

Manipolazione e sentiment analysis messaggi Twitter attraverso DB SQL e NoSQL

Dipartimento di Informatica: Università di Torino

Autori: Alessandro Saracco, Diego Ercoli



Corso: MAADB

A.A. 2021/2022

Piano della Presentazione

- Analisi dei requisiti
- Progettazione e Aspetti Implementativi
- Risultati
- Conclusione



Piano della Presentazione

- **Analisi dei requisiti**

- Progettazione e Aspetti Implementativi

- Risultati

- Conclusioni

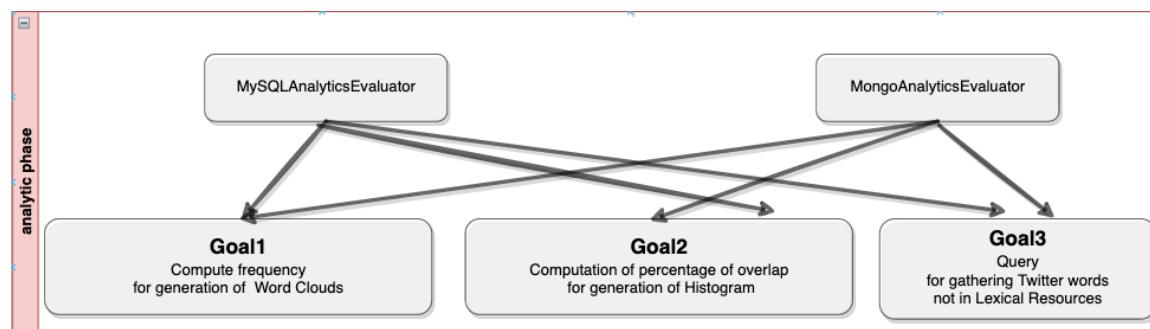
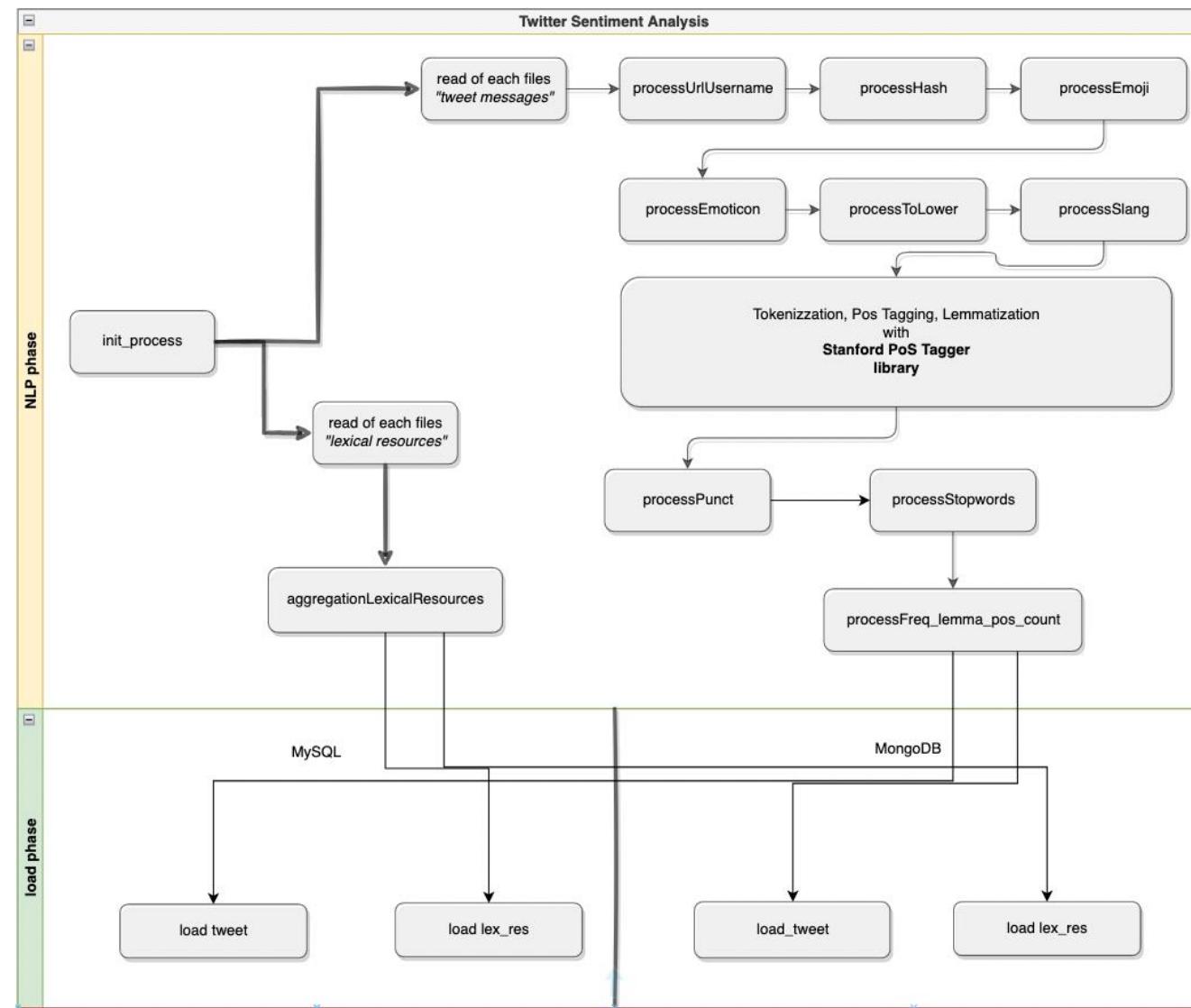


Use case

- **Goal1:**
- **Goal2:**



Activity Diagram



Piano della Presentazione

- Analisi dei requisiti
- **Progettazione e Aspetti Implementativi**
- Risultati
- Conclusioni



Piano della Presentazione

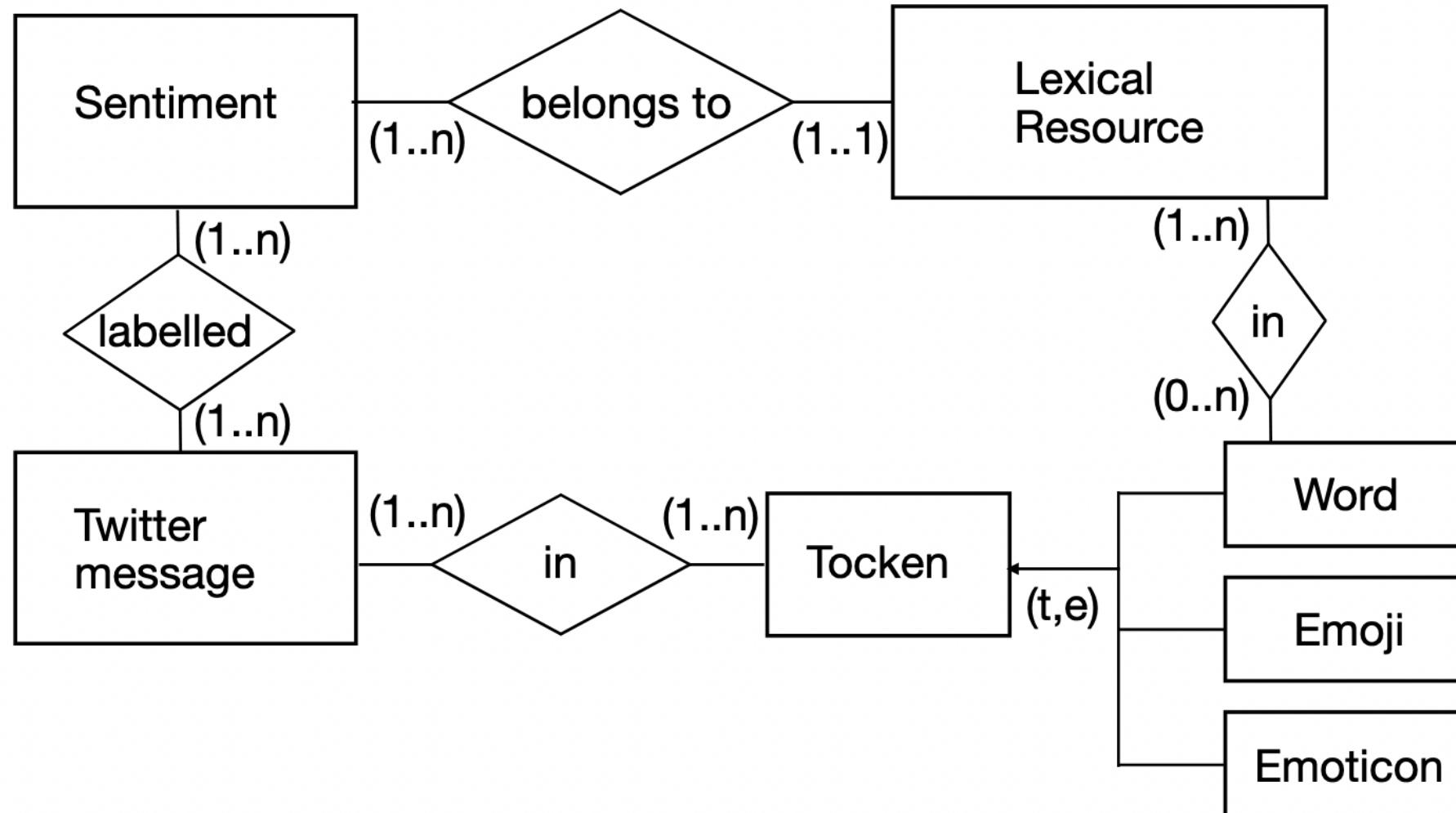
- Analisi dei requisiti
- Progettazione e Aspetti Implementativi

Data Modeling

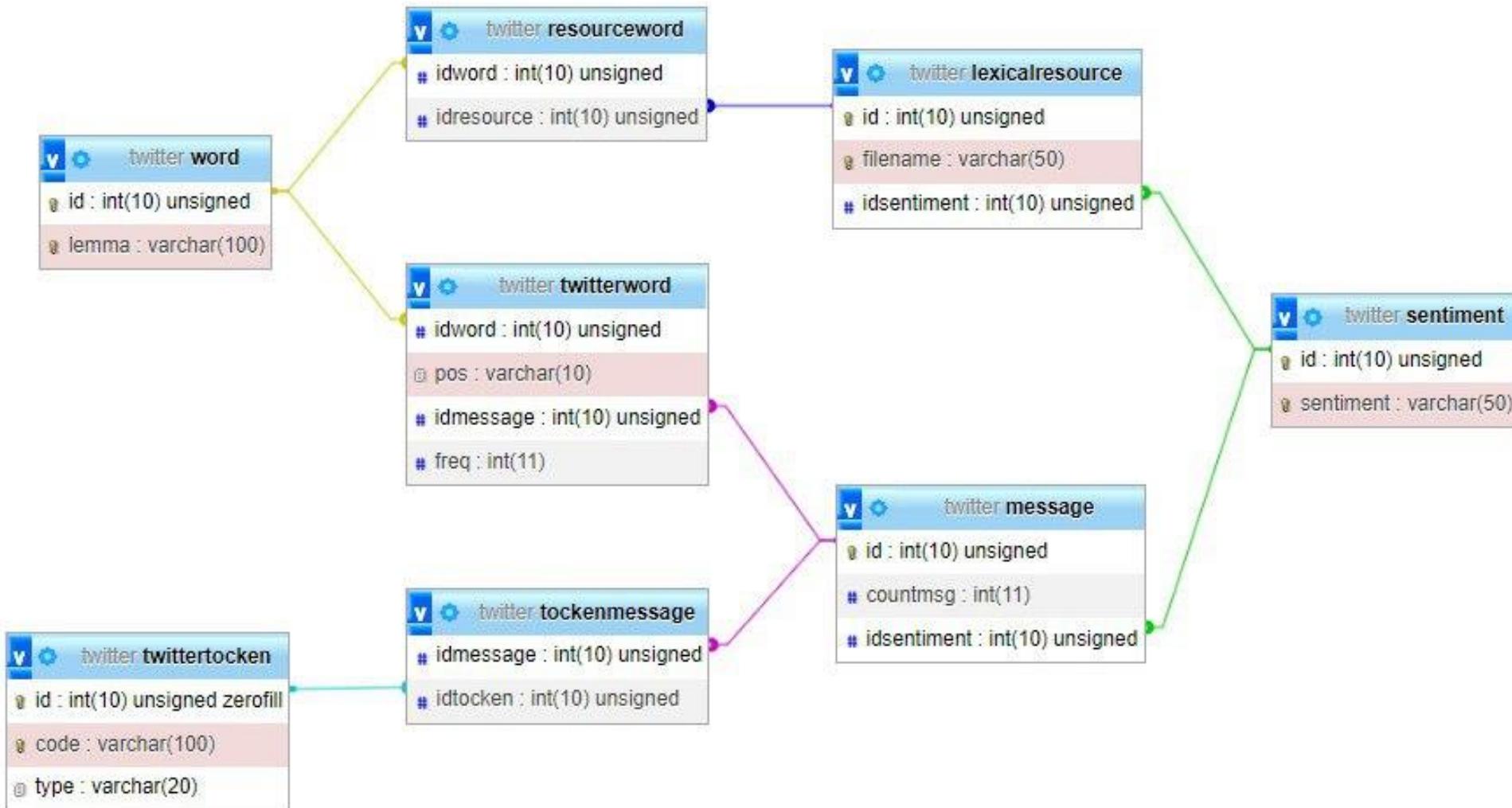
- Risultati
- Conclusioni



Diagramma Entità-Relazione



Schema Logico DB-Relazionale

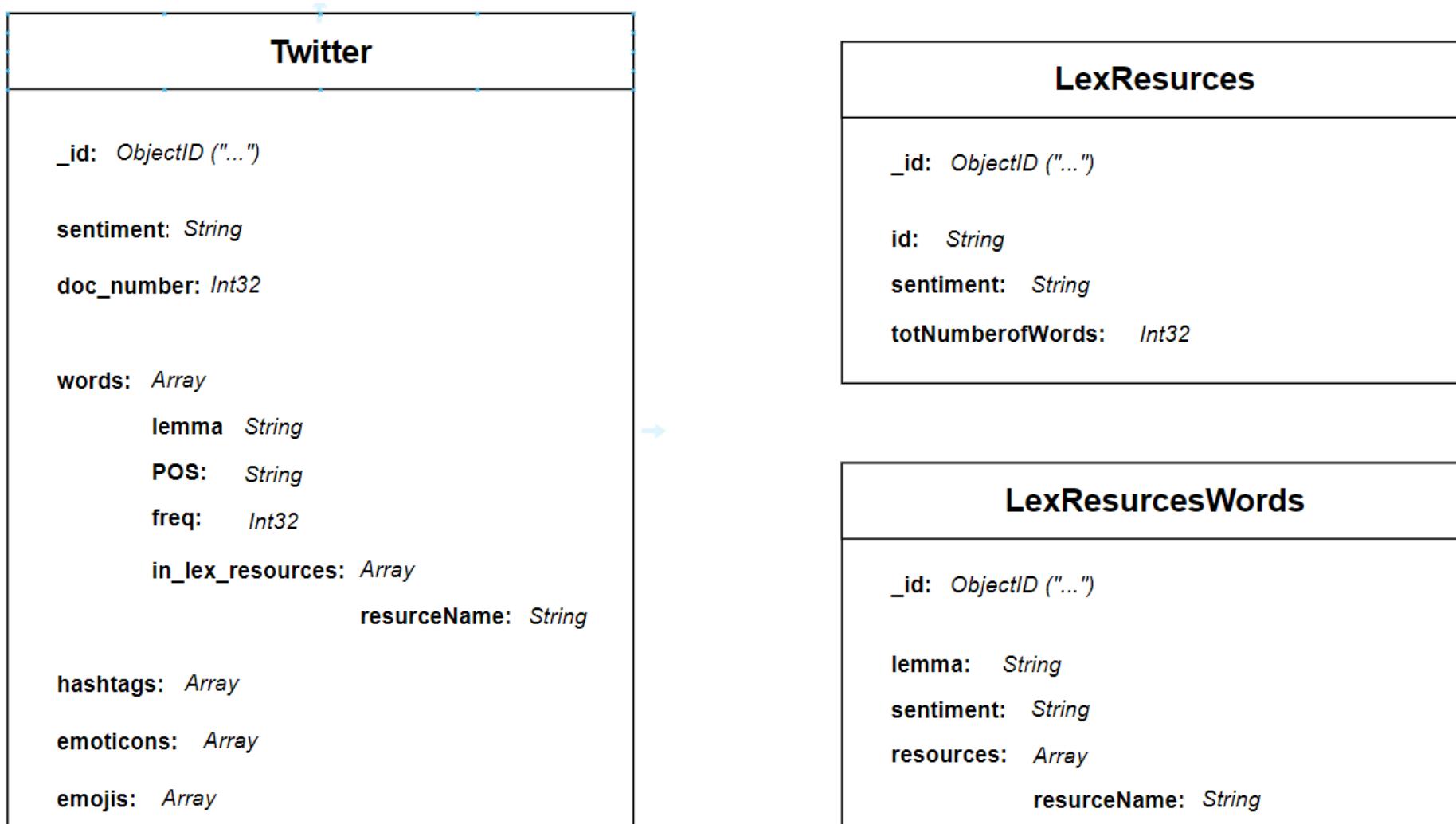


Traduzione Schema Logico DB-Relazionale

- Il database relazionale è **normalizzato**: ciò certifica la qualità dello schema in quanto predisposto ad eseguire qualsiasi tipo di query, oltre a garantire efficienza negli aggiornamenti
- La relazione **IS-A** è stata tradotta accorpando il genitore della generalizzazione nei figli ed unificando alcune entità figlie:
 - Elimazione entità padre Tocken
 - I figli Emoticon, Emoji, Hashtag sono stati accorpati in un'unica tabella (con un campo discriminatorio “type”, in quanto condiviscono la stessa struttura e le stesse associazioni con le altre entità.
 - Il figlio Word è rappresentato in una tabella separata, avendo diversi attributi ed un'associazione con LexicalResource



Struttura MongoDB



Schema in MongoDB

- Un tema comune di tutti i NoSQL databases è quello di essere schemaless e **aggregate-oriented**
- Ciò offre la possibilità di modellare gli aggregati in funzione delle queries e degli use cases che si presume di voler eseguire.
- In particolare, in un'ambiente a cluster vogliamo minimizzare il numero di nodi da interrogare per poter reperire tutti i dati correlati di una specifica istanza/record
- In quest'ottica nei DB documentali, è importante definire efficacemente la struttura aggregato con i suoi nested objects. In questo modo, quando siamo interessati ad una particolare istanza del dominio, le porzioni di dati che dovranno essere accedute e manipolate insieme, faranno parte dell'aggregato e quindi risiederanno nello stesso nodo



Impedence Mismatch

- Un vantaggio intrinseco dei databases NoSQL è quello di risolvere il problema dell'impedence mismatch, tipico invece dei modelli relazionali.
- Infatti il modello relazionale organizza dati in una struttura a tavole e righe, mentre nel paradigma OOP si manipolano oggetti

	OOP	RDBMS
Associazioni	puntatore ad oggetto	foreign key
Ereditarietà	classe e sottoclasse	single table, one table, ...
Identificatore univoco	non dichiarato	primary key

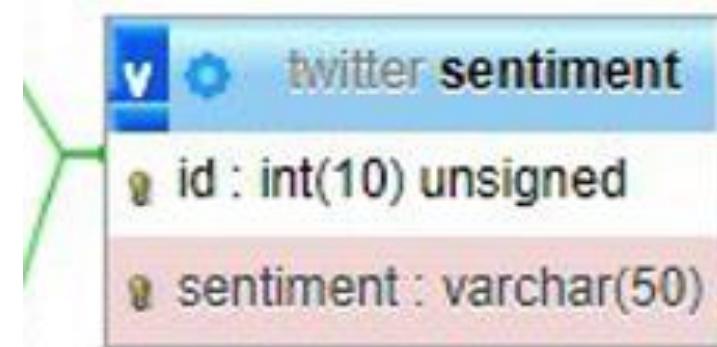
- Come effettuare l'integrazione nella parte di progetto in MySQL?



Object-Relational Mapping-1

- Abbiamo deciso di sperimentare il framework **Sql Alchemy ORM** per associare lo schema del database relazionale ai data objects (Model) della nostra applicazione in Python.
- Il modello di dominio diventa una fotografia del Database, definendo opportuni meta-tag

```
● ● ●  
class Sentiment(Base):  
    __tablename__ = 'Sentiment'  
    id = Column(Integer, primary_key=True)  
    sentiment = Column(String)  
  
    def __str__(self):  
        return "[" + str(self.sentiment) + "]"  
  
    def __repr__(self):  
        return self.__str__()
```



Object-Relational Mapping-2

- Vanno inoltre definite per ciascuna associazione le opportune **Foreign Keys**
- Deve inoltre essere tradotta l'associazione **IS-A**



Object-Relational Mapping-3

ORM in action: esecuzione di una query di inserimento auto-generata



```

session = Session()
sentiment = Sentiment(sentiment="Anger")
message1 = Message(countmsg=1)
message1.tockens = [Emoticon(code=":"), Hashtag(code="#cantwait")]
message1.words = [TwitterWord(word=Word(lemma="report"), pos="NN"),
                  TwitterWord(word=Word(lemma="enemy"), pos="NN"),
                  TwitterWord(word=Word(lemma="kill"), pos="VERB")]
message2 = Message(countmsg=1)
message2.tockens = []
sentiment.tweets.append(message1)
session.add(sentiment)
print(message1)
session.commit()
session.close()

```

```

INSERT INTO `Sentiment` (sentiment) VALUES (%(sentiment)s)
[generated in 0.00010s] {'sentiment': 'Anger'}
INSERT INTO `Word` (lemma) VALUES (%(lemma)s)
[generated in 0.00009s] {'lemma': 'report'}
INSERT INTO `Word` (lemma) VALUES (%(lemma)s)
[cached since 0.001721s ago] {'lemma': 'enemy'}
INSERT INTO `Word` (lemma) VALUES (%(lemma)s)
[cached since 0.003423s ago] {'lemma': 'kill'}
INSERT INTO `TwitterTocken` (code, type) VALUES (%(code)s, %(type)s)
[generated in 0.00007s] {'code': ':', 'type': 'emoticon'}
INSERT INTO `TwitterTocken` (code, type) VALUES (%(code)s, %(type)s)
[cached since 0.001446s ago] {'code': '#cantwait', 'type': 'hashtag'}
INSERT INTO `Message` (countmsg, idsentiment) VALUES (%(countmsg)s, %(idsentiment)s)
[generated in 0.00006s] {'countmsg': 1, 'idsentiment': 1}
INSERT INTO `TockenMessage` (idmessage, idtocken) VALUES (%(idmessage)s, %(idtocken)s)
[generated in 0.00006s] ({'idmessage': 1, 'idtocken': 1}, {'idmessage': 1, 'idtocken': 2})
INSERT INTO `TwitterWord` (pos, idword, idmessage, freq) VALUES (%(pos)s, %(idword)s, %(idmessage)s, %(freq)s)
[generated in 0.00006s] ({'pos': 'NN', 'idword': 1, 'freq': None}, {'pos': 'NN', 'idword': 2, 'freq': 1})

```



Inserimento in MongoDB

L'inserimento risulta essere molto più agevole, dal momento che i dati sono **denormalizzati** e gli oggetti in memoria vengono salvati in formato Json (in realtà BSON), senza necessità di ricorrere a framework

```
document_tweet = {}

document_tweet["sentiment"] = sent
document_tweet["doc_number"] = row

document_tweet["words"] = tweet_freq_count

if hashtags:
    document_tweet["hashtags"] = [item for item in set(hashtags)]
if emoji:
    document_tweet["emojis"] = [item for item in set(emoji)]
if emoticons:
    document_tweet["emoticons"] = [item for item in set(emoticons)]

#inserimento
twitterCollection.insert_one(document_tweet)
print(f" Inserita riga-{row} in sentimento {sent}")
```



Piano della Presentazione

- Analisi dei requisiti

- Progettazione e Aspetti Implementativi

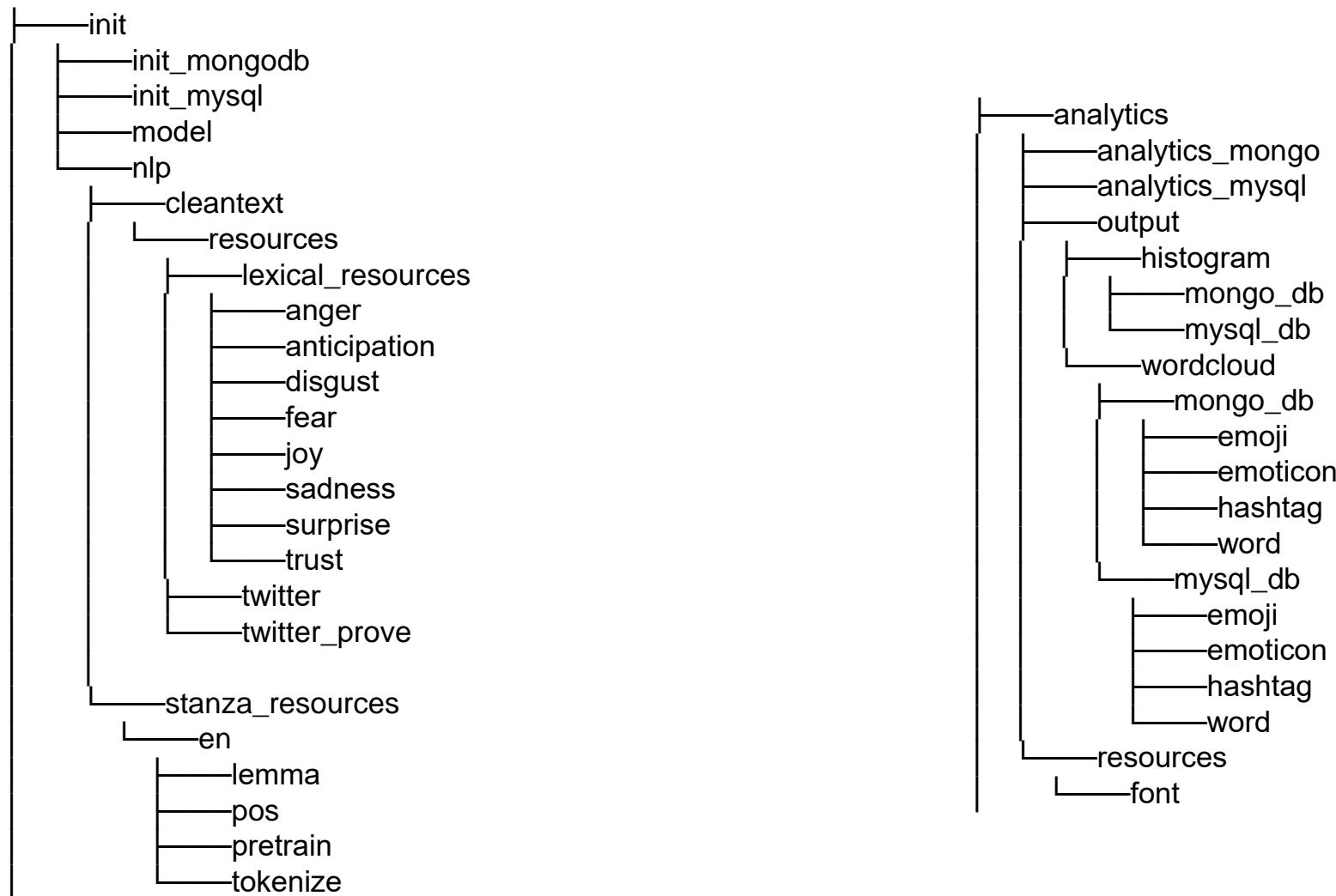
Struttura del progetto

- Risultati

- Conclusioni



Strutturazione progetto: packages

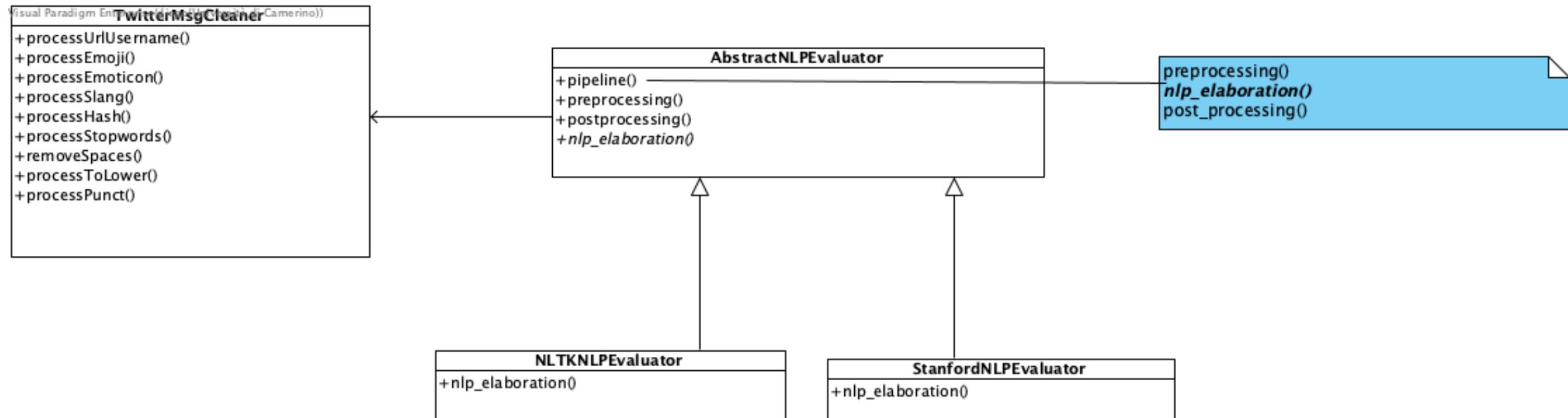


OOP: Design patterns

- Allo scopo di favorire la modularità e soprattutto il **riuso del codice** si è scelto di organizzare parte del codice sotto forma di classi
- Per definire la struttura di un algoritmo all'interno di un metodo, delegando alcuni passi dell'algoritmo alle sottoclassi, abbiamo utilizzato un famoso design pattern di tipo comportamentale chiamato **Template Method**
- Alcune caratteristiche di questo pattern:
 - Implementare la parte invariante di un algoritmo una volta per tutte e lasciare alle sottoclassi il compito di implementare il comportamento che può variare (**polimorfismo**) definendo metodi astratti.
 - Portare un comportamento fra sottoclassi a fattore comune per evitare codice duplicato (**boilerplate**)



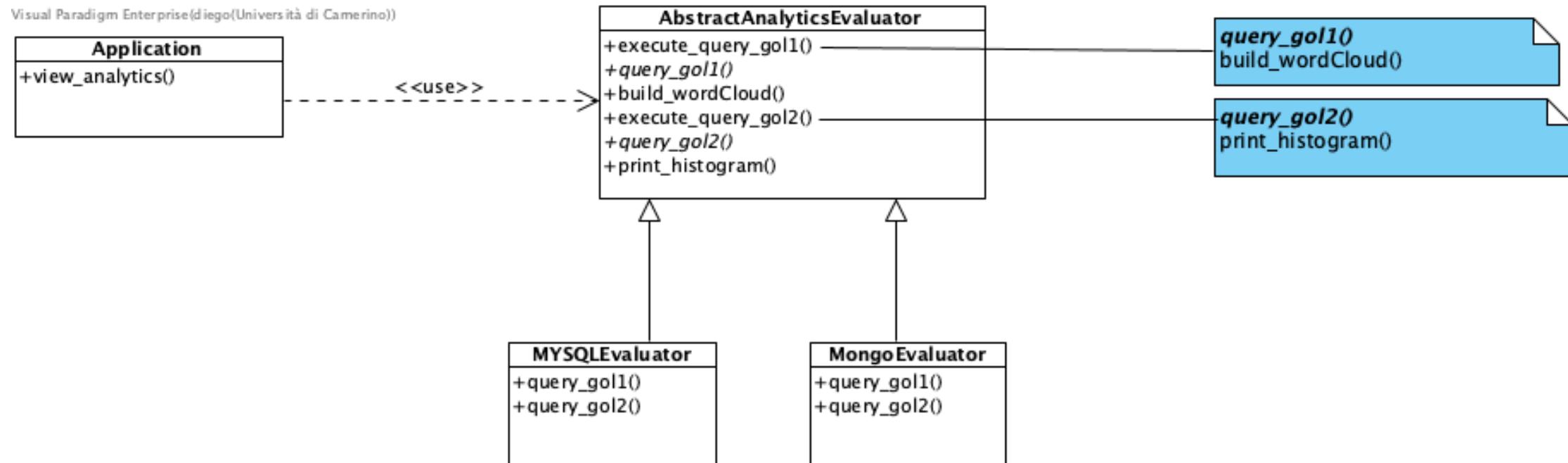
Template Method in NLP Pipeline



- La classe **TwitterMsgCleaner** è un componente generale riutilizzabile anche in altri progetti
- Il metodo **pipeline** definisce la struttura dell'algoritmo, delegando l'implementazione del metodo centrale ***nlp_elaboration*** alle sottoclassi
- L'utente può scegliere se effettuare l'elaborazione con la libreria NLTK o Stanford.



Template Method in Analytics



- In questo suo utilizzo, si può apprezzare ancora meglio i vantaggi offerti da questo design pattern
- I metodi per la generazione delle wordcloud e gli histogrammi vengono portati “a fattor comune”.
- Il metodo **execute_query_gol1** (**execute_query_gol2**) definisce il comportamento generale dell’algoritmo. Delega al metodo polimorfico, la modalità con cui ottenere il risultato. Inoltre, cronometra il tempo di esecuzione della query



Piano della Presentazione

- Analisi dei requisiti
- Progettazione e Aspetti Implementativi

Architettura Mongo DB Atlas

- Risultati
- Conclusioni



Descrizione Mongo Atlas

Per la realizzazione di questo progetto abbiamo scelto di usare nello specifico **MongoDB Atlas**, il quale consente di ospitare un database su una piattaforma di Cloud Computing (tipicamente questi servizi vengono denominati come DBaaS ovvero DataBase as a Service).

Con **MongoDB Atlas** è possibile usufruire gratuitamente del servizio scegliendo un tipo di server con RAM e CPU condivise e con una capacità di storage di 512 MB

Inoltre ci consente di avere un ambiente pratico da testare senza bisogno di alcun tipo di installazione e gestione a livello di infrastruttura.

Una volta creato, il **Cluster** sarà formato da **3 repliche** (Replica Set) ognuna delle quali mantiene gli stessi dati, fornendo ridondanza ed alta affidabilità.

Alla fine della procedura verrà poi definito il database e le 3 collezioni per lo storage dei dati



Caratteristiche MongoDB

Le caratteristiche fondamentali che rendono MongoDB il DB NoSQL più utilizzato sono:

- **Query ad hoc**
 - MongoDB supporta ricerche per campi, intervalli e regular expression.
- **Alta Disponibilità**
 - MongoDB fornisce alta disponibilità e aumento del carico gestito attraverso i replica set. Un replica set consiste in due o più copie dei dati. Ogni replica può avere il ruolo di copia primaria o secondaria in qualunque momento.
- **Sharding**
 - MongoDB scala orizzontalmente usando lo sharding. Infatti, l'utente deve scegliere una chiave di sharding, che determina come i dati di una collection saranno distribuiti tra i vari nodi, e questi dati sono divisi in intervalli (basati sulla chiave di shard) e distribuiti su molteplici shard. Inoltre MongoDB include un meccanismo di bilanciamento dei dati, spostando gli intervalli di dati da uno shard troppo carico a uno shard meno carico, in modo da bilanciare la distribuzione dei dati all'interno del cluster.
- **Aggregazione**
 - MongoDB supporta due modalità di aggregazione dei dati: il MapReduce e l'Aggregation Framework. Quest'ultimo lavora come una pipeline e permette di ottenere risultati molto più rapidamente del MapReduce grazie all'implementazione in C++.



Piano della Presentazione

- Analisi dei requisiti
- Progettazione e Aspetti Implementativi

Analytics

- Risultati
- Conclusioni



GOAL1-MySQL

- Per l'acquisizione di insight (vedi goal1, goal2) è stato sufficiente formulare delle query SQL aggregate sfruttando la definizione dello schema normalizzato.



```
def most_frequent_words():
    """
    Calcola per ogni sentimento le parole avente un certo pos-tag
    che occorrono con maggiore frequenza
    :param threshold:
    :return:
    """
    str1 =
        select S.sentiment as sentiment, W.lemma as code, sum(TW.freq) as freq
        from Word as W inner join TwitterWord TW on W.id = TW.idword
            inner join Message M on TW.idmessage = M.id
                inner join Sentiment S on M.idsentiment = S.id
        group by idsentiment,TW.idword
        order by sentiment,freq desc, code
```



```
def most_frequent_tokens(type):
    """
    Calcola per ogni sentimento i token (emoticon, hashtag, emoji)
    che occorrono con maggiore frequenza
    """
    str1 =
        select S.sentiment as sentiment, TT.code, count(*) as freq
        from TwitterTocken TT inner join TockenMessage TM on TT.id = TM.idtocken
            inner join Message M on TM.idmessage = M.id
                inner join Sentiment S on M.idsentiment = S.id
        where TT.type = %s
        group by idsentiment,TT.id
        order by sentiment,freq desc,code
```



GOAL2-MySQL

- Calcolo della sovrapposizione lessicale tra messaggi twitter e risorse lessicali per un certo sentimento

```
def resource_overlap():
    """
    Per ogni sentimento S, calcola il numero di parole condivise
    tra l'insieme X contenente tutti i lemmi di tutti i messaggi twitter relativi al sentimento S
    e l'insieme Y contenente tutti i lemmi di tutte le risorse lessicali relative al sentimento S
    :return:
    """
    str1 =
        select S.sentiment as sentiment, count(*) as overlap
        from Word as W inner join TwitterWord TW on W.id = TW.idword
            inner join Message M on TW.idmessage = M.id
                inner join ResourceWord RW on W.id = RW.idword
                    inner join LexicalResource LR on RW.idresource = LR.id
                        inner join Sentiment S on M.idsentiment = S.id and LR.idsentiment = S.id
        group by S.id
        order by sentiment
```

```
def count_N_twitter_words():
    """
    Per ogni sentimento, calcola il numero di parole presenti nel file di twitter
    :return:
    """
    str1 =
        select S.sentiment, count(idword) as N_twitter_words
        from TwitterWord TW inner join Message M on TW.idmessage = M.id
            inner join Sentiment S on M.idsentiment = S.id
        group by S.id
        order by sentiment
```

```
def count_N_lexical_words():
    """
    Per ogni sentimento, calcola il numero di parole presenti nel file di twitter
    :return:
    """
    str1 =
        select S.sentiment as sentiment, count(*) as N_lex_words
        from ResourceWord RW inner join LexicalResource LR on RW.idresource = LR.id
            inner join Sentiment S on LR.idsentiment = S.id
        group by S.id
        order by sentiment
```



GOAL3-MySQL

- ❑ Raccogliere le parole “nuove” presenti nella sorgente Tweet ma non nelle risorse lessicali

```
def find_new_resources():
    """
    Trova le nuove parole da aggiungere alle risorse lessicografiche
    :return:
    """

    str1 =
        select S.sentiment, w.lemma, tw.pos, count(*) as freq
        from Word w inner join TwitterWord tw on w.id = tw.idword
                      inner join Message M on tw.idmessage = M.id
                      inner join Sentiment S on M.idsentiment = S.id
        where w.id not in (
            select rw.idword
            from ResourceWord rw
        )
        group by S.id,tw.idword, tw.pos
        order by sentiment, freq desc
```



Aggregation Pipeline

Per la realizzazione dei due goal indicati, sono state eseguite le seguenti pipeline di aggregazione:

□ Goal 1: Generazione delle Word Cloud

1. Pipeline per il conteggio **lemmi** più frequenti per ogni sentimento
2. Pipeline per il conteggio degli **hashtag** più frequenti per ogni sentimento
3. Pipeline per il conteggio delle **emoticon** più frequenti per ogni sentimento
4. Pipeline per il conteggio delle **emoji** più frequenti per ogni sentimento

□ Goal 2: Generazioni Istogrammi

1. Pipeline per calcolare le statistiche (percentuali) delle parole delle risorse lessicali presenti anche nei messaggi di Twitter, definite da due variabili:

$$\circ \text{ } perc_presence_lex_res = \frac{N.\text{shared words}}{N.\text{lex_words}}$$

$$\circ \text{ } perc_presence_twitter = \frac{N.\text{shared words}}{N.\text{twitter_words}}$$



Aggregation Pipeline: un esempio pratico

Nelle successive slides, viene riportato un esempio di una pipeline, nello specifico, la pipeline per il

Conteggio della frequenza dei lemmi per ciascun sentimento

La pipeline verrà mostrata per stage in quanto non sarebbe stato possibile mostrare l'intero codice nell'interezza in un'unica diapositiva



Esempio pipeline aggregation: frequenza lemmi

In questa pipeline di aggregation andiamo ad effettuare il conteggio delle frequenze di ciascun lemma per ogni sentimento, e ci servirà per la generazione delle Word Cloud

Infatti, successivamente andremo a prendere i primi N risultati della pipeline, con N = parametro scelto arbitrariamente per definire quante parole andare a mostrare sulle Word Cloud.



Esempio pipeline aggregation: frequenza lemmi

Stage 1 : \$group

```
"_id": "$sentiment",
"words_list": {
    "$push": "$words"
}
```



```
_id: "sadness"
  words_list: Array
    0: Object
      lemma: "randomly"
      POS: "RB"
      freq: 1
    1: Object
_id: "joy"
  words_list: Array
    0: Object
      lemma: "girll"
      POS: "UH"
      freq: 1
    1: Object
_id: "fear"
  words_list: Array
    0: Object
      POS: "NNP"
      freq: 1
      lemma: "taylor"
    1: Object
```

Stage 2 : \$project

```
"words": {
    "$reduce": {
        "input": "$words_list",
        "initialValue": [],
        "in": {
            "$concatArrays": [
                "$$value",
                "$$this"
            ]
        }
    }
}
```



```
_id: "sadness"
  words: Array
    0: Object
      lemma: "randomly"
      POS: "RB"
      freq: 1
    1: Object
    2: Object
_id: "fear"
  words: Array
    0: Object
      lemma: "taylor"
      POS: "NNP"
      freq: 1
    1: Object
    2: Object
_id: "joy"
  words: Array
    0: Object
      freq: 1
      lemma: "girll"
      POS: "UH"
    1: Object
    2: Object
```



Esempio pipeline aggregation: frequenza lemmi

Stage 3 : \$unwind

```
{ path: "$words"}
```



```
▼ words: Object
  POS: "RB"
  freq: 1
  lemma: "randomly"
  _id: "sadness"
```

```
▼ _id: "sadness"
  ▼ words: Object
    lemma: "hot"
    POS: "JJ"
    freq: 1
```

```
▼ _id: "sadness"
  ▼ words: Object
    lemma: "social"
    POS: "JJ"
    freq: 1
```

Stage 4 : \$group

```
"_id": {
  "sentiment": "$_id",
  "lemma": "$words.lemma"
},
"count": {
  "$sum": {
    "$toInt": "$words.freq"
}}
```



```
▼ _id: Object
  lemma: "friend"
  sentiment: "anger"
  count: 3
```

```
▼ _id: Object
  sentiment: "disgust"
  lemma: "hum"
  count: 1
```

```
▼ _id: Object
  sentiment: "surprise"
  lemma: "day"
  count: 1
```



Esempio pipeline aggregation: frequenza lemmi

Stage 5 : \$sort

```
{ "count": -1 }
```



```
_id:Object
sentiment:"surprise"
lemma:"slut"
count:13
```

```
_id:Object
sentiment:"joy"
lemma:"laugh"
count:12
```

```
_id:Object
sentiment:"surprise"
lemma:"laugh"
count:12
```

Stage 6 : \$group

```
"_id": "$_id.sentiment",
"words": {
    "$push": {
        "lemma": "$_id.lemma",
        "freq": "$count"
    }
}
```



```
_id:"disgust"
words:Array
0:Object
lemma:"feel"
freq:5
1:Object
lemma:"laugh"
freq:5
```

```
_id:"surprise"
words:Array
0:Object
lemma:"slut"
freq:13
1:Object
lemma:"laugh"
freq:12
```

```
_id:"anticipation"
words:Array
0:Object
lemma:"laugh"
freq:9
1:Object
lemma:"slut"
freq:7
```



Piano della Presentazione

- Analisi dei requisiti
- Progettazione e Aspetti Implementativi
- **Risultati**
- Conclusione



Tempistiche Goal-1

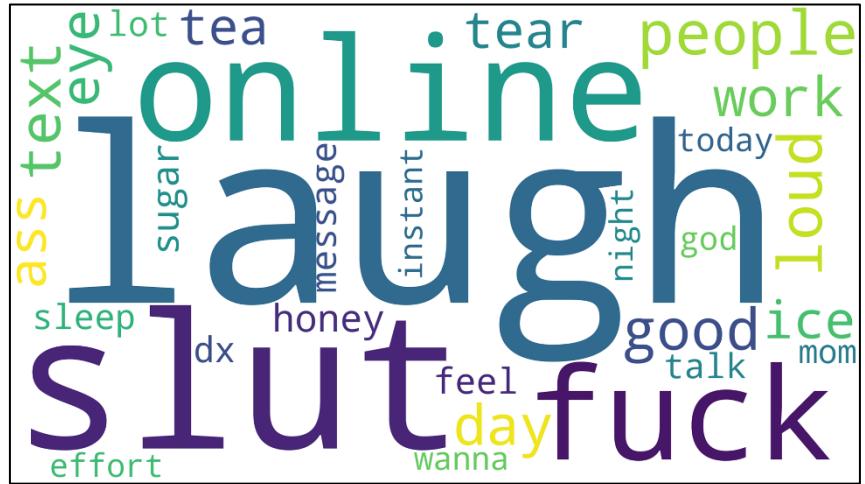
```
MYSQL_DB ha richiesto per il conteggio di word all'incirca 23.06 secondi
anger -> {'laugh': 3668, 'slut': 3591, 'online': 2929, 'fuck': 2742, 'people': 2686, 'loud': 2648, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
anticipation -> {'laugh': 4211, 'slut': 4139, 'day': 3583, 'good': 3325, 'loud': 3169, 'online': 2984, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
disgust -> {'laugh': 4478, 'slut': 3916, 'loud': 3099, 'online': 2746, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'trust': 3097, 'joy': 7439}
fear -> {'laugh': 4507, 'slut': 4272, 'good': 3989, 'loud': 3097, 'lot': 3073, 'day': 2788, 'effort': 272, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
joy -> {'laugh': 7439, 'lot': 5581, 'good': 5551, 'effort': 5139, 'voluntary': 5134, 'loud': 4650, 'slut': 4997, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
sadness -> {'slut': 4644, 'online': 3627, 'laugh': 3563, 'good': 2863, 'feel': 2747, 'day': 2687, 'work': 2586, 'text': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
surprise -> {'laugh': 6893, 'slut': 4997, 'loud': 4608, 'ass': 2586, 'online': 2428, 'eye': 2218, 'text': 2586, 'work': 2586, 'day': 2586, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
trust -> {'lot': 10265, 'effort': 9980, 'voluntary': 9965, 'good': 5688, 'laugh': 4158, 'day': 2816, 'loud': 2648, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
```

```
Mongo_DB ha richiesto per il conteggio di word all'incirca 10 secondi
anger -> {'laugh': 3668, 'slut': 3591, 'online': 2929, 'fuck': 2742, 'people': 2686, 'loud': 2648, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
anticipation -> {'laugh': 4211, 'slut': 4139, 'day': 3583, 'good': 3325, 'loud': 3169, 'online': 2984, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
disgust -> {'laugh': 4478, 'slut': 3916, 'loud': 3099, 'online': 2746, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'trust': 3097, 'joy': 7439}
fear -> {'laugh': 4507, 'slut': 4272, 'good': 3989, 'loud': 3097, 'lot': 3073, 'day': 2788, 'effort': 272, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
joy -> {'laugh': 7439, 'lot': 5581, 'good': 5551, 'effort': 5139, 'voluntary': 5134, 'loud': 4650, 'slut': 4997, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
sadness -> {'slut': 4644, 'online': 3627, 'laugh': 3563, 'good': 2863, 'feel': 2747, 'day': 2687, 'work': 2586, 'text': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439}
surprise -> {'laugh': 6893, 'slut': 4997, 'loud': 4608, 'ass': 2586, 'online': 2428, 'eye': 2218, 'text': 2586, 'work': 2586, 'day': 2586, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439, 'sadness': 4644}
trust -> {'lot': 10265, 'effort': 9980, 'voluntary': 9965, 'good': 5688, 'laugh': 4158, 'day': 2816, 'loud': 2648, 'text': 2586, 'work': 2586, 'day': 2586, 'eye': 2214, 'lot': 20265, 'ass': 2586, 'voluntary': 9965, 'good': 5688, 'surprise': 6893, 'sadness': 4644, 'fear': 4507, 'anticipation': 4211, 'disgust': 4478, 'trust': 3097, 'joy': 7439, 'sadness': 4644}
```

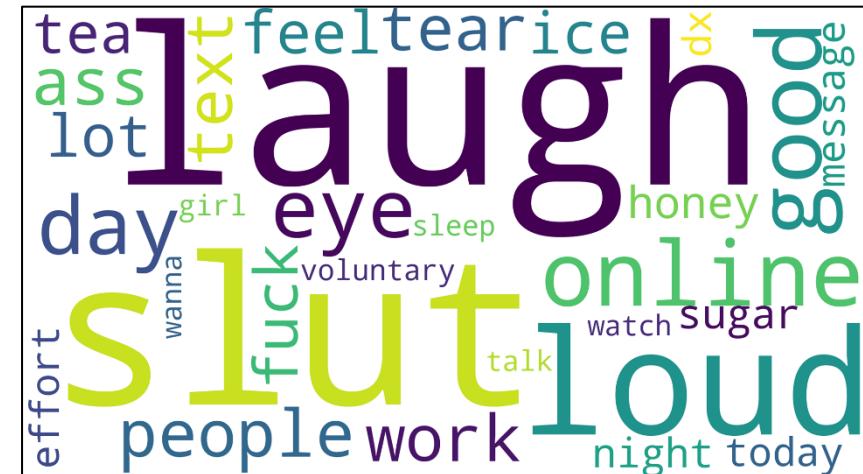


Risultati: Word Cloud (lemmi)

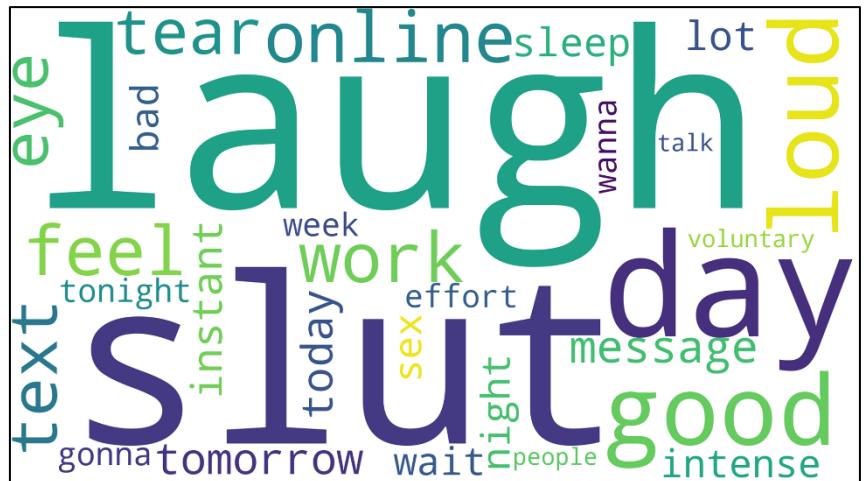
ANGER



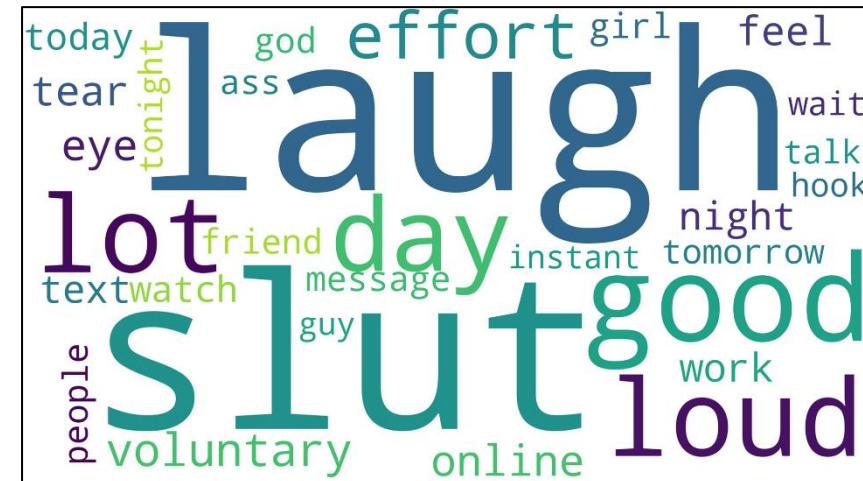
DISGUST



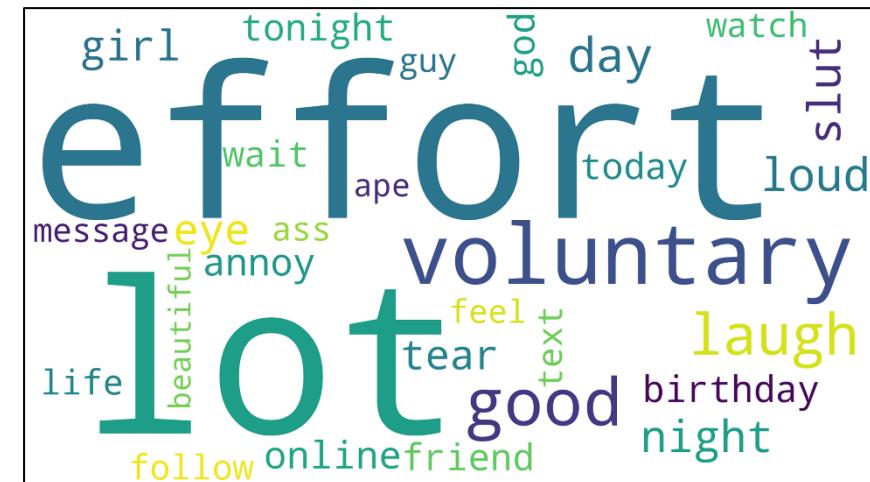
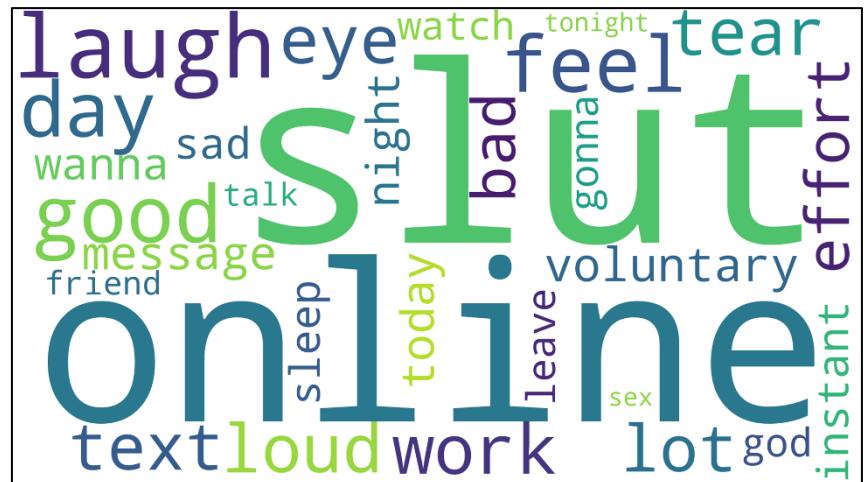
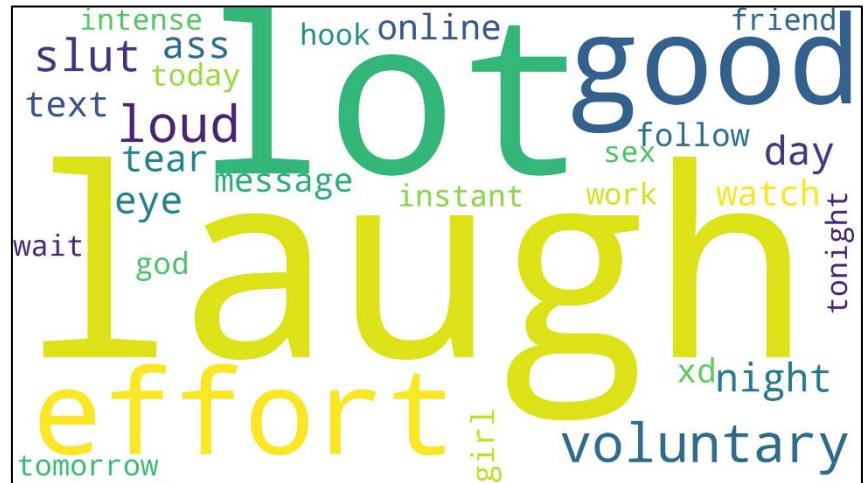
ANTICIPATION



FEAR



Risultati: Word Cloud (lemmi)

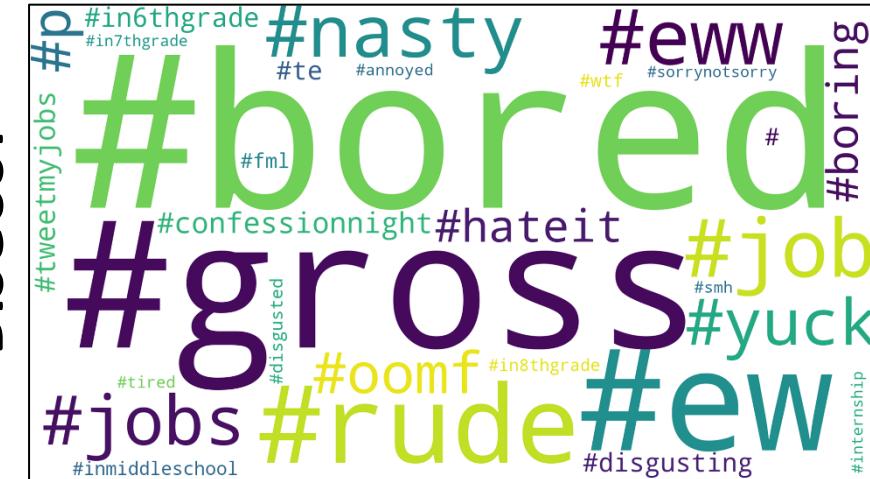


Risultati: Word Cloud (hashtag)

ANGER



DISGUST



ANTICIPATION

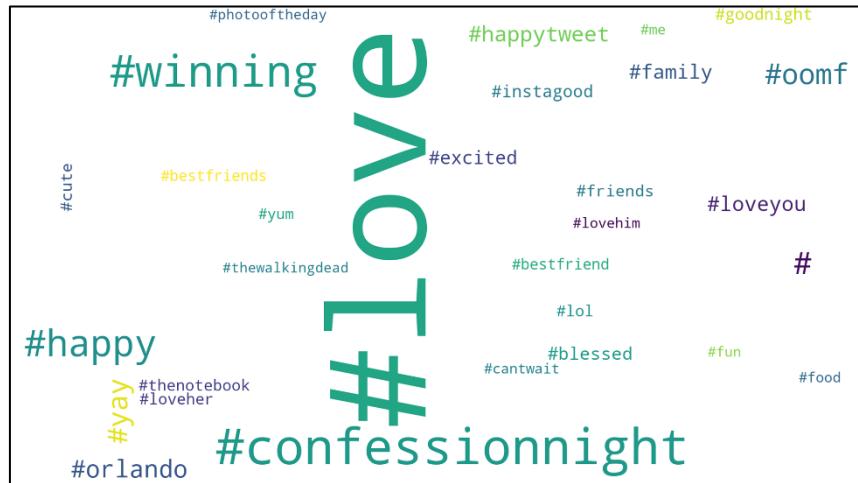


FEAR

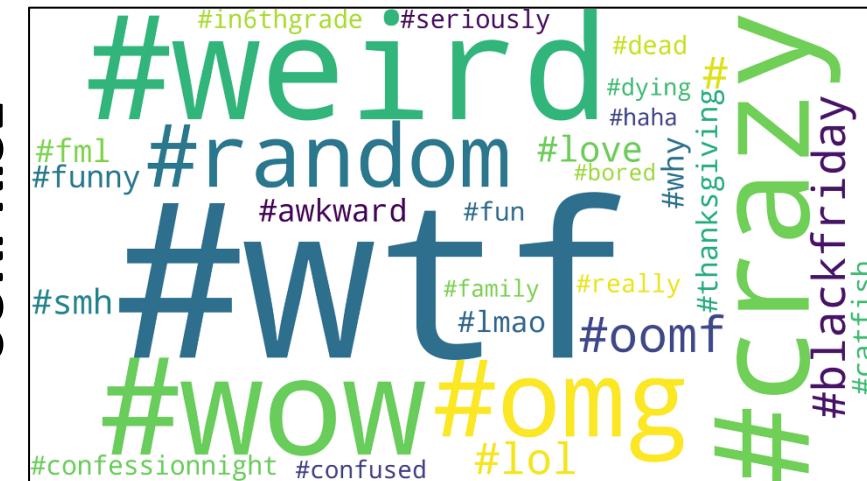


Risultati: Word Cloud (hashtag)

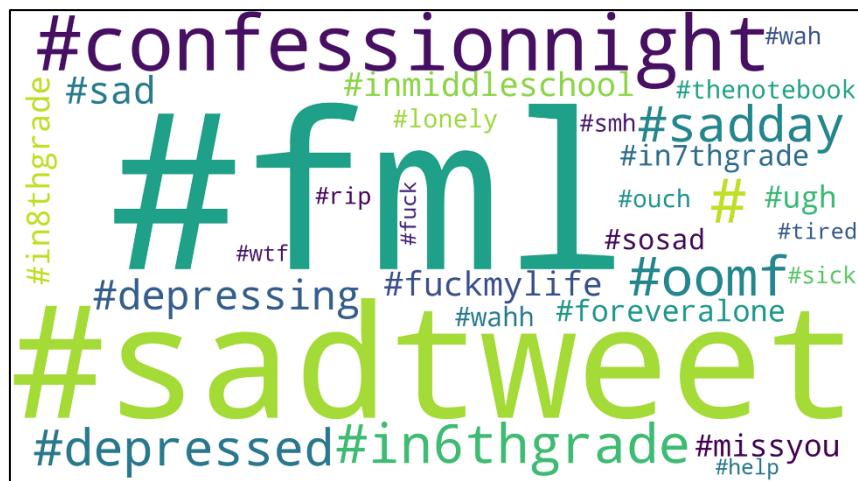
JOY



SURPRISE



SADNESS

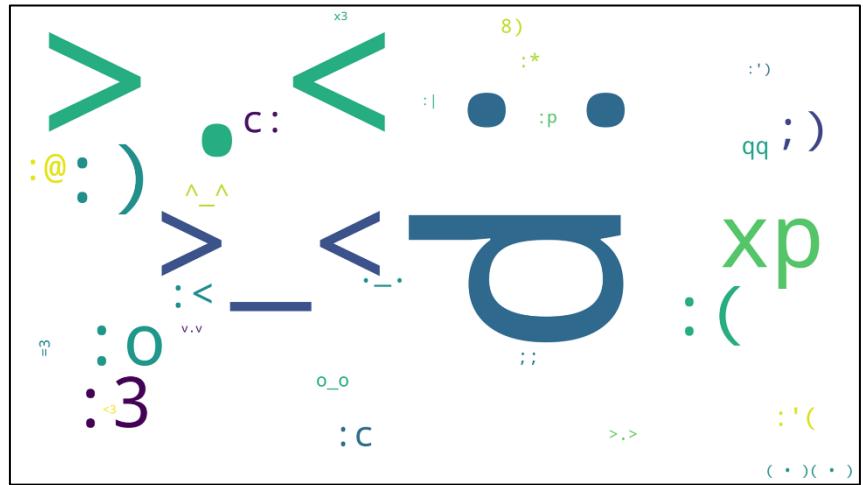


TRUST

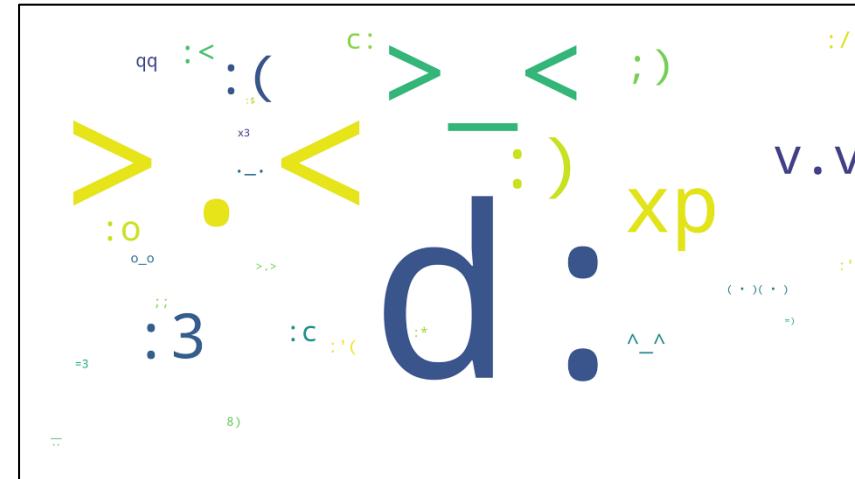


Risultati: Word Cloud (emoticon)

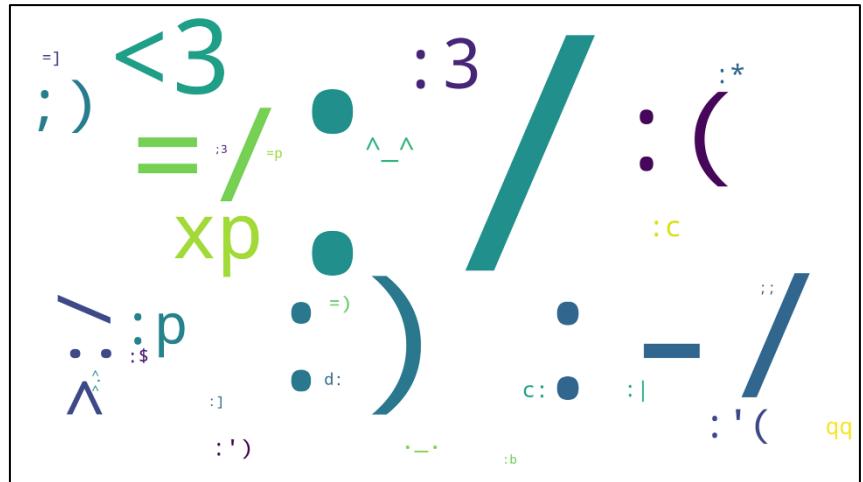
ANGER



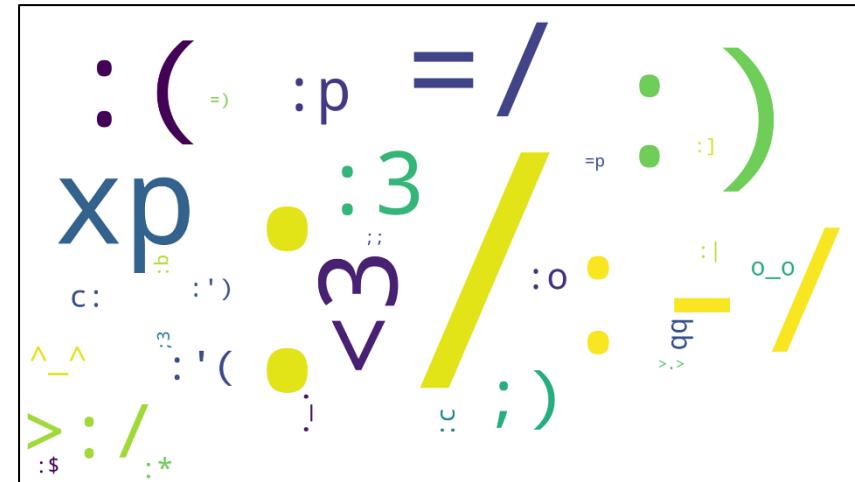
DISGUST



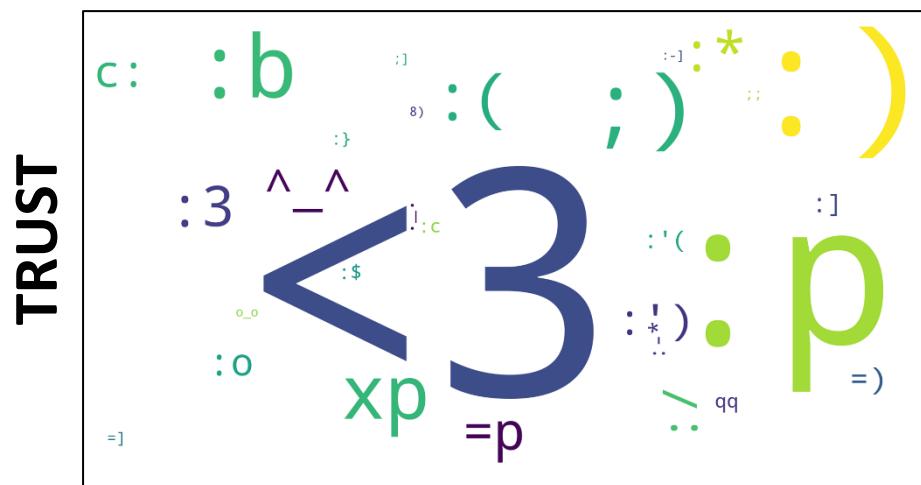
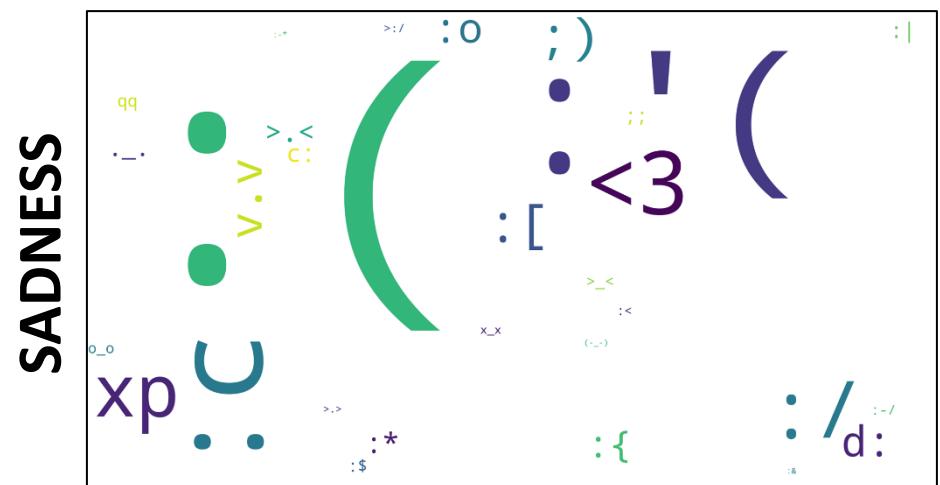
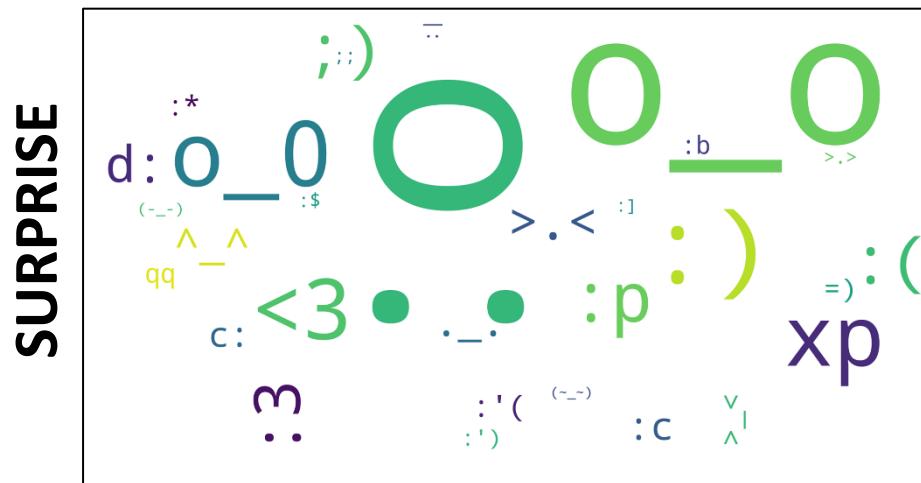
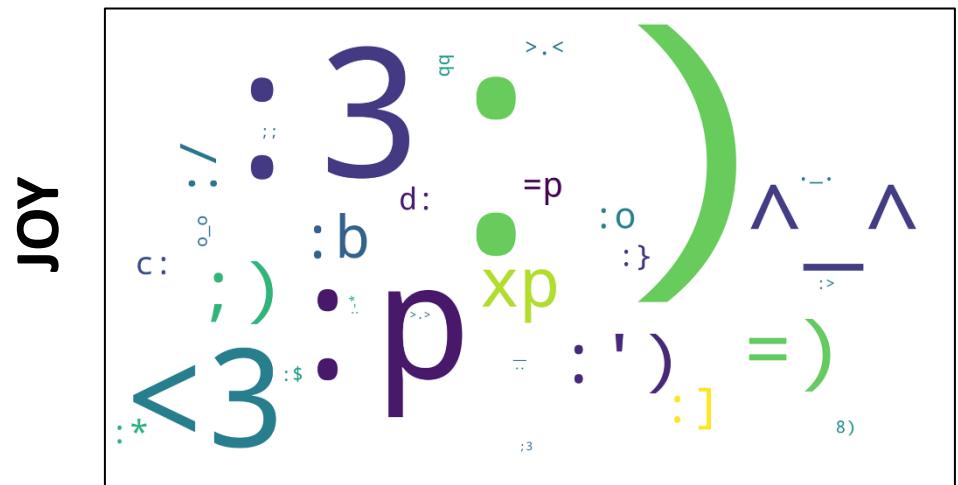
ANTICIPATION



FEAR



Risultati: Word Cloud (emoticon)



Risultati: Word Cloud (emoji)



Risultati: Word Cloud (emoji)

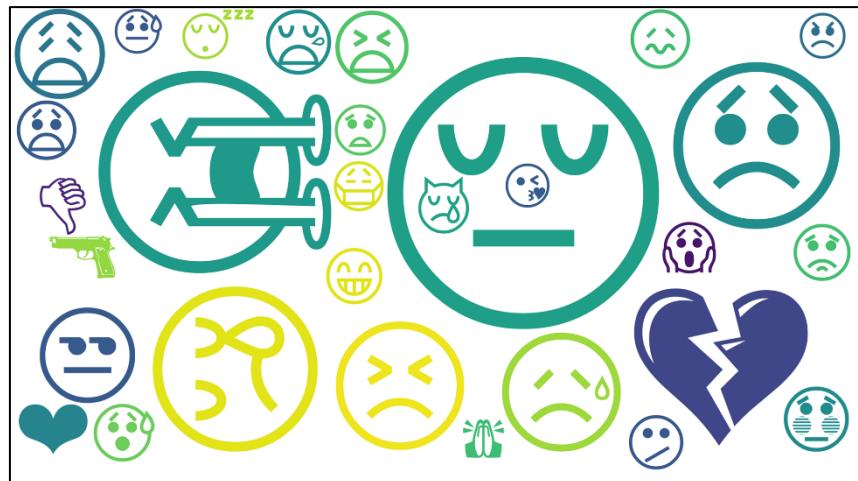
JOY



SURPRISE



SADNESS



TRUST



Tempistiche Goal-2

MYSQL_DB ha richiesto per il calcolo delle percentuali inerenti la sovrapposizione tra messaggi Twitter e risorse lessicali all'incirca 22.61 secondi
perc_presence_lex_res perc_presence_twitter

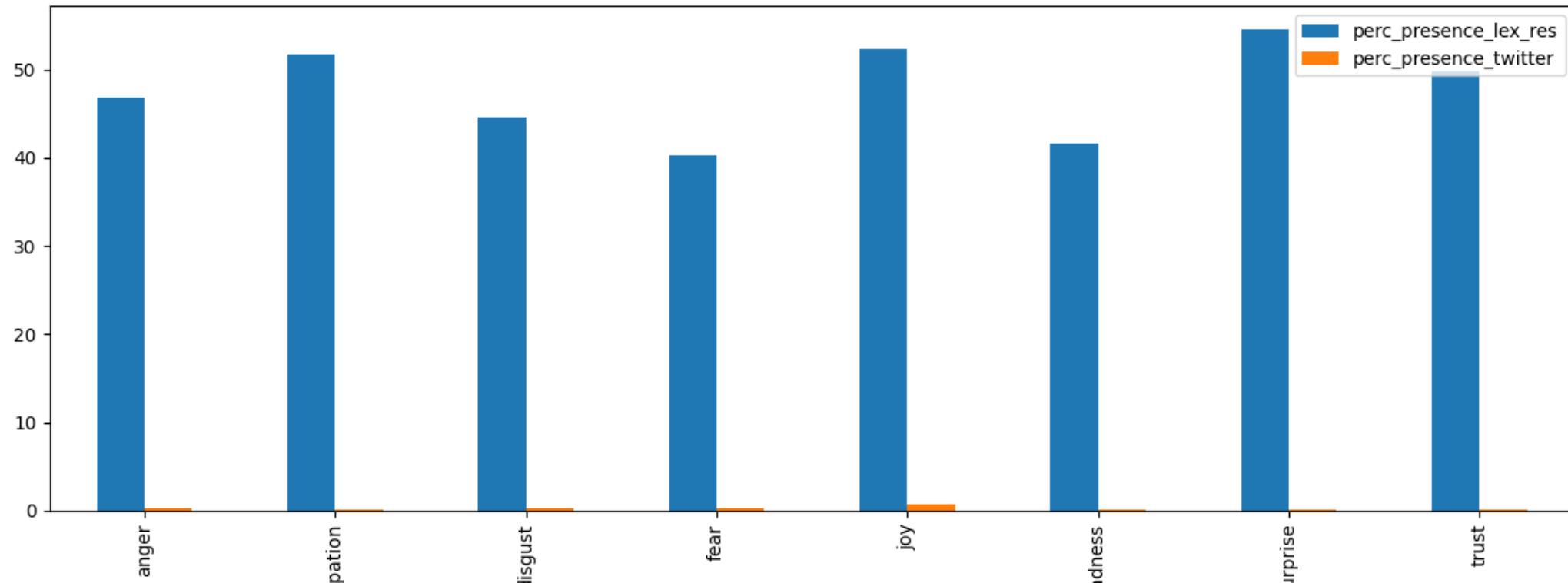
anger	46.79%	0.25%
anticipation	51.71%	0.18%
disgust	44.59%	0.24%
fear	40.23%	0.23%
joy	52.35%	0.78%
sadness	41.66%	0.19%
surprise	54.51%	0.1%
trust	49.79%	0.21%

NOSQL_DB ha richiesto per il calcolo delle percentuali inerenti la sovrapposizione tra messaggi Twitter e risorse lessicali all'incirca 11.09 secondi
perc_presence_lex_res perc_presence_twitter

anger	46.79%	0.25%
anticipation	51.71%	0.18%
disgust	44.59%	0.24%
fear	40.23%	0.23%
joy	52.35%	0.78%
sadness	41.66%	0.19%
surprise	54.51%	0.1%
trust	49.79%	0.21%



Risultati: Istogrammi Percentuali



Nuova Risorsa Lessicale

Di seguito vengono elencate l'elenco di parole presenti nella sorgente **Tweet** ma non nelle **risorse lessicali**: così abbiamo costruito una nuova risorsa lessicale, adatta a rappresentare i messaggi tweet

Sentiment	Lemma	POS	Freq
anger	eye	NNS	1975
anger	ass	NN	1911
...

Sentiment	Lemma	POS	Freq
anticipation	eye	NNS	2387
anticipation	day	NN	2266
...

Sentiment	Lemma	POS	Freq
disgust	eye	NNS	2049
disgust	ass	NN	1958
...

Sentiment	Lemma	POS	Freq
fear	eye	NNS	2199
fear	day	NN	2036
...

Sentiment	Lemma	POS	Freq
joy	night	NN	2374
joy	eye	NNS	2263
...

Sentiment	Lemma	POS	Freq
sadness	eye	NNS	2296
sadness	tear	NNS	2145
...

Sentiment	Lemma	POS	Freq
surprise	tear	NNS	1706
surprise	ice	NN	1618
...

Sentiment	Lemma	POS	Freq
trust	day	NN	2265
trust	night	NN	2169
...



Piano della Presentazione

- Analisi dei requisiti
- Progettazione e Aspetti Implementativi
- Risultati
- Conclusioni



Considerazioni finali

- Grazie al superamento dell'impedence mismatch, in fase di caricamento MongoDB offre delle agevolazioni. Inoltre, grazie alla possibilità di aggiungere nodi in un cluster e di definire sharding, MongoDB offre vantaggi in termini di scalabilità orizzontale in scrittura.
- In fase di caricamento, da un punto di vista empirico, abbiamo notato performances maggiori rispetto alla controparte MySQL, per via della aggregate orientation, denormalizzazione e l'assenza di framework per l'ORM
- Inoltre, la replicabilità dei dati di MongoDB offre vantaggi in termini di scalabilità in lettura, recupero da guasti e ridondanza dei dati.
- D'altra parte, per applicazioni critiche (come ad esempio transazioni bancarie), i vantaggi del modello relazionale sono encomiabili grazie al soddisfacimento delle proprietà ACID. Inoltre, in fase di aggiornamento, lo schema normalizzato offre notevole efficienza.

