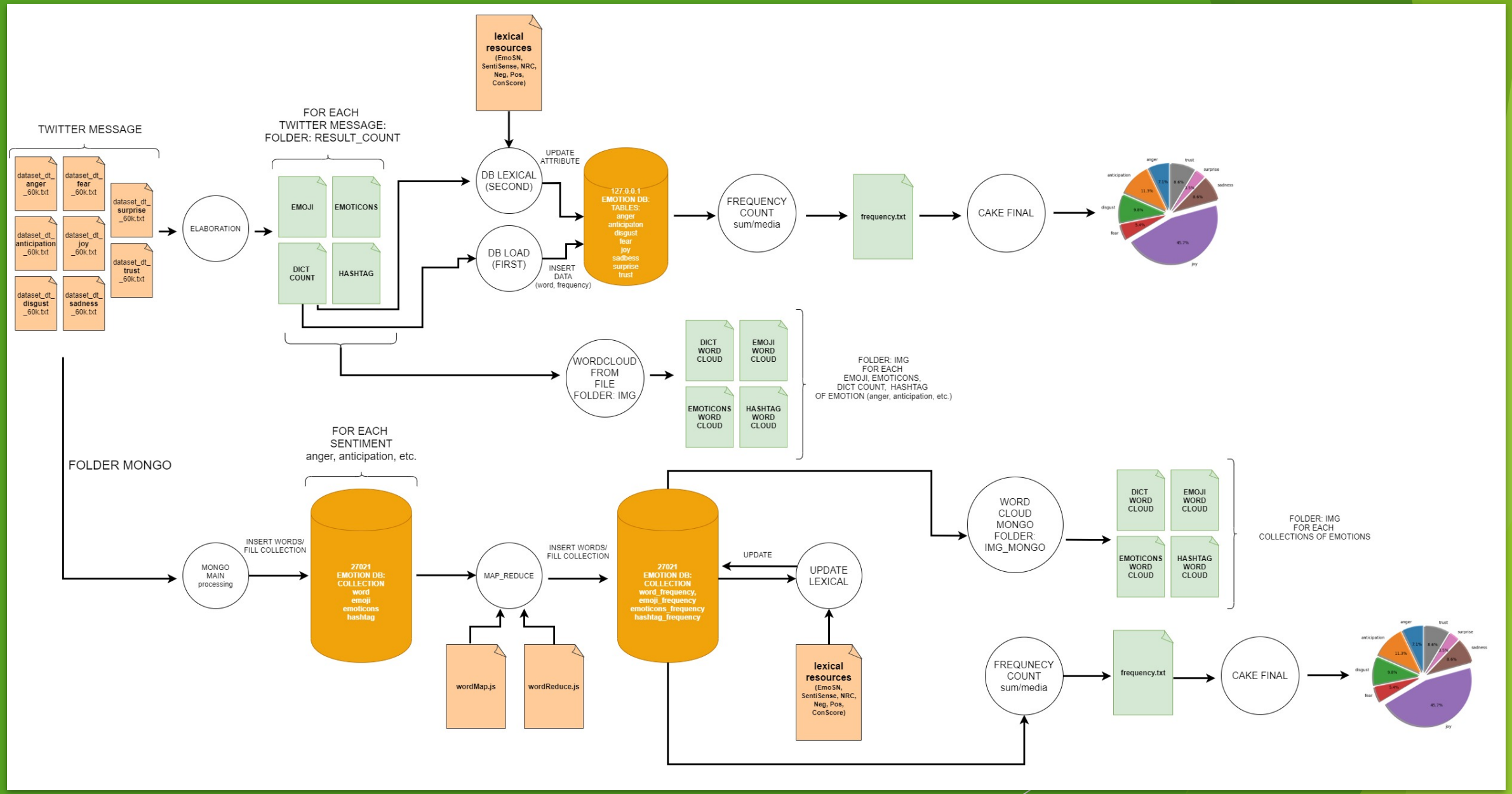
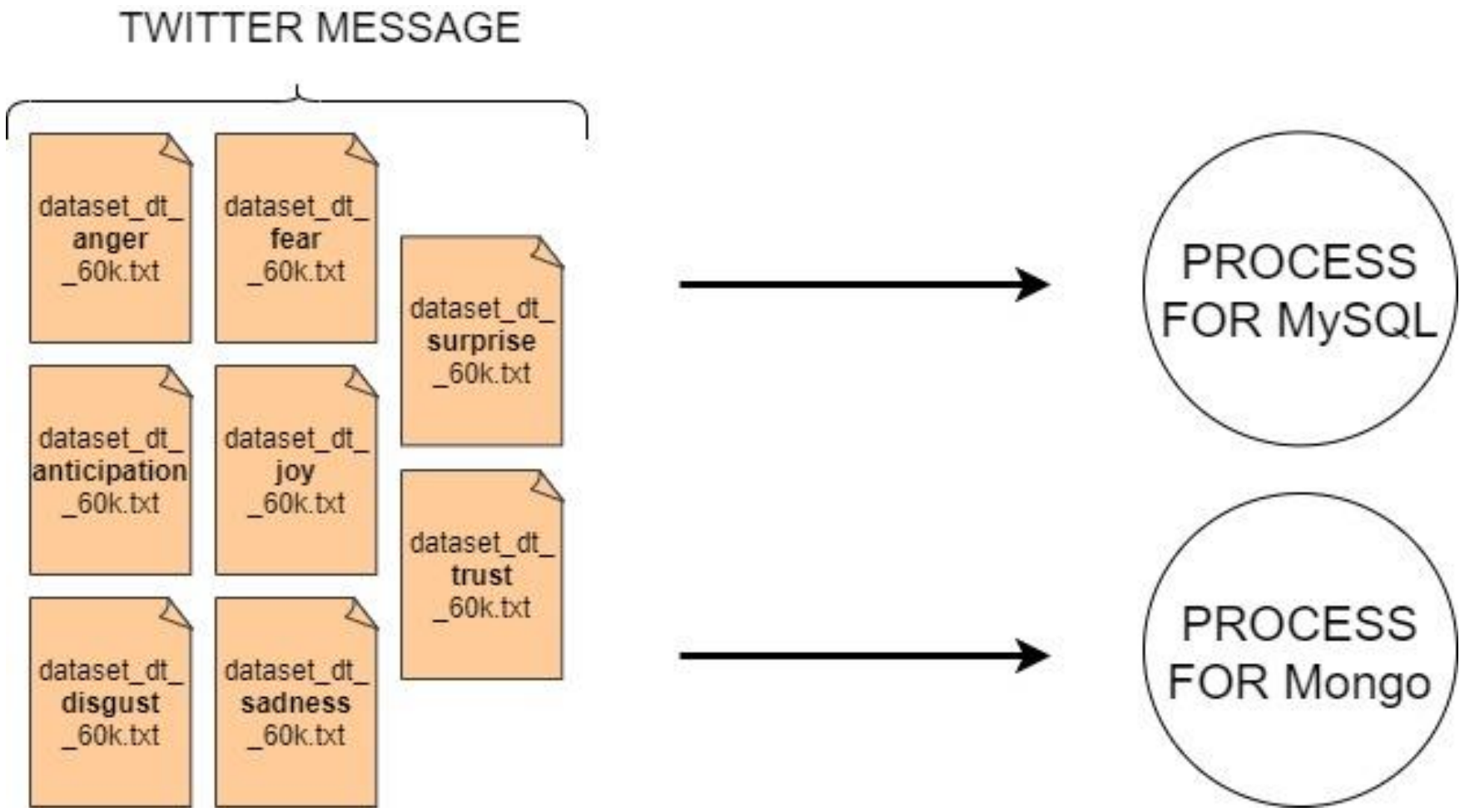
Progetto di Modelli e Architetture Avanzate di DataBase (MAADB)

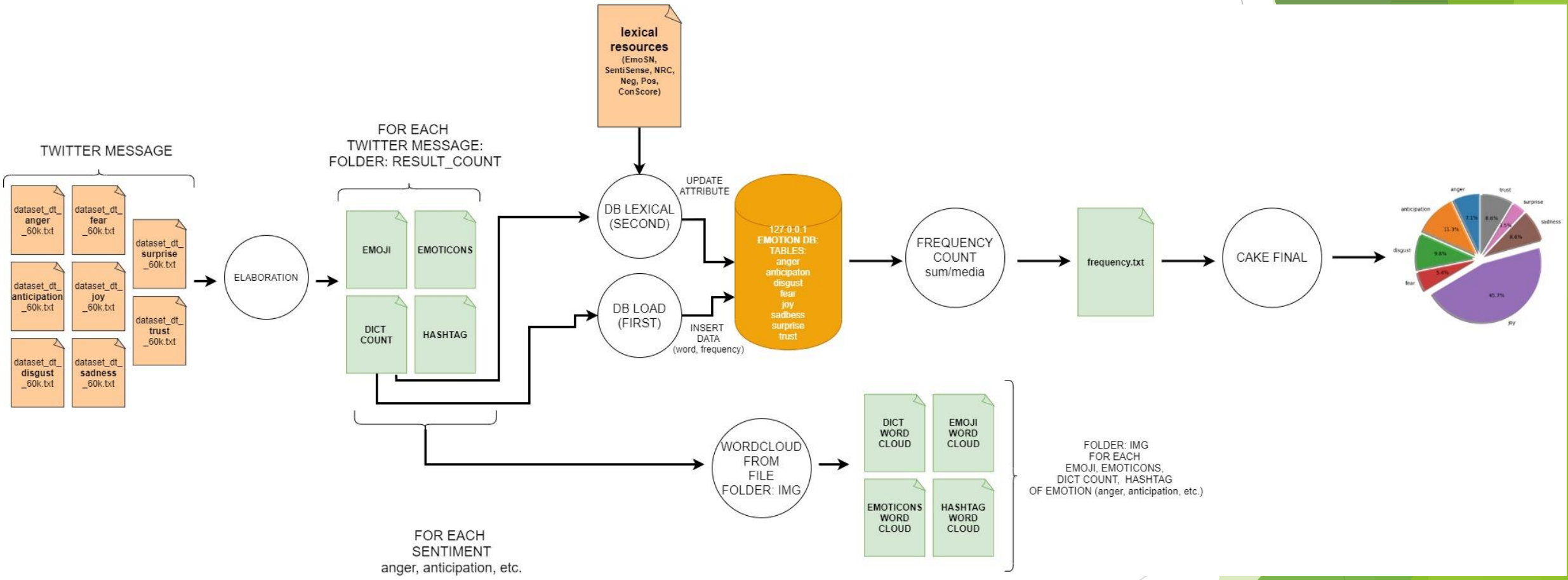
Bortolotti Simone
De Cenzo Davide
Marignati Luca

PROJECT SCHEMA

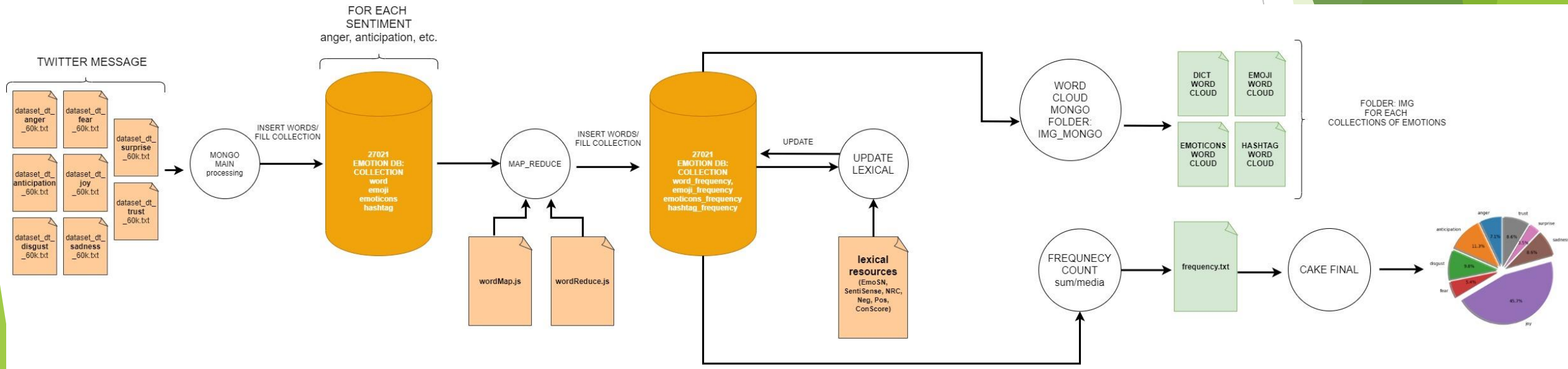

TWITTER MESSAGE
dataset_dt_ anger _60k.txt
dataset_dt_ fear _60k.txt
dataset_dt_ surprise _60k.txt
dataset_dt_ anticipation _60k.txt
dataset_dt_ joy _60k.txt
dataset_dt_ trust _60k.txt
dataset_dt_ disgust _60k.txt
dataset_dt_ sadness _60k.txt
ELABORATION
FOR EACH TWITTER MESSAGE: FOLDER: RESULT_COUNT
EMOJI
EMOTICONS
DICT COUNT
HASHTAG
lexical resources (EmoSN, SentiSense, NRC, Neg, Pos, ConScore)
DB LEXICAL (SECOND)
UPDATE ATTRIBUTE
DB LOAD (FIRST)
INSERT DATA (word, frequency)
127.0.0.1 EMOTION DB: TABLES: anger anticipaton disgust fear joy sadbess surprise trust
FREQUENCY COUNT sum/media
frequency.txt
CAKE FINAL
WORDCLOUD FROM FILE FOLDER: IMG
DICT WORD CLOUD
EMOJI WORD CLOUD
EMOTICONS WORD CLOUD
HASHTAG WORD CLOUD
FOLDER: IMG FOR EACH EMOJI, EMOTICONS, DICT COUNT, HASHTAG OF EMOTION (anger, anticipation, etc.)
FOLDER MONGO
FOR EACH SENTIMENT anger, anticipation, etc.
MONGO MAIN processing
INSERT WORDS/ FILL COLLECTION
27021 EMOTION DB: COLLECTION word emoji emoticons hashtag
MAP_REDUCE
wordMap.js
wordReduce.js
INSERT WORDS/ FILL COLLECTION
27021 EMOTION DB: COLLECTION word_frequency, emoji_frequency emoticons_frequency hashtag_frequency
UPDATE
UPDATE LEXICAL
lexical resources (EmoSN, SentiSense, NRC, Neg, Pos, ConScore)
WORD CLOUD MONGO FOLDER: IMG_MONGO
DICT WORD CLOUD
EMOJI WORD CLOUD
EMOTICONS WORD CLOUD
HASHTAG WORD CLOUD
FOLDER: IMG FOR EACH COLLECTIONS OF EMOTIONS
FREQUNECY COUNT sum/media
frequency.txt
CAKE FINAL


TWITTER MESSAGE
dataset_dt_anger_60k.txt
dataset_dt_fear_60k.txt
dataset_dt_surprise_60k.txt
dataset_dt_anticipation_60k.txt
dataset_dt_joy_60k.txt
dataset_dt_trust_60k.txt
dataset_dt_disgust_60k.txt
dataset_dt_sadness_60k.txt
PROCESS FOR MySQL
PROCESS FOR Mongo

Process for MySQL



Process for Mongo



FILE SYSTEM

Main folder

archivi_risorse_lessicali	• E.g. Anger, Anticipation, conScore, etc.
twitter_message	• Initial DATASET (e.g. dataset_dt_anger_60k.txt, etc.)
img	• Wordcloud (MySQL) • 4 for each emotion: word + hashtag + emoji + emoticons
img_mongo	• wordcloud (Mongo collection) • 4 for each emotion: word + hashtag + emoji + emoticons
result_count	• PRE-LOAD DB: results of ELABORATION.PY (NTLK) • 4 for each emotion: word + hashtag + emoji + emoticons
Mongo	• Mongo elaboration

Main file (1/2) - MySQL

ELABORATION.PY

- INPUT: twitter message
- processing twitter message (string + nltk operation) + frequency count
- save to local array/dictionary (emoji, emotions, hashtag, word)
- OUTPUT: save local array/dictionary to file into folder «result_count»

DB_LOAD.PY

- INPUT: read file from folder «result_count»
- OUTPUT: insert data to db («emotion»)
  - Table = "anger", "anticipation", "disgust", etc.

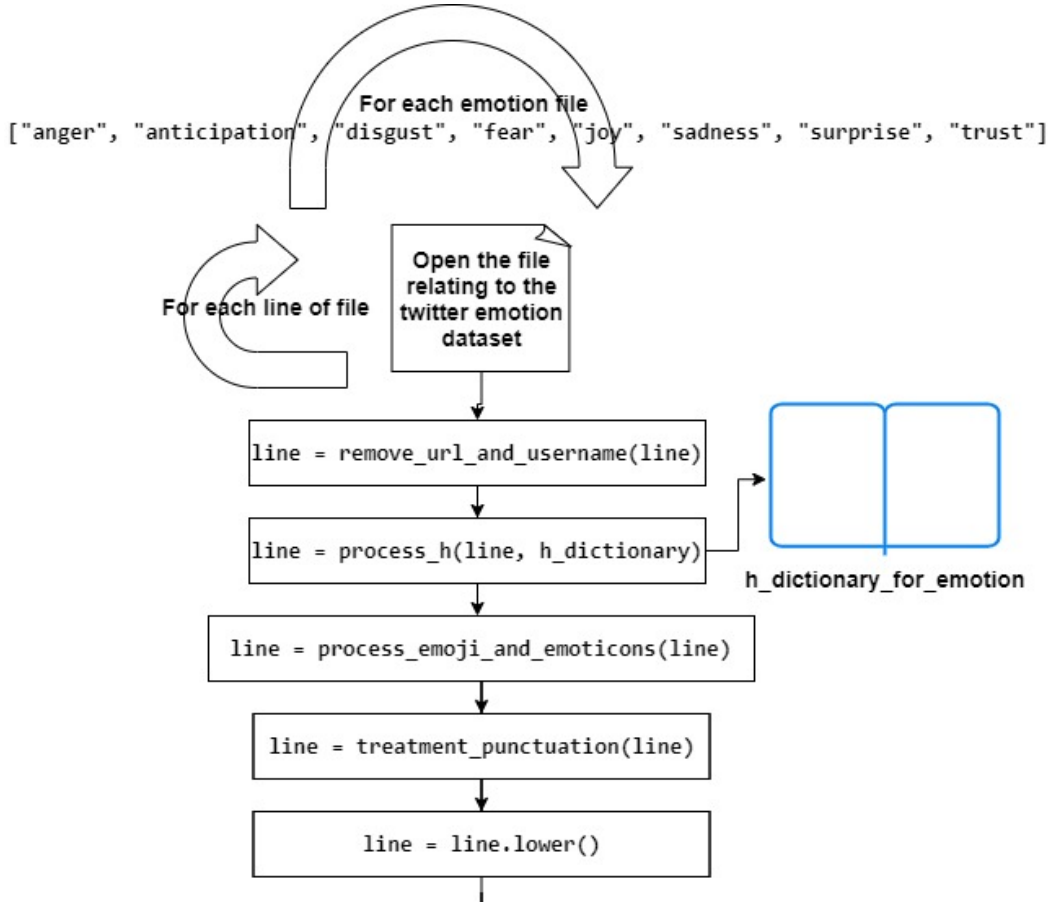
DB_LEXICAL.PY

- INPUT: read file from folder «lexical resources»
- OUTPUT: Update table ("anger", "anticipation", "disgust", etc.)
- boolean values (emo_sn, …) + conScore (afinn, ...) + Neg/Pos (gi_neg, …)

Main file (2/2) - MySQL

FREQUENCY_COUNT.PY	• INPUT: «select sum from DB» • average calculation • OUTPUT: save to file «frequency.txt»
WORDCLOUD_FROM_FILE.PY	• INPUT: read result count (e.g anger_emoji.txt, etc.) • Create wordcloud IMG • OUTPUT: save file to folder «wordcloud» • 4 for each emotion: word + hashtag + emoji + emoticons

Elaboration.py (1/3) - example
Processing twitter message



Example (step by step):

(pretty legit) 👌 ♡ 😍 :) @ atm the DISNEY gallery URL #nofilter

(pretty legit) 👌 ♡ 😍 :) @ atm the DISNEY gallery #nofilter

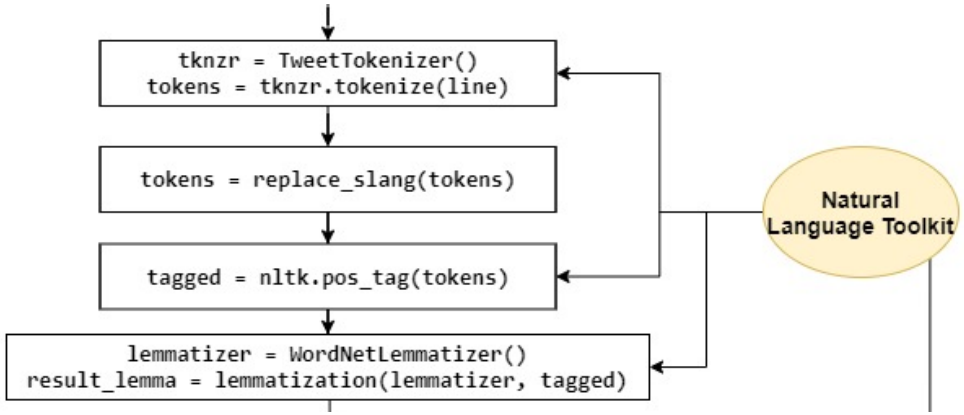
(pretty legit) 👌 ♡ 😍 :) @ atm the DISNEY gallery

(pretty legit) @ atm the DISNEY gallery

pretty legit @ atm the DISNEY gallery

pretty legit @ atm the disney gallery

Elaboration.py (2/3) – example
Processing twitter message



pretty legit @ atm the disney gallery

['pretty', 'legit', '@', 'atm', 'the', 'disney', 'galleries']

['pretty', 'legit', '@', 'at the moment', 'the', 'disney', 'galleries ']

[('pretty', 'RB'), ('legit', 'JJ'), ('@', 'NNP'),
('at the moment', 'VBD'), ('the', 'DT'),
('disney', 'NN'), ('galleries ', 'NNS')]

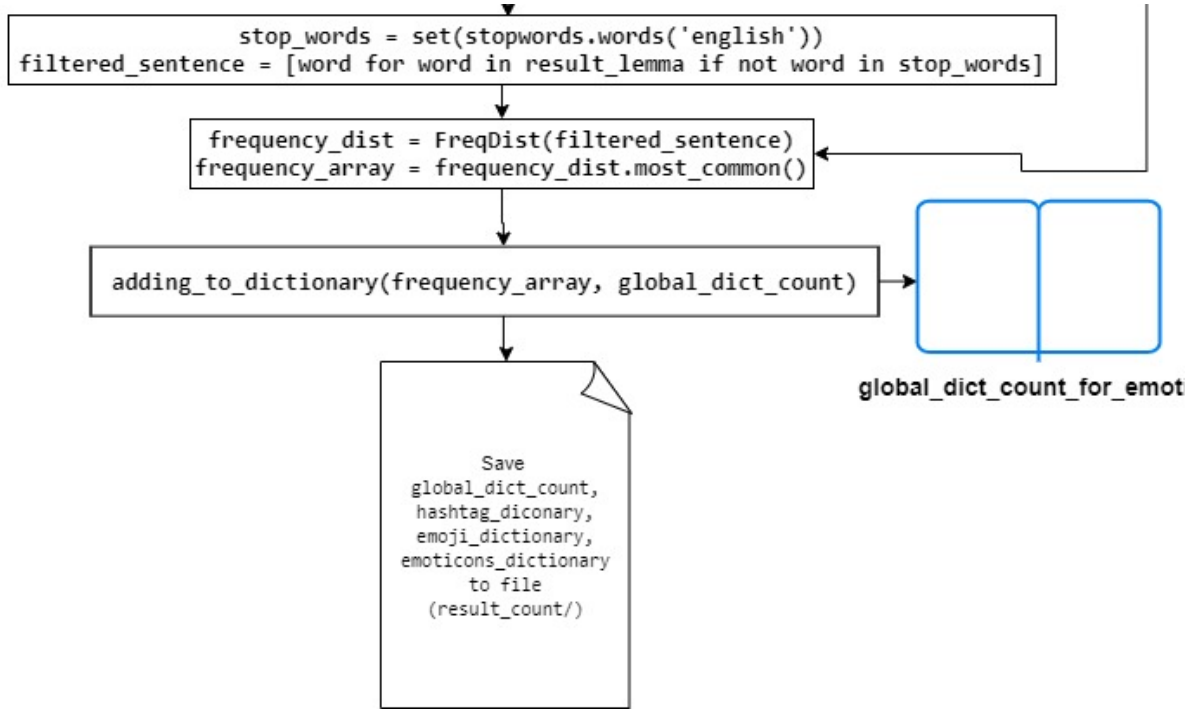
TAG	PART OF SPEECH (POS)
JJ	adjective
NNP	proper noun
VBD	verb, past tense took
DT	determiner
NNS	noun, plural
RB	adverbvery, silently

['pretty', 'legit', '@', 'at the moment', 'the', 'disney', 'gallery']

POS TAGGING

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	$
NNPS	proper noun, plural	*Carolinas*	#	pound sign	#
PDT	predeterminer	*all, both*	`	left quote	` or ``
POS	possessive ending	*'s*	'	right quote	' or ''
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Elaboration.py (3/3) – example
Processing twitter message



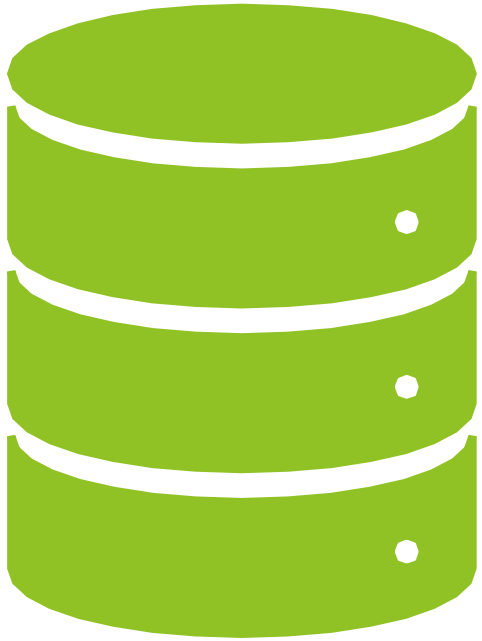
['pretty', 'legit', '@', 'at the moment', 'the', 'disney', 'gallery']

['pretty', 'legit', '@', 'at the moment', 'disney', 'gallery']

[('pretty', 1), ('legit', 1), ('at the moment', 1), ('disney', 1), ('gallery', 1)]

ADD TO GLOBAL_DICT_COUNT (SUM)

SAVE TO FILE
words, hashtag, emoji, emoticons
for each emotion



Data structure

MySQL

Nome	Tipo	Codifica caratteri
id	int(11)	
word	varchar(99)	utf8_general_ci
emo_sn	tinyint(4)	
nrc	tinyint(4)	
sentisense	tinyint(4)	
afinn	int(11)	
anew_aro	float	
anew_dom	float	
anew_pleas	float	
dal_activ	float	
dal_imag	float	
dal_pleas	float	
gi_neg	tinyint(4)	
hl_neg	tinyint(4)	
list_neg	tinyint(4)	
liwc_neg	tinyint(4)	
gi_pos	tinyint(4)	
hl_pos	tinyint(4)	
list_pos	tinyint(4)	
liwc_pos	tinyint(4)	
frequency	int(11)	

→ ID AUTO INCREMENT

→ WORD

Basic resources
(emo_sn, nrc, sentisense)

Resources with score
(conScore)

Negative resources
(Neg)

Negative resources
(Pos)

→ FREQUENCY

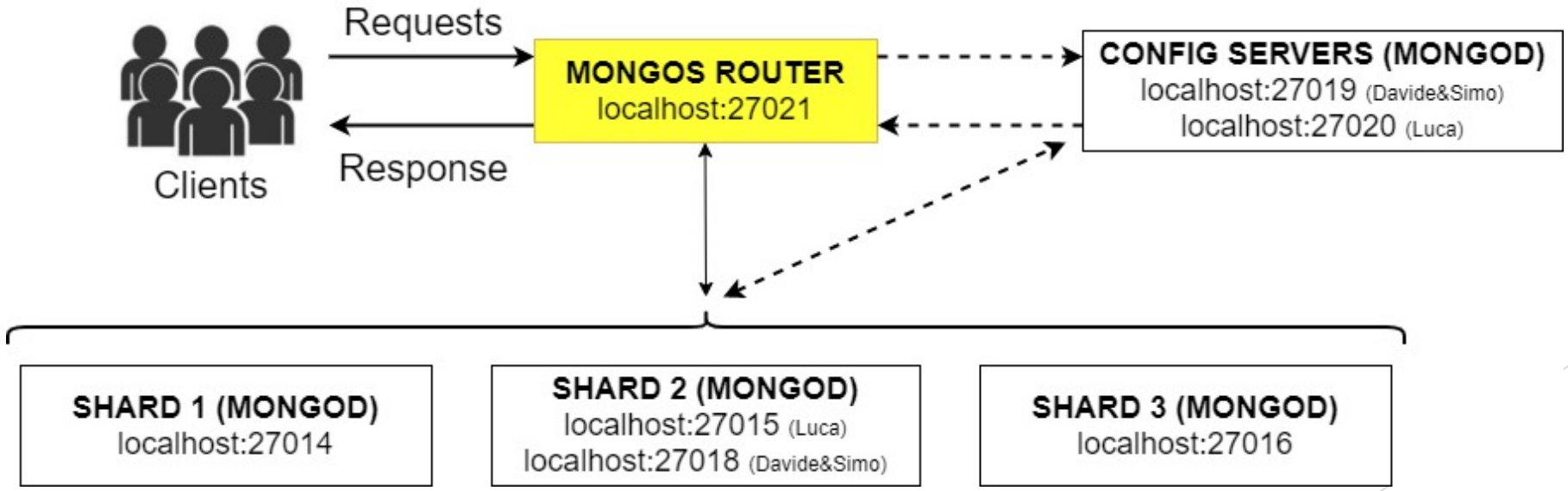
For each emotion: 8 table

MongoDB

CLUSTER ARCHITECTURE (1/3)

MONGOS ROUTER

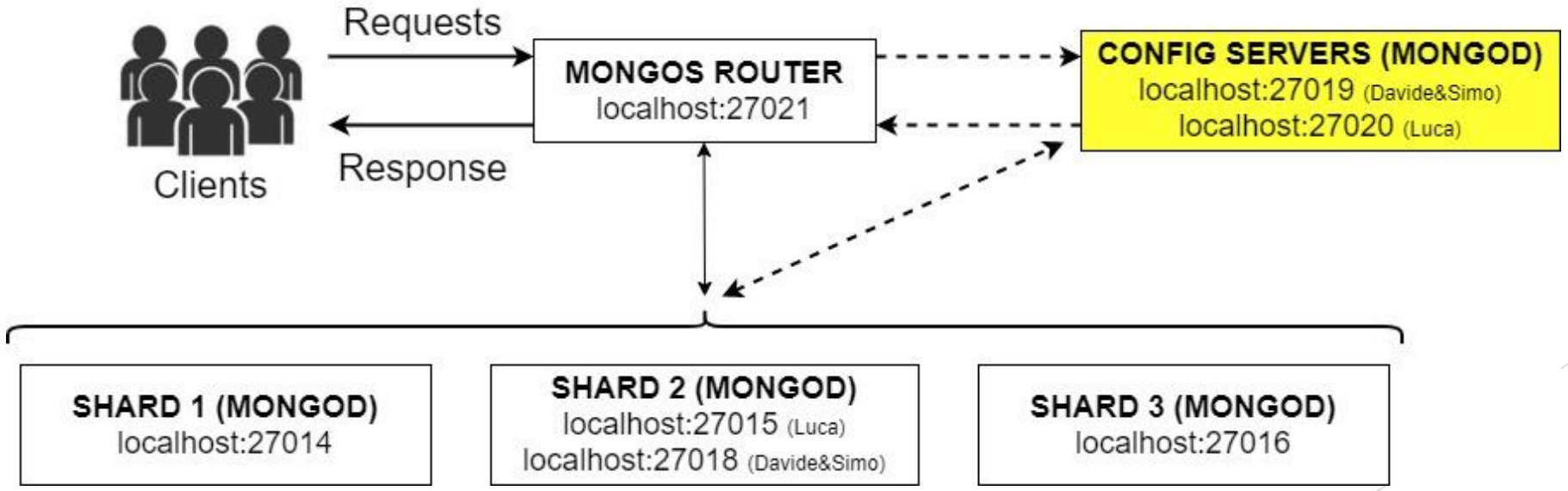
- Router/balancer
- Interfaccia alle applicazioni client;
- Le applicazioni esterne non devono preoccuparsi dell'architettura del sistema, ma possono solamente connettersi a questo nodo come se fosse un semplice database MongoDB;
- Tutte le query e le operazioni di scrittura vengono quindi inviate ai nodi router, che le smistano al cluster.



CLUSTER ARCHITECTURE (2/3)

CONFIG SERVERS (MONGOD)

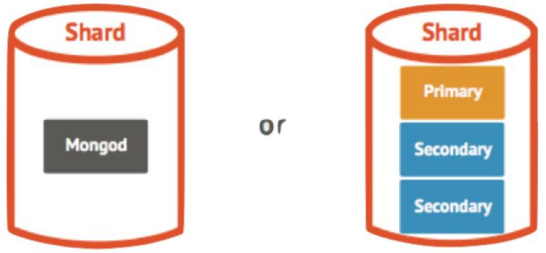
- Contiene informazioni sull'architettura del cluster;
- Si occupa della distribuzione, del reperimento dei dati e dello smistamento delle operazioni (a quale nodo inviare le richieste di lettura e di scrittura)
- Nota: per le architetture di produzione, si raccomanda di utilizzarne almeno tre nodi di tipo config servers. Utilizzarne uno solo è infatti un pericolo, perché in caso di guasto tutto il cluster diventa inutilizzabile.

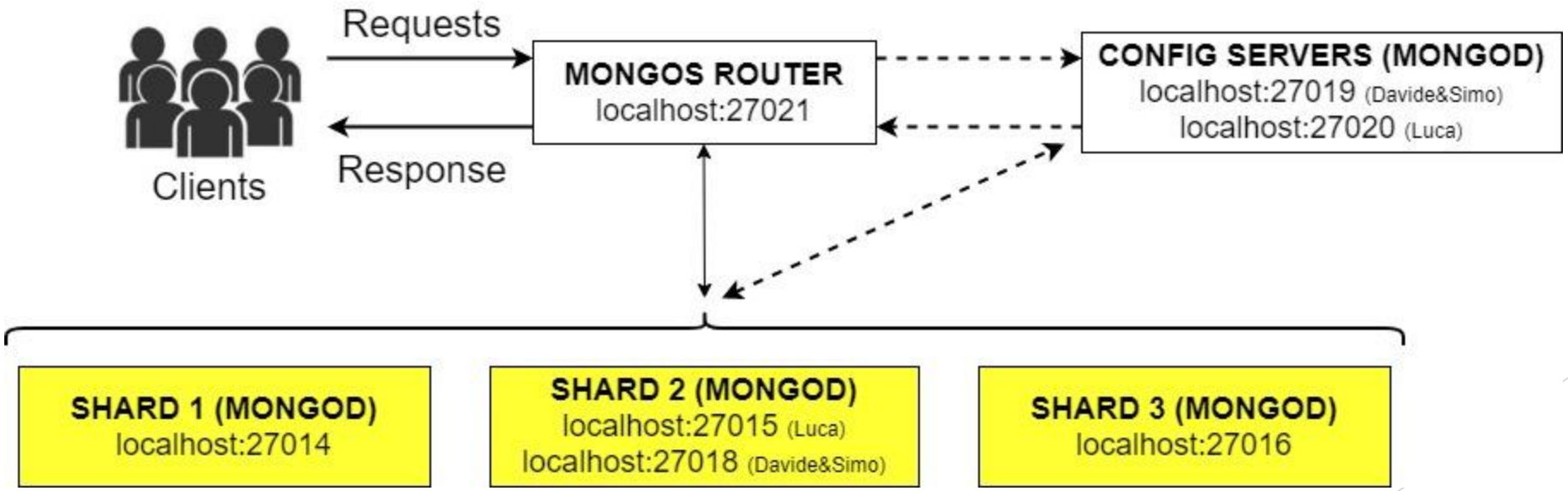


CLUSTER ARCHITECTURE (3/3)

SHARD (MONGOD)

- nodo di un cluster
- può essere un singolo mongod oppure un Replica Set





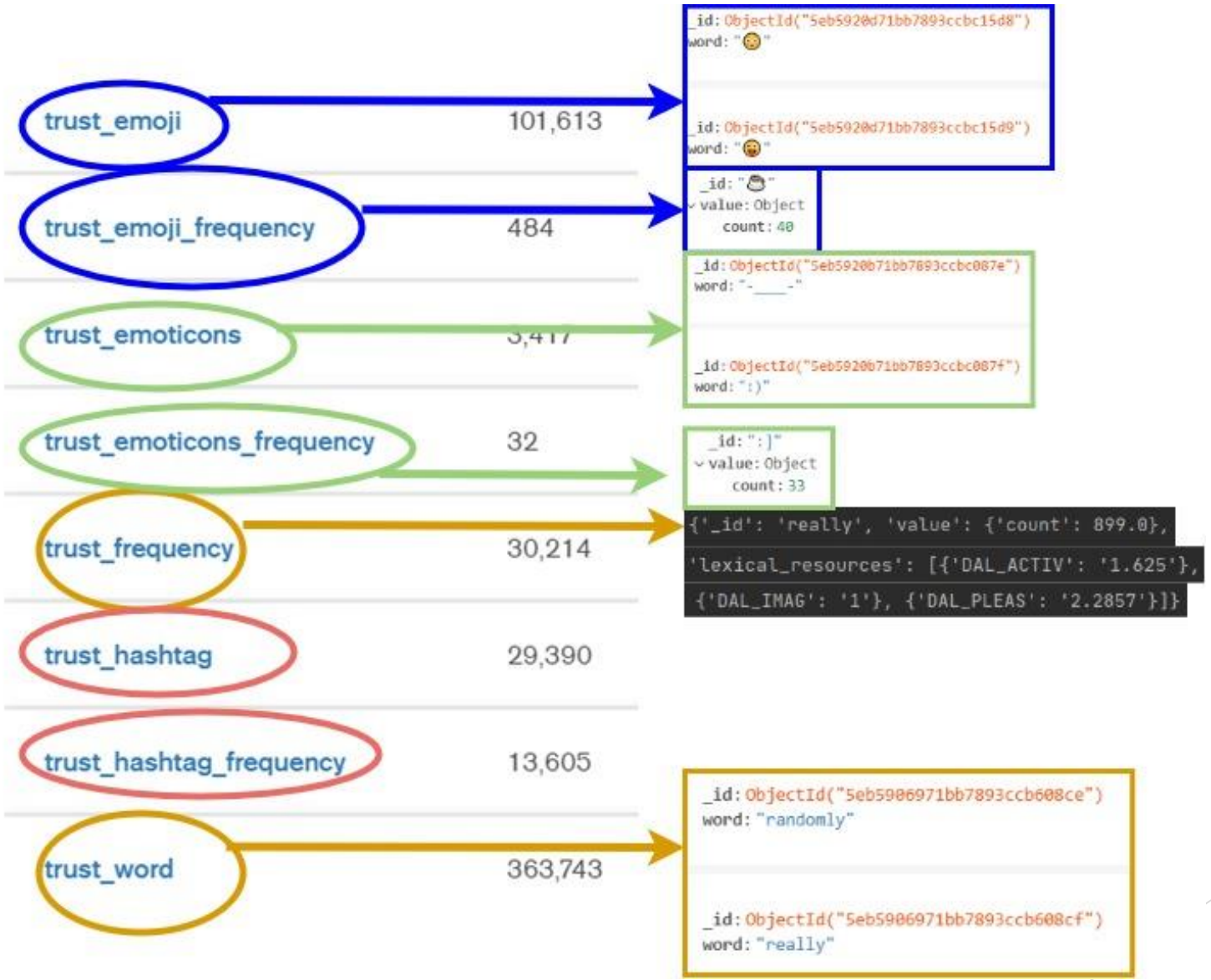
Sharding Data

Example

Collection: trust_word

```
mongos> db.trust_word.getShardDistribution()

Shard shard0001 at localhost:27015
 data : 5.55MiB docs : 153289 chunks : 2
 estimated data per chunk : 2.77MiB
 estimated docs per chunk : 76644

Shard shard0002 at localhost:27016
 data : 2.1MiB docs : 57980 chunks : 1
 estimated data per chunk : 2.1MiB
 estimated docs per chunk : 57980

Shard shard0000 at localhost:27014
 data : 5.55MiB docs : 152474 chunks : 2
 estimated data per chunk : 2.77MiB
 estimated docs per chunk : 76237

Totals
 data : 13.2MiB docs : 363743 chunks : 5
 Shard shard0001 contains 42.07% data, 42.14% docs in cluster, avg obj size on shard : 38B
 Shard shard0002 contains 15.9% data, 15.93% docs in cluster, avg obj size on shard : 38B
 Shard shard0000 contains 42.02% data, 41.91% docs in cluster, avg obj size on shard : 38B
```

COLLECTIONS



For each emotion:
8 collection

Document example – word: “Love”

{

'_id': 'love’,

'value': {'count': 10677.0},

'lexical_resources’: [

{'GI_POS': 1}, {'HL_POS': 1}, {'LIST_POS': 1}, {'LIWC_POS': 1},

{'AFINN': '3'}, {'ANEW_ARO': '6.44'},{'ANEW_DOM': '7.11’},{'ANEW_PLEAS': '8.72’},

{'DAL_ACTIV': '2.6364’}, {'DAL_IMAG': '1.4'}, {'DAL_PLEAS': '3’}

]

}

MAP REDUCE

map_reduce (1/5)

```python
import pymongo
from bson import Code

client = pymongo.MongoClient("mongodb://localhost:27021/?readPreference=primary&appname=MongoDB%20Compass%20Community"
                             "&ssl=false&retryWrites=false&w=majority")
db = client['emotion']

dataset_sentiment = ["anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", "trust"]

for emotion in dataset_sentiment:

    emotion_word = db[emotion + '_word']
    emotion_emoji = db[emotion + '_emoji']
    emotion_emoticons = db[emotion + '_emoticons']
    emotion_hashtag = db[emotion +'_hashtag']

    # Load map and reduce functions
    map = Code(open('wordMap.js', 'r').read())
    reduce = Code(open('wordReduce.js', 'r').read())

    results = emotion_word.map_reduce(map, reduce, emotion + "_frequency")
    emotion_emoji.map_reduce(map, reduce, emotion + "_emoji_frequency")
    emotion_emoticons.map_reduce(map, reduce, emotion + "_emoticons_frequency")
    emotion_hashtag.map_reduce(map, reduce, emotion + "_hashtag_frequency")
```

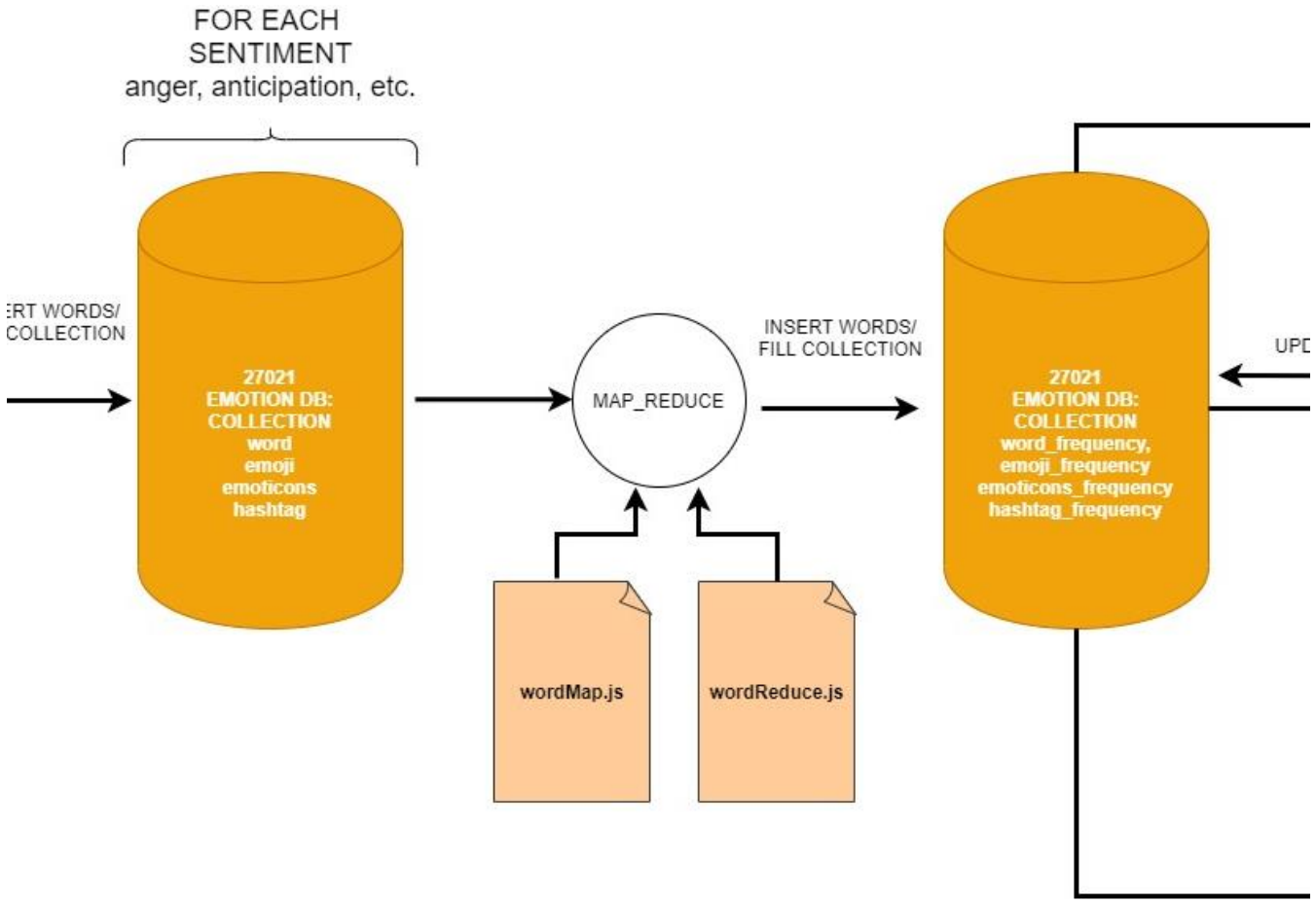
map_reduce (2/5)
wordMap.js

```
1  function wordMap(){
2
3      var words =  this.word.match(/^\s*\S+(\s?\S)*\s*$/g);
4
5      if(words == null) {
6          return;
7      }
8
9      for (var i = 0; i < words.length; i++){
10         emit(words[i], {count: 1});
11     }
12 }
```

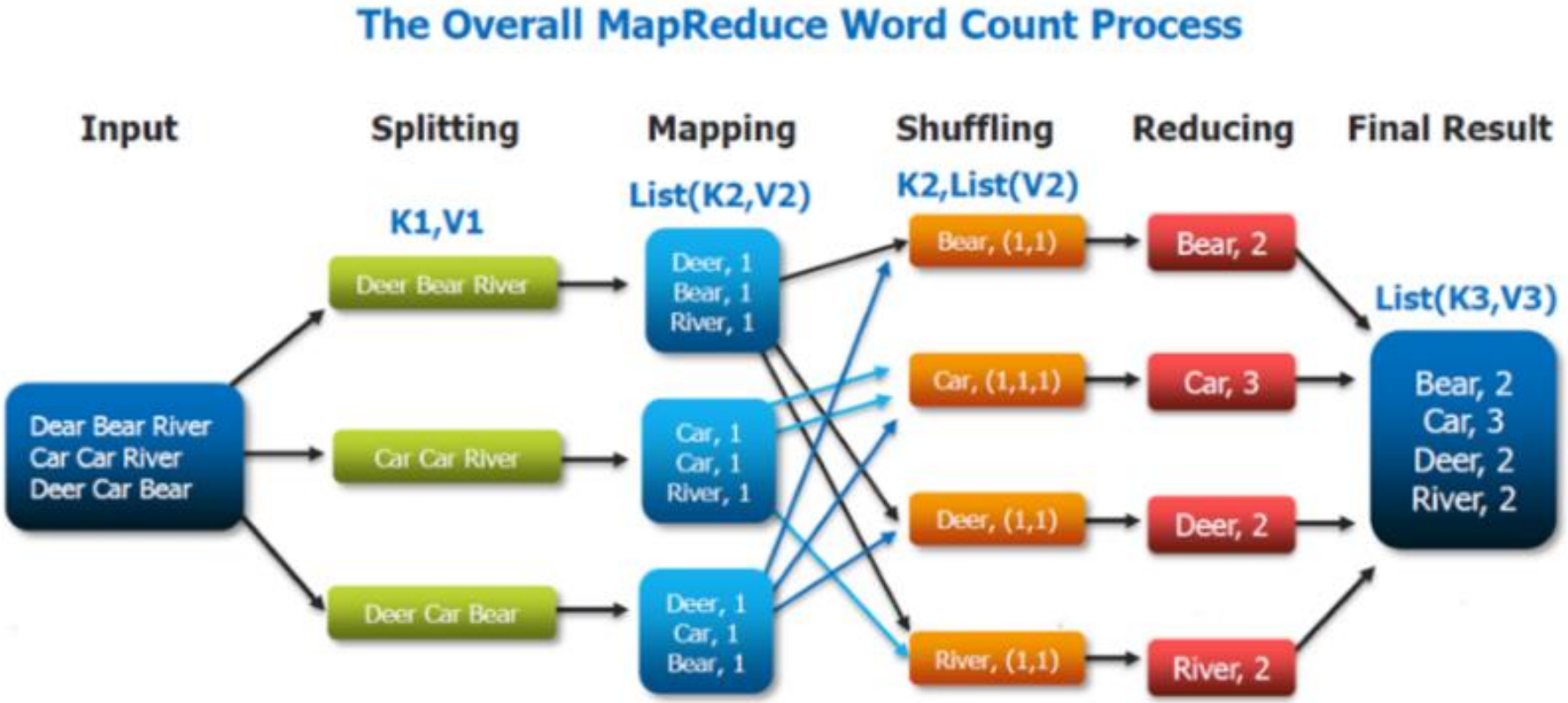
map_reduce (3/5)
wordReduce.js

```
1  function wordReduce(key, values){
2      var total = 0;
3      for (var i = 0; i < values.length; i++){
4          total += values[i].count;
5      }
6      return {count: total};
7  }
```

map_reduce (4/5)



map_reduce (5/5)



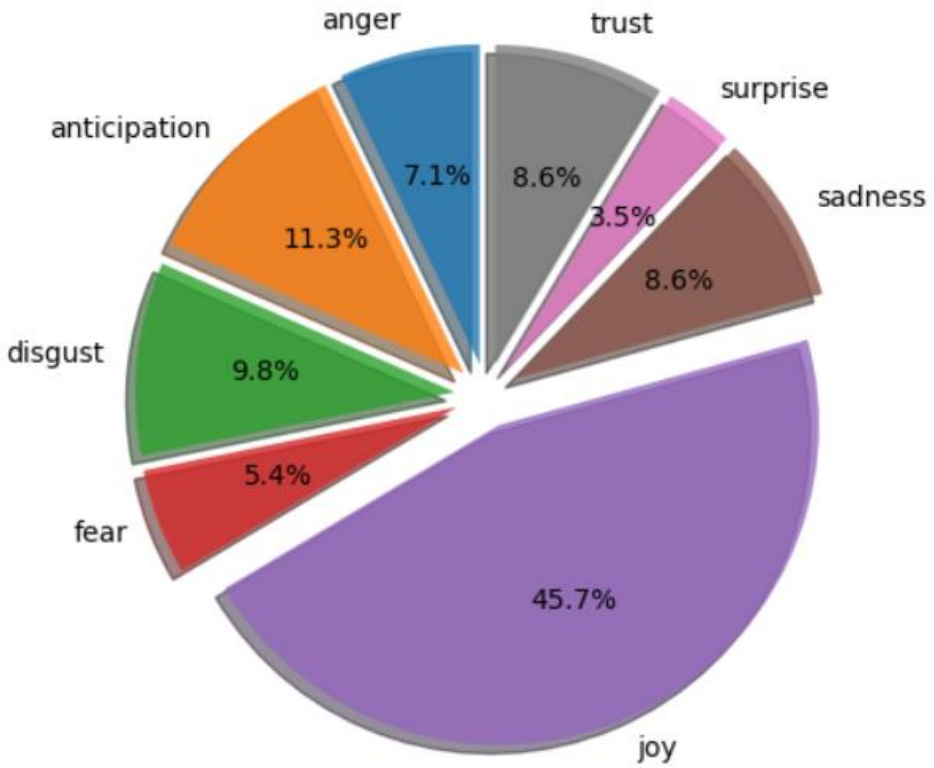
Results

Statistics of use of lexical resources (MySQL/Mongo) (1/2)

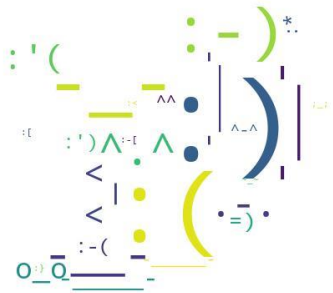
anger: 6.03 %	anticipation: 9.56 %	disgust: 8.27 %	fear: 4.55 %
joy: 38.74 %	sadness: 7.27 %	surprise: 2.96 %	trust: 7.30 %

media = 10.58 %

Statistics of use of lexical resources (MySQL/Mongo) (2/2)

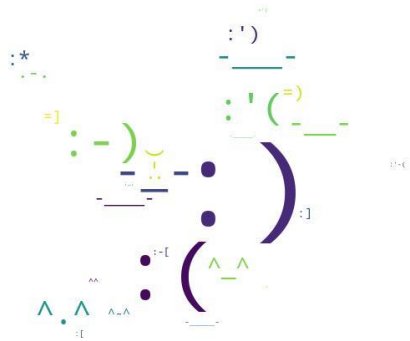


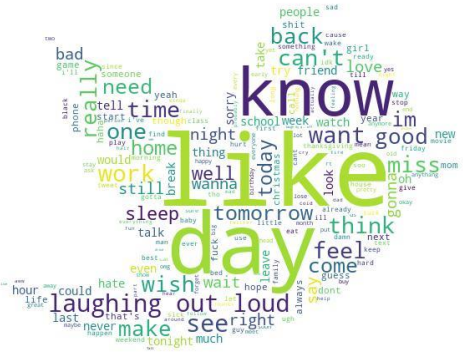
WORDCLOUD



Anger

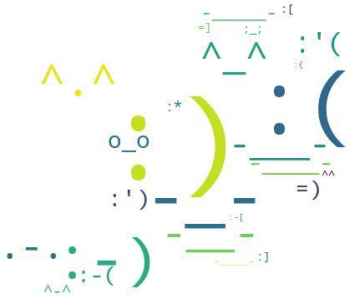
- EMOTICONS
- EMOJI
- HASHTAG
- WORDS

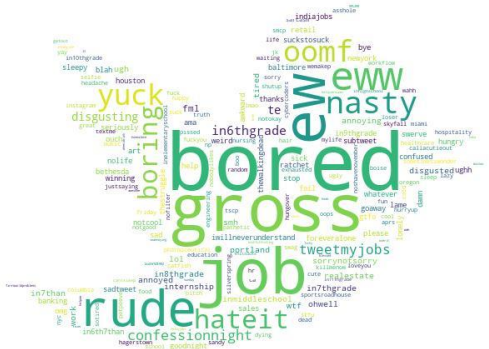




Anticipation

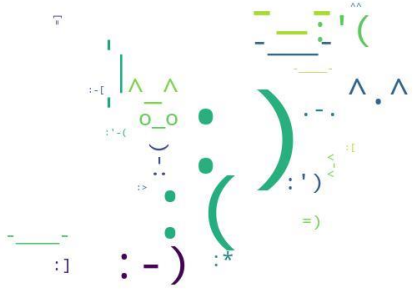
- EMOTICONS
- EMOJI
- HASHTAG
- WORDS





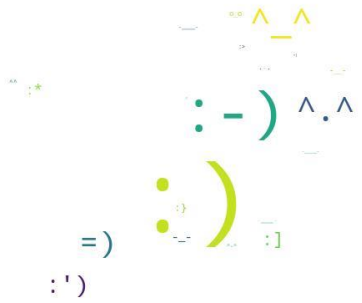
Disgust

- EMOTICONS
- EMOJI
- HASHTAG
- WORDS



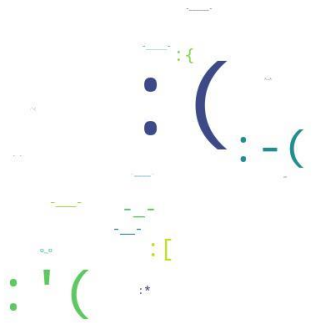
Fear

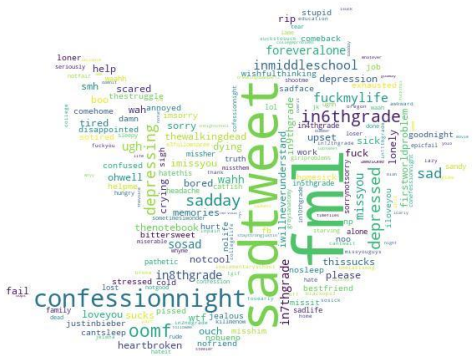
- EMOTICONS
- EMOJI
- HASHTAG
- WORDS



Joy

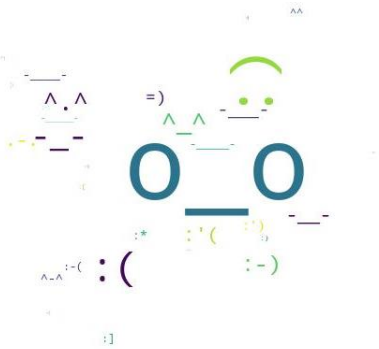
- EMOTICONS
- EMOJI
- HASHTAG
- WORDS

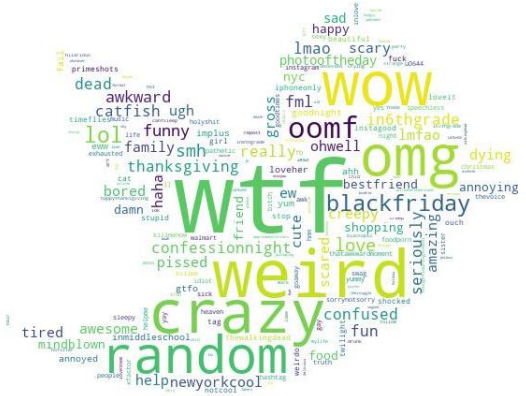




Sadness

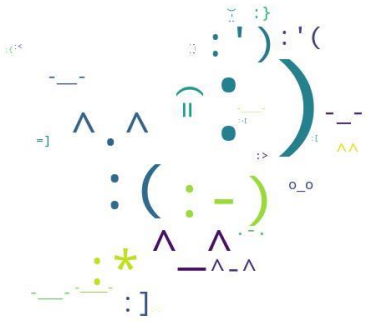
- EMOTICONS
- EMOJI
- HASHTAG
- WORDS

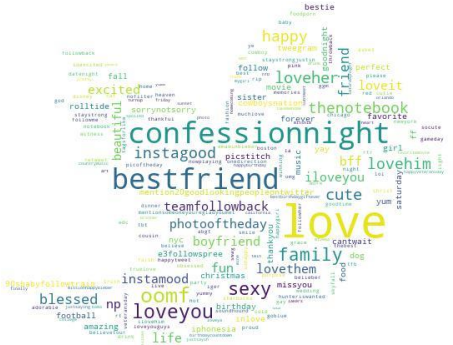




Surprise

- EMOTICONS
- EMOJI
- HASHTAG
- WORDS





Trust

- EMOTICONS
- EMOJI
- HASHTAG
- WORDS

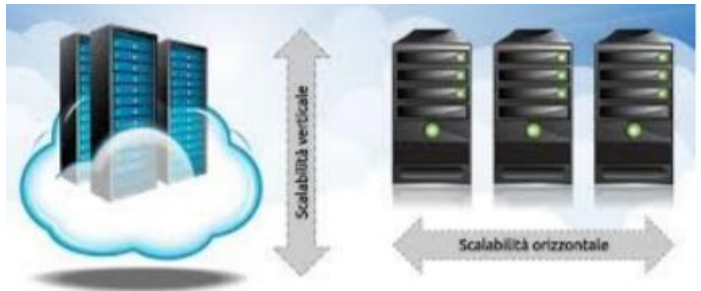
Conclusions

Conclusions (1/2)

PROPERTY	MySQL	MongoDB
Modello dei dati	• Struttura dati fissa; • Adatto per una struttura di dati che non cambierà nel tempo; • Si creano le tabelle richieste, le colonne e si specifica il tipo di dati per ogni colonna; • Lo schema fisso comporta la presenza di valori NULL; • Prima di poter memorizzare i dati, è necessario definire tabelle e colonne.	• Flessibilità e dinamicità dello schema; • Struttura KEY-VALUE; • Possibilità di annidamenti (nested); • I singoli documenti hanno una propria struttura, che può essere diversa dagli altri → in ogni momento è possibile creare nuovi campi con un valore qualsiasi.
Ricerca dei dati	• Utilizzo dell'operatore JOIN che permette di combinare dati da più tabelle; • Le FOREIGN KEY consentono di creare relazioni tra tabelle; • Una ricerca efficace richiede una conoscenza approfondita del modello dei dati di riferimento.	• Grazie all'annidamento (nested) tutti i dati necessari sono presenti in un solo documento. • L'assenza di operazioni di JOIN tra i documenti comporta performance più alte per i tempi di risposta.

Conclusions (2/2)

PROPERTY	MySQL	MongoDB
Vincoli e integrità dei dati	• Non accetta alcun tipo di dato che non rispetti la strutture imposta dal programmatore.	• Non esiste un vincolo sul tipo di dato. • Svantaggio: non essendoci dei controlli sull'integrità dei dati, il compito ricade totalmente sull'applicativo che dialoga col database.
Scalabilità	• Scalabilità verticale; • A fronte di un maggiore carico di lavoro si procede a incrementare l'hardware sulle macchine server; • La replica e il clustering sono disponibili, ma comportano complessità implementativa.	• Scalabilità orizzontale → caratteristica importante per i Big Data; • Consente di distribuire i dati e le operazioni su macchine differenti al fine di parallelizzare le operazioni; • È possibile configurare più nodi che si replicano automaticamente senza un singolo punto di errore, cosi da evitare che il crash di un server porti all'interruzione del servizio; • Elevata scalabilità e disponibilità dei dati.



GRAZIE PER L'ATTENZIONE