installed. Please install it using 'pip install pymongo'")
    exit(1)

import json, os

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["Twitter"]

# read the json file from data/trends/*.json and insert them into the database in the
collection "Trends"
for filename in os.listdir("database/data/trends"):
    if filename.endswith(".json"):
        with open("database/data/trends/" + filename, "r") as f:
            data = json.load(f)
            db["Trends"].insert_many(data)
            print("Inserted", len(data), "documents into the collection 'Trends'")

# read the json file from data/tweets/*.json and insert them into the database in the
collection "Tweets"
for filename in os.listdir("database/data/tweets"):
    if filename.endswith(".json"):
        with open("database/data/tweets/" + filename, "r") as f:
            data = json.load(f)
            db["Tweets"].insert_many(data)
            print("Inserted", len(data), "documents into the collection 'Tweets'")
```

```
# read the json file from data/users/*.json and insert them into the database in the
collection "Users"
for filename in os.listdir("database/data/users"):
    if filename.endswith(".json"):
        with open("database/data/users/" + filename, "r") as f:
            data = json.load(f)
            db["Users"].insert_many(data)
            print("Inserted", len(data), "documents into the collection 'Users'")

print("Done")
```

### 5.3. Operazioni di inserimento

Si riporta di seguito l'esempio di operazione di inserimento nella collezione *Trend* presente nel file *insertTrend.js*.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

today = new Date().toString();

id = "6543cb8f825f3a0c441ce76c"

const trend = {
  _id: id,
  name: "AI",
  date: today,
  location: "Worldwide",
  url: "https://twitter.com/search?q=AI&src=typed_query"
}

if (db.Trends.findOne({ _id: id }) != null) {

  print("Trend already exists.")

} else {

  db.Trends.insertOne(trend)
  print("Trend inserted successfully.")

}
```

Lo script controlla prima che il trend che si intende inserire non sia già presente e, se non lo è, viene inserito, altrimenti viene stampato un messaggio di errore.

Si riporta di seguito l'esempio di operazione di inserimento nella collezione *User* presente nel file *insertUser.js*.

```
db = connect("localhost:27017")

// connect to pre-existing database Twitter
db = db.getSiblingDB('Twitter')

// create a user who published a new tweet related to a trend
const user = {
  _id: "6543cb92825f3a0c441ce76d",
  profile_name: "John Doe",
  username: "@johndoe",
  verified: false,
  bio: "I'm a programmer.",
  location: "Napoli",
  url: "johndoe.com",
  following: 192,
  followers: 2100000
}

if (db.Users.findOne({ username: "@johndoe" }) != null) {

  print("User already exists.")

} else {

  db.Users.insertOne(user)
  print("User inserted successfully.")

}
```

Lo script controlla che l'utente non esista già e, se non esiste, lo inserisce.

Si riporta di seguito l'esempio di operazione di inserimento nella collezione *Tweet* presente nel file *insertTweet.js*.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

const trend = db.Trends.findOne({ _id: "6543cb8f825f3a0c441ce76c" })

if (trend == null) {

  print("Trend doesn't exist.")

}
```

```
} else {

    const user = db.Users.findOne({ username: "@johndoe" })

    if (user == null) {

        print("User doesn't exist.")

    } else {

        id = "6543cb95825f3a0c441ce76e"

        // tweet to insert
        const tweet = {
            _id: id,
            username: "@johndoe",
            name: "John Doe",
            text: "AI is the future.",
            replies: 0,
            retweets: 0,
            likes: 0,
            shares: 0,
            sentiment: 0.2,
            url: "https://twitter.com/johndoe/status/1",
            trends: [trend._id],
            user_id: user._id
        }

        // insert tweet
        db.Tweets.insertOne(tweet)

        // insert tweet id in the user's tweets array
        db.Users.updateOne(
            { _id: user._id },
            { $push: { tweets: id } }
        )

        // insert tweet id in the trend's tweets array
        db.Trends.updateOne(
            { _id: trend._id },
            { $push: { tweets: id } }
        )

        print("Tweet inserted successfully.")

    }
}
```

```
}
```

Un tweet viene inserito se esiste il trend a cui afferisce se esiste l'utente che lo ha scritto.

## 5.4. Operazioni di aggiornamento

Si riporta di seguito l'esempio di operazione di aggiornamento nella collezione *Tweet* presente nel file *updateTweet.js*.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

tweet = db.Tweets.findOne({_id: "6543cb95825f3a0c441ce76e"})

if (tweet == null) {

    print("Tweet doesn't exist.")

} else {

    db.Tweets.updateOne(
        { _id: tweet._id },
        { $set: { likes: 10, shares: 2 } }
    )

    print("Tweet updated successfully.")

}
```

Tale script effettua l'aggiornamento del numero di likes e di shares del tweet, se esso esiste.

Si riporta di seguito l'esempio di operazione di aggiornamento nella collezione *Trend* presente nel file *updateTrend.js*.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

const trend = db.Trends.findOne({_id: "6543cb8f825f3a0c441ce76c"})

if (trend == null) {

    print("Trend doesn't exist.")

}
```

```
} else {  
  
  db.Trends.updateOne(  
    { _id: trend._id },  
    { $set: { date: new Date() } }  
  )  
  
  print("Trend updated successfully.")  
  
}
```

In questo script se il trend esiste, ne viene aggiornata la data a quella attuale.

Si riporta di seguito l'esempio di operazione di aggiornamento nella collezione *User* presente nel file *updateUser.js*.

```
db = connect("localhost:27017")  
  
db = db.getSiblingDB('Twitter')  
  
const user = db.Users.findOne({ username: "@johndoe" })  
  
if (user == null) {  
  
  print("User doesn't exist.")  
  
} else {  
  
  db.Users.updateOne(  
    { _id: user._id },  
    { $set: { bio: "I'm a programmer and a data scientist." } }  
  )  
  
  print("User updated successfully.")  
  
}
```

In questo script, se l'utente esiste, ne viene aggiornata la biografia.

## 5.5. Operazioni di rimozione

Si riporta di seguito l'esempio di operazione di rimozione dalla collezione *User* presente nel file *deleteUser.js*.

```
db = connect("localhost:27017")
```

```
db = db.getSiblingDB('Twitter')

user = db.Users.findOne({ username: "@johndoe" })

if (user == null) {

    print("User doesn't exist.")

} else {

    if (user.tweets == null) {

        print("User doesn't have any tweets.")

    } else {

        user.tweets.forEach(tweet_id => {

            tweets = db.Tweets.find({ _id: tweet_id })

            tweets.forEach(tweet => {

                tweet.trends.forEach(trend_id => {
                    db.Trends.updateOne(
                        { _id: trend_id },
                        { $pull: { tweets: tweet._id } }
                    )
                })

                db.Tweets.deleteOne(
                    { _id: tweet._id }
                )
            })
        })
    }

}

db.Users.deleteOne(
    { username: user.username }
)

print("User deleted successfully.")
```

Se l'utente esiste e ha scritto dei tweet, essi vengono cancellati dalla collezione *Tweet* e ne viene rimosso il riferimento dai rispettivi trend a cui afferiscono. Infine viene rimosso l'utente.

Si riporta di seguito l'esempio di operazione di rimozione dalla collezione *Tweet* presente nel file *deleteTweet.js*.

```
db = connect("localhost:27017")

// connect to pre-existing database Twitter
db = db.getSiblingDB('Twitter')

tweet = db.Tweets.findOne({ _id: "6543cb95825f3a0c441ce76e" })

if (tweet == null) {

    print("Tweet doesn't exist.")

} else {

    // find the user who wrote the tweet and remove the id of this tweet from his/her
    tweets array
    db.Users.updateOne(
        { _id: tweet.user_id },
        { $pull: { tweets: tweet._id } }
    )

    tweet.trends.forEach(trend => {
        // find the trend related to the tweet and remove the id of this tweet from its
        tweets array
        db.Trends.updateOne(
            { _id: trend },
            { $pull: { tweets: tweet._id } }
        )
    })

    db.Tweets.deleteOne(
        { _id: tweet._id }
    )

    print("Tweet deleted successfully.")

}
```

Per la rimozione del tweet, se esso esiste, se ne rimuove il riferimento sia dal trend a cui afferisce, che dall'utente che lo ha scritto e poi si effettua la rimozione.



Si riporta di seguito l'esempio di operazione di rimozione dalla collezione *Trend* presente nel file *deleteTrend.js*.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

id = "6543cb8f825f3a0c441ce76c"

trend = db.Trends.findOne({ _id: id })

if (trend == null) {

    print("Trend doesn't exist.")

} else {

    if (trend.tweets == null) {

        print("Trend doesn't have any tweets.")

    } else {

        trend.tweets.forEach(tweet_id => {

            tweets = db.Tweets.find({ _id: tweet_id })

            tweets.forEach(tweet => {
                db.Users.updateOne(
                    { _id: tweet.user_id },
                    { $pull: { tweets: tweet._id } }
                )
            })

            db.Tweets.deleteOne(
                { _id: tweet_id }
            )

        })

    }

    db.Trends.deleteOne({ _id: id })

    print("Trend deleted successfully.")
}
```

}

Se il trend che si intende cancellare esiste nella collezione, allora si prendono tutti i tweet che vi fanno riferimento, si rimuove il loro riferimento dall'array dei tweet degli utenti che li hanno scritti e si cancellano. Infine si cancella il trend.

## 6. Query

### 6.1. Aggregation: Average sentiment per Trend

*Per ogni trend, selezionare tutti i tweet associati al trend e mostrare il sentiment ottenuto come media dei sentiment dei tweet selezionati. La query si trova nel file Operation1.js.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

trends = db.getCollection('Trends').find({});

trends.forEach(function (trend) {

    result = db.getCollection('Trends').aggregate([
        {
            $match: {
                name: trend.name,
                location: trend.location,
                date: trend.date
            }
        }, {
            $unwind: {
                path: '$tweets'
            }
        }, {
            $lookup: {
                from: 'Tweets',
                localField: 'tweets',
                foreignField: '_id',
                as: 'tweetsData'
            }
        }, {
            $unwind: {
                path: '$tweetsData'
            }
        }, {
            // print the trend name and the average sentiment
            $group: {
                _id: {
                    name: '$name',
                    location: '$location',
                    date: '$date'
                },
            },
```

```
        sentiment: {
            $avg: '$tweetsData.sentiment'
        }
    }
}

});

printjson(result.toArray());

});
```

## 6.2. Aggregation: Sentiment percentages

*Per ogni trend, selezionare tutti i tweet che vi afferiscono e, per ogni categoria di sentiment che il tweet può assumere, stampare la percentuale di tweet che hanno ottenuto quel determinato sentiment. La query si trova nel file `Operation2.js`.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

trends = db.getCollection('Trends').find({});

trends.forEach(function (trend) {
    result = db.getCollection('Trends').aggregate(
        [
            {
                $match: {
                    name: trend.name,
                    location: trend.location,
                    date: trend.date
                }
            }, {
                $unwind: {
                    path: '$tweets'
                }
            }, {
                $lookup: {
                    from: 'Tweets',
                    localField: 'tweets',
                    foreignField: '_id',
                    as: 'tweetsData'
                }
            }, {
                $unwind: {
                    path: '$tweetsData'
```

```
    }
  }, {
    $group: {
      _id: '$_id',
      totalTweets: {
        $sum: 1
      },
      positiveTweets: {
        $sum: {
          $cond: [
            {
              $gt: [
                '$tweetsData.sentiment', 0.2
              ]
            }, 1, 0
          ]
        }
      },
      neutralTweets: {
        $sum: {
          $cond: [
            {
              $and: [
                {
                  $gte: [
                    '$tweetsData.sentiment', -0.2
                  ]
                }, {
                  $lte: [
                    '$tweetsData.sentiment', 0.2
                  ]
                }
              ]
            }, 1, 0
          ]
        }
      },
      negativeTweets: {
        $sum: {
          $cond: [
            {
              $lt: [
                '$tweetsData.sentiment', -0.2
              ]
            }, 1, 0
          ]
        }
      }
    }
  }
}
```

```
        }
      }
    }
  }, {
    $project: {
      totalTweets: 1,
      positivePercentage: {
        $multiply: [
          {
            $divide: [
              '$positiveTweets', '$totalTweets'
            ]
          }, 100
        ]
      },
      neutralPercentage: {
        $multiply: [
          {
            $divide: [
              '$neutralTweets', '$totalTweets'
            ]
          }, 100
        ]
      },
      negativePercentage: {
        $multiply: [
          {
            $divide: [
              '$negativeTweets', '$totalTweets'
            ]
          }, 100
        ]
      }
    }
  }
]
);

printjson(result.toArray());
});
```

### 6.3. Aggregation: Trend diffusion degree

*Dato un certo trend, identificare tutti gli utenti che hanno pubblicato tweet che vi afferiscono e sulla base del loro numero di followers identificare quante persone sono state all'incirca raggiunte*

*dal trend per ognuno di quei tweet, come somma del numero di followers. La query si trova nel file **Operation3.js**.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

trendName = "#Halloween"

result = db.getCollection('Tweets').aggregate([
  {
    $match: {
      text: {
        $regex: '#Halloween',
        $options: 'i'
      }
    }
  }, {
    $lookup: {
      from: 'Users',
      localField: 'user_id',
      foreignField: '_id',
      as: 'usersData'
    }
  }, {
    $unwind: {
      path: '$usersData'
    }
  }, {
    $group: {
      _id: '$_id',
      usersReached: {
        $sum: {
          $subtract: [
            '$usersData.followers', 1
          ]
        }
      }
    }
  }
])

printjson(result.toArray())
```

## 6.4. Aggregation: User coherence score

*Per ogni utente, raggruppare i tweet che ha scritto per topic e, per ogni cluster, assegnare all'utente un punteggio di coerenza, fare la media dei punteggi ottenuti. La query si trova nel file `Operation4.js`.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

result = db.getCollection("Users").aggregate([
  {
    $lookup: {
      from: "Tweets",
      localField: "tweets",
      foreignField: "_id",
      as: "userTweets"
    }
  },
  {
    $unwind: "$userTweets"
  },
  {
    $lookup: {
      from: "Trends",
      localField: "userTweets.trends",
      foreignField: "_id",
      as: "tweetTrends"
    }
  },
  {
    $unwind: "$tweetTrends"
  },
  {
    $group: {
      _id: {
        userId: "$_id",
        trendId: "$tweetTrends._id"
      },
      userTweets: { $push: "$userTweets" },
      coherenceScores: { $avg: "$userTweets.sentiment" }
    }
  },
  {
```



```
    $group: {
      _id: "$_id.userId",
      coherenceScoresByUser: {
        $push: {
          trendId: "$_id.trendId",
          coherenceScore: "$coherenceScores"
        }
      }
    },
    {
      $project: {
        _id: 0,
        userId: "$_id",
        coherenceScoresByUser: 1
      }
    }
  ]);

printjson(result.toArray())
```

## 6.5. Aggregation: User's sentiment percentages

*Dato un utente, prendi i tweet che ha scritto e calcola le percentuali di tweet con sentiment positivo, negativo e neutro. La query si trova nel file Operation5.js.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

username = "@Mando3Updates"

result = db.getCollection("Users").aggregate([
  {
    $match: {
      username: username
    }
  },
  {
    $lookup: {
      from: "Tweets",
      localField: "tweets",
      foreignField: "_id",
      as: "userTweets"
    }
  },
])
```

```

{
  $unwind: "$userTweets"
},
{
  $group: {
    _id: "$_id",
    userTweets: { $push: "$userTweets" },
    positiveTweets: { $sum: { $cond: [{ $gt: ['$tweetsData.sentiment', 0.2] }, 1,
0] } },
    neutralTweets: {
      $sum: {
        $cond: [{
          $and: [{
            $gte: [
              '$tweetsData.sentiment', -0.2
            ]
          }, {
            $lte: [
              '$tweetsData.sentiment', 0.2
            ]
          }
        ]
      }, 1, 0]
    }
  },
    negativeTweets: { $sum: { $cond: [{ $lt: ['$tweetsData.sentiment', -0.2] }, 1,
0] } }
  }
},
{
  $project: {
    _id: 0,
    userId: "$_id",
    totalTweets: { $size: "$userTweets" },
    positivePercentage: { $multiply: [{ $divide: ["$positiveTweets", { $size:
"$userTweets" }] }, 100] },
    negativePercentage: { $multiply: [{ $divide: ["$negativeTweets", { $size:
"$userTweets" }] }, 100] },
    neutralPercentage: { $multiply: [{ $divide: ["$neutralTweets", { $size:
"$userTweets" }] }, 100] },
  }
}
])

```

## 6.6. Aggregation: Engagement metrics computation

*Per ogni trend calcolare il numero medio di like, condivisioni e retweet che i suoi post hanno avuto. Incollare lo script per la query. La query si trova nel file Operation6.js.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

result = db.getCollection("Trends").aggregate([
  {
    $unwind: {
      path: '$tweets'
    }
  }, {
    $lookup: {
      from: 'Tweets',
      localField: 'tweets',
      foreignField: '_id',
      as: 'tweetsData'
    }
  }, {
    $unwind: {
      path: '$tweetsData'
    }
  },
  {
    $group: {
      _id: '$_id',
      name: {
        '$first': '$name'
      },
      avgLikes: {
        '$avg': '$tweetsData.likes'
      },
      avgShares: {
        '$avg': '$tweetsData.shares'
      },
      avgRetweets: {
        '$avg': '$tweetsData.retweets'
      }
    }
  }, {
    $project: {
      _id: 0,
      name: 1,
```

```
        avgLikes: 1,
        avgShares: 1,
        avgRetweets: 1
    }
}
])

printjson(result.toArray())
```

## 6.7. Aggregation: Discussions' detection

*Dato un trend, per ogni tweet ad esso associato, verificare se i suoi commenti hanno dato origine a una discussione identificando eventuali sentiment discordanti. La query si trova nel file `Operation7.js`.*

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

trendName = "#bonustrasporti"
trendLocation = "Italy"
trendDate = "2023-11-01T16:29:31.371203"

result = db.getCollection("Trends").aggregate([
{
    $match: {
        name: trendName,
        location: trendLocation,
        date: trendDate
    }
}, {
    $unwind: {
        path: '$tweets'
    }
}, {
    $lookup: {
        from: 'Tweets',
        localField: 'tweets',
        foreignField: '_id',
        as: 'tweetsData'
    }
}, {
    $unwind: {
        path: '$tweetsData'
    }
}
```

```
    }, {
      $unwind: {
        path: '$tweetsData.comments'
      }
    }, {
      $group: {
        _id: '$tweetsData._id',
        text: {
          $first: '$tweetsData.text'
        },
        comments: {
          $push: '$tweetsData.comments'
        }
      }
    }, {
      $project: {
        _id: 0,
        hasDiscordantSentiment: {
          $reduce: {
            input: '$comments',
            initialValue: false,
            in: {
              $cond: {
                if: {
                  $ne: [
                    '$$this.sentiment', '$$value.sentiment'
                  ]
                },
                then: true,
                else: '$$value'
              }
            }
          }
        }
      }
    }
  ]
})

printjson(result.toArray())
```

## 6.8. Altre query

### 6.8.1. Utente da più tempo su Twitter

Ottieni l'utente che si trova da più tempo su Twitter.

```
db = connect("localhost:27017")
```

```
db = db.getSiblingDB('Twitter')

users = db.Users.find({}).toArray()

users.sort(function (a, b) {
    return a.age - b.age
})

print("The eldest user is: " + users[0].username + " which " + users[0].joined_date)
```

### 6.8.2. Tweet più condivisi

Ottieni i tweets più condivisi.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

tweets = db.Tweets.find({}).sort({shares: -1}).limit(10).toArray()

toPrint = tweets.map(function (tweet) {
    return {text: tweet.text, trend: db.Trends.findOne({_id: { $in: tweet.trends}}).name,
    shares: tweet.shares}
})

print("The 10 most shared tweets are: ")
printjson(toPrint)
```

### 6.8.3. Trends più popolari

Ottieni i trend più popolari.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

minTweets = 5

print("Trends with more than " + minTweets + " tweets:\n")

db.Trends.find({}).forEach(function (trend) {

    if (trend.tweets.length > minTweets) {

        print("Trend: " + trend.name + " has " + trend.tweets.length + " tweets\n")
    }
})
```

```
}  
  
))
```

#### 6.8.4. Utenti più popolari

Ottieni i tweet più popolari.

```
db = connect("localhost:27017")  
  
db = db.getSiblingDB('Twitter')  
  
minFollowers = 1000  
  
print("Users with more than " + minFollowers + " followers:\n")  
  
db.Users.find({}).forEach(function (user) {  
  
    if (user.followers > minFollowers) {  
  
        print("User: " + user.username + " has " + user.followers + " followers and " +  
user.tweets.length + " tweets\n")  
  
    }  
  
})
```

#### 6.8.5. Ottieni i tweet per trends

Stampa tutti i tweet pubblicati per un trend.

```
db = connect("localhost:27017")  
  
db = db.getSiblingDB('Twitter')  
  
db.Trends.find({}).forEach(function (trend) {  
  
    print("Trend: " + trend.name + "\n")  
  
    trend.tweets.forEach(function (tweet) {  
  
        toPrint = db.Tweets.findOne({ _id: tweet })  
  
        if (toPrint != null) {
```

```
print("\tTweet: " + toPrint.text + "\n")

if (toPrint.comments != null) {

    toPrint.comments.forEach(function (comment) {

        print("\t\tComment: " + comment.text + "\n")

    })
}

print("]\n")

}

})

})
```

### 6.8.6. Ottieni i tweet per gli utenti

Stampa tutti i tweet pubblicati dagli utenti.

```
db = connect("localhost:27017")

db = db.getSiblingDB('Twitter')

db.Users.find({}).forEach(function (user) {

    print("User: " + user.username + "\n")

    user.tweets.forEach(function (tweet) {

        toPrint = db.Tweets.findOne({ _id: tweet })

        if (toPrint != null) {

            print("\tTweet: " + toPrint.text + "\n")

        }

    })

})
```



})