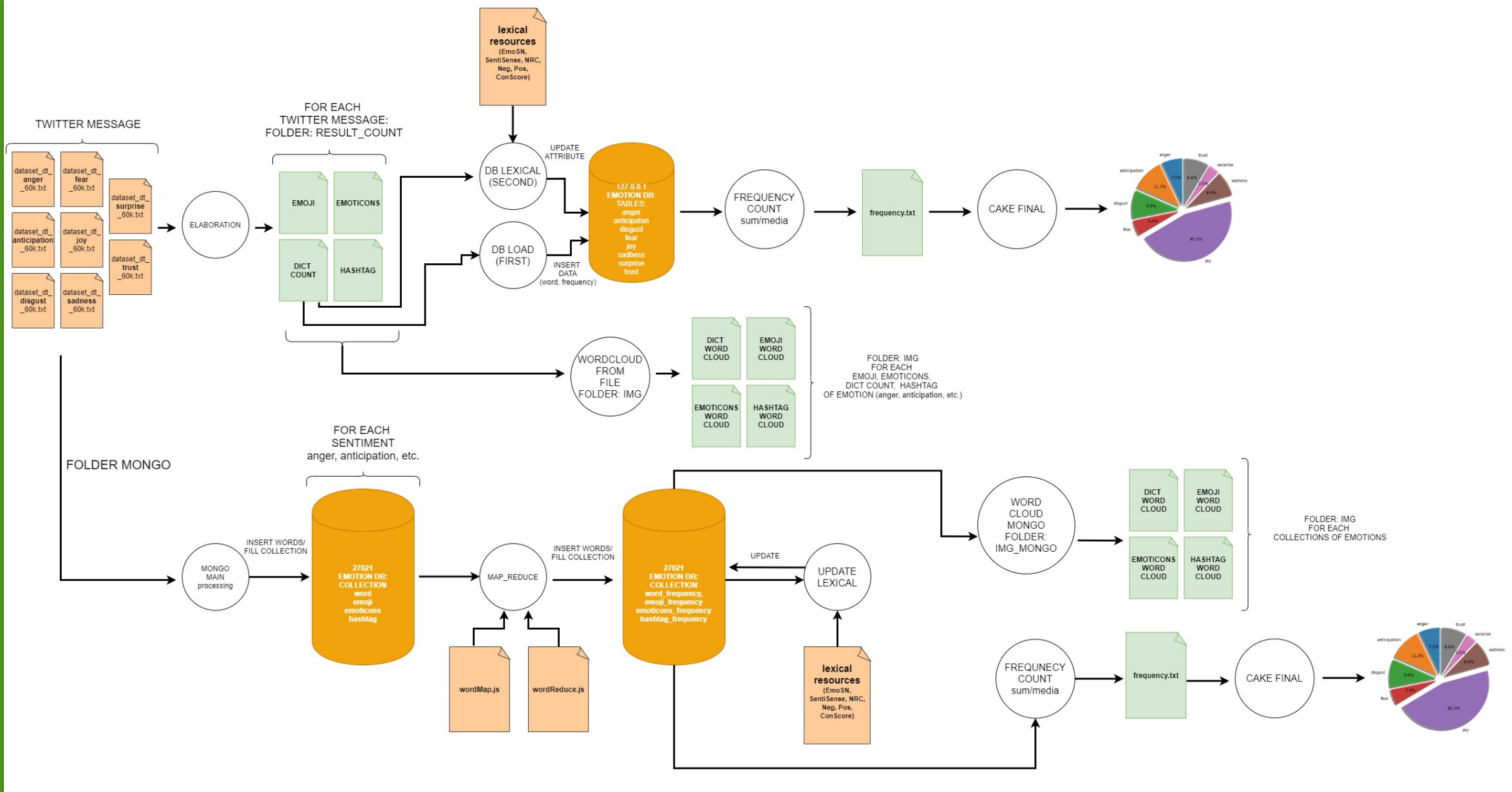


Progetto di Modelli e Architetture Avanzate di DataBase (MAADB)

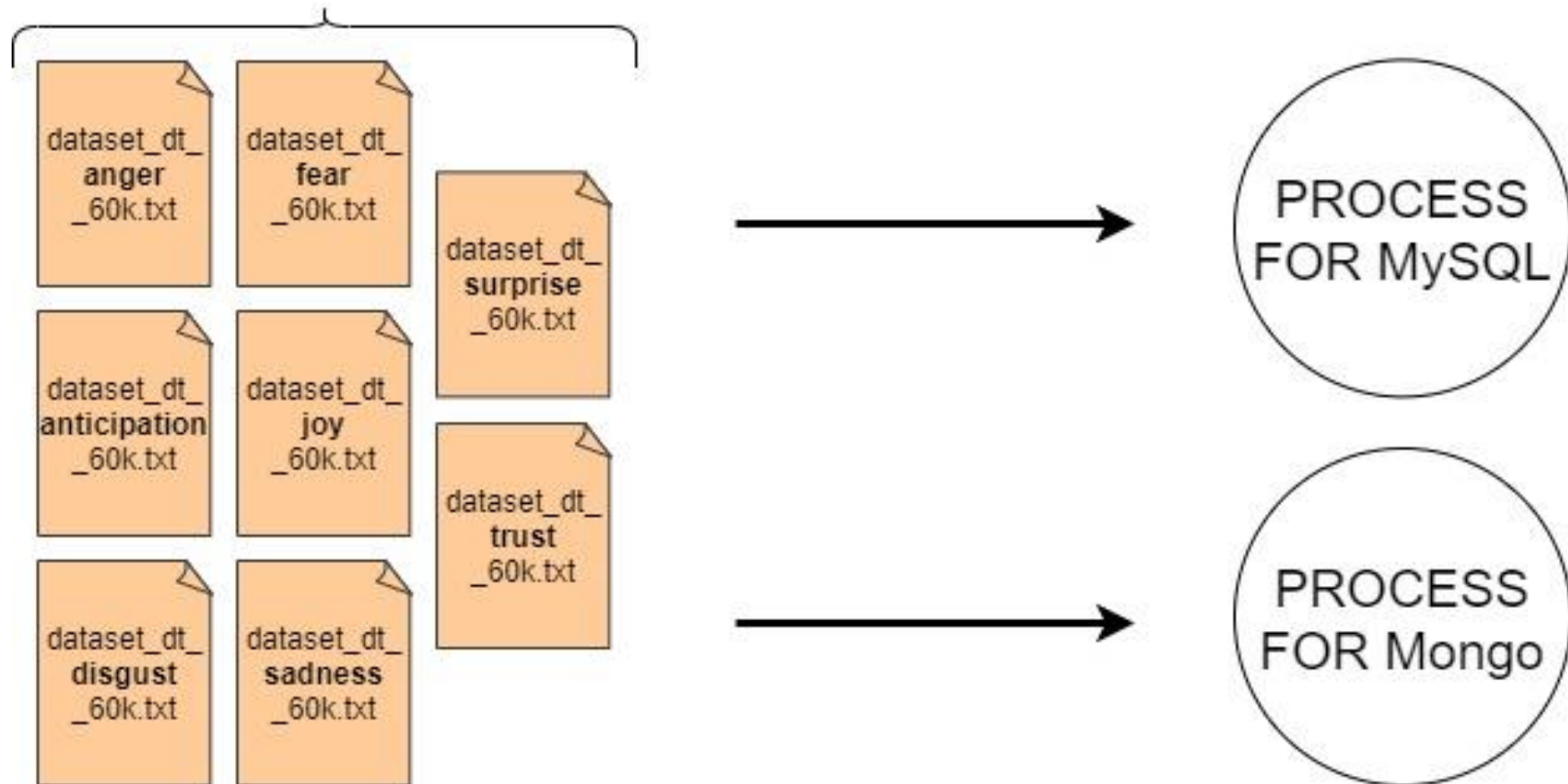
Bortolotti Simone  
De Cenzo Davide  
Marignati Luca

# PROJECT SCHEMA

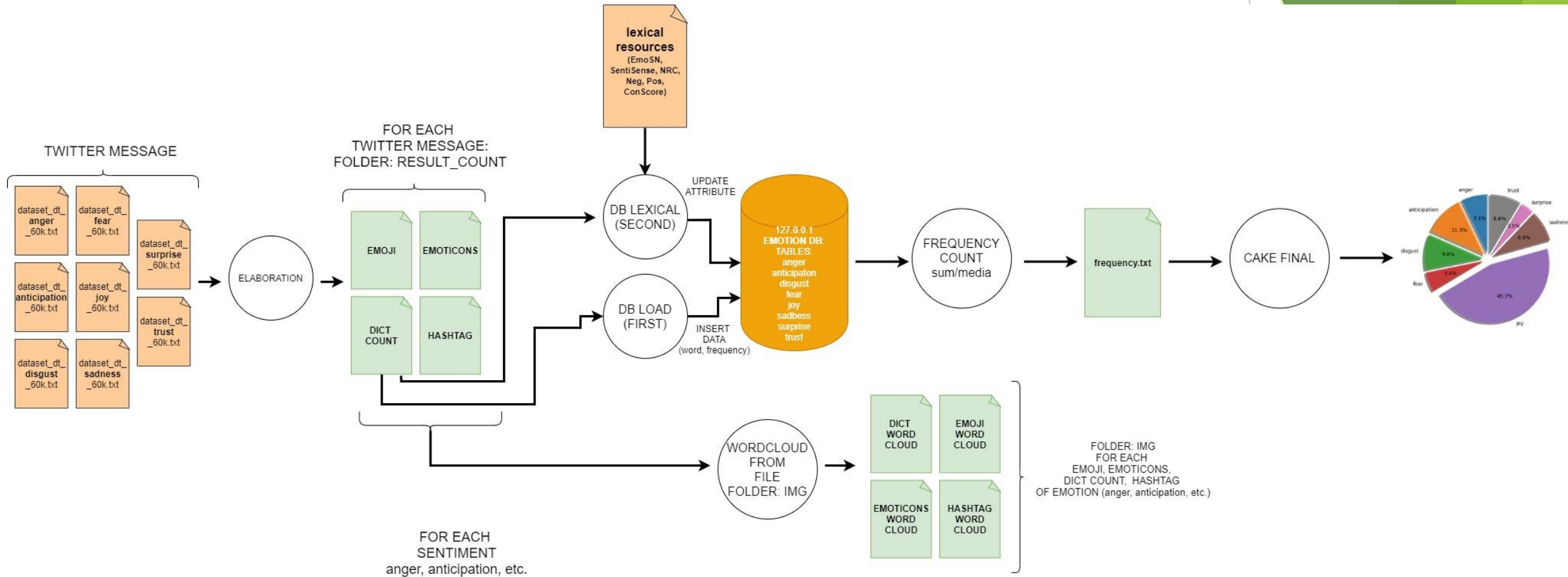




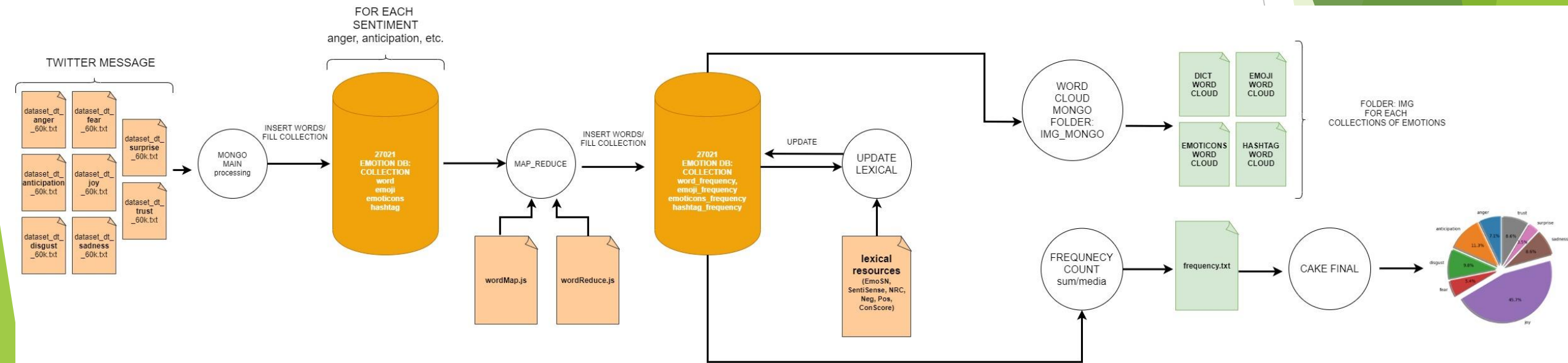
## TWITTER MESSAGE



# 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

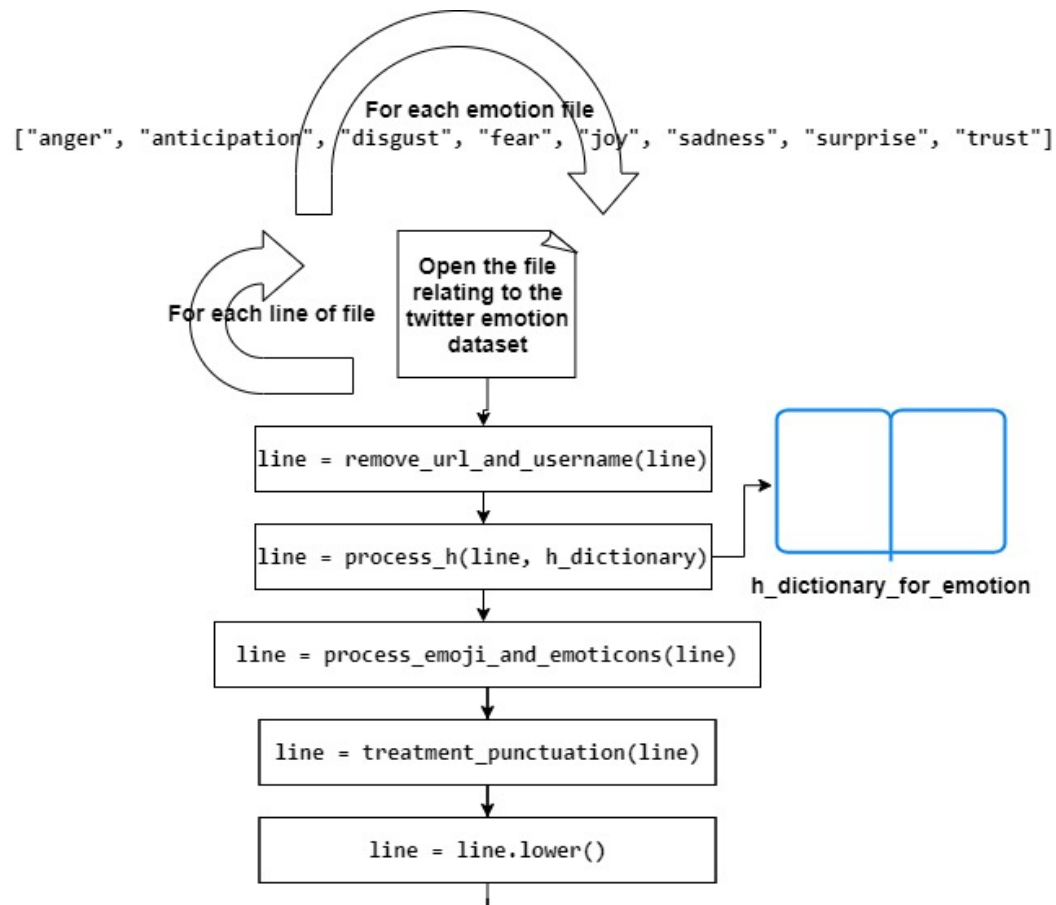
- INPUT: «select sum from DB»
- average calculation
- OUTPUT: save to file «frequency.txt»

## 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

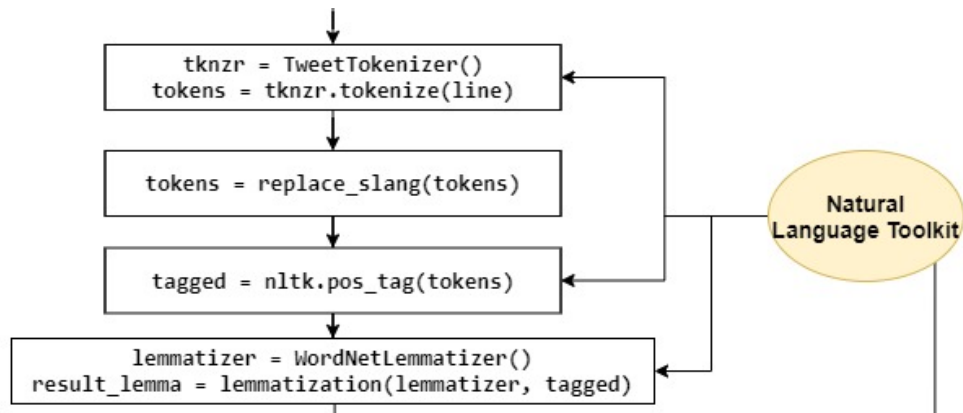
(pretty legit) @ atm the DISNEY gallery

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')]

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

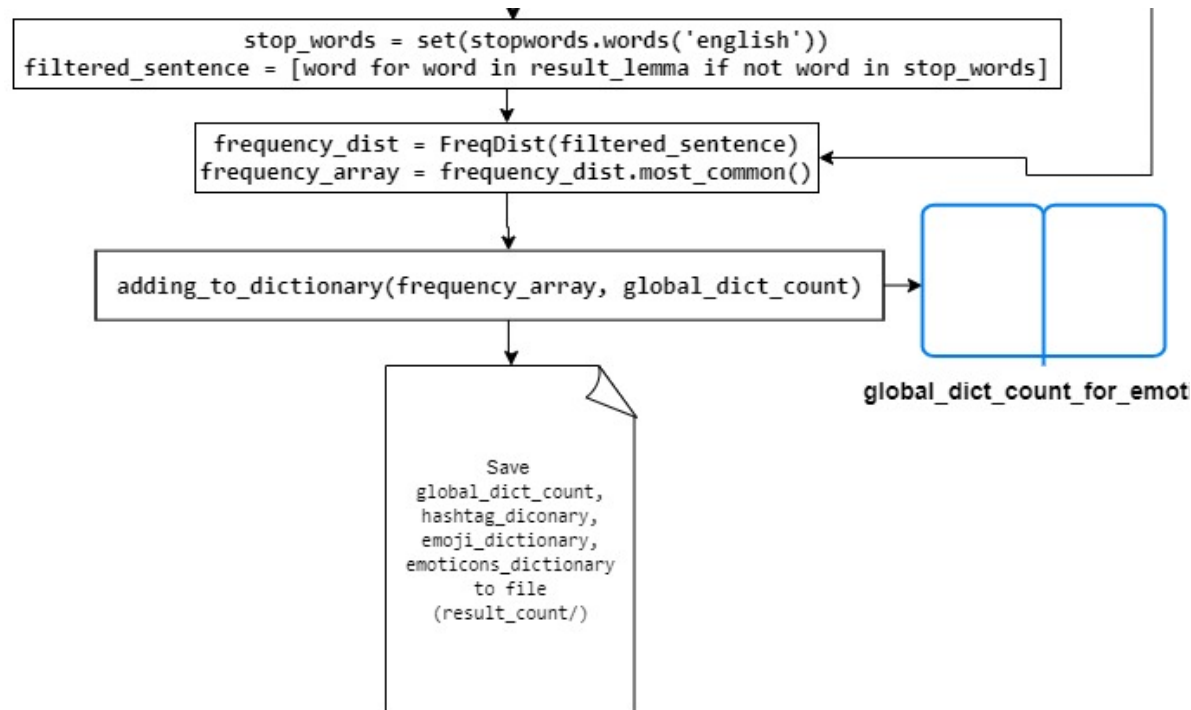
TAG	PART OF SPEECH (POS)
JJ	adjective
NNP	proper noun
VBD	verb, past tense took
DT	determiner
NNS	noun, plural
RB	adverbvery, silently

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

# POS TAGGING

# 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

```
mirror_mod = modifier_ob.  
#set mirror object to mirror  
mirror_mod.mirror_object  
operation == "MIRROR_X":  
mirror_mod.use_x = True  
mirror_mod.use_y = False  
mirror_mod.use_z = False  
operation == "MIRROR_Y":  
mirror_mod.use_x = False  
mirror_mod.use_y = True  
mirror_mod.use_z = False  
operation == "MIRROR_Z":  
mirror_mod.use_x = False  
mirror_mod.use_y = False  
mirror_mod.use_z = True  
#selection at the end -add  
mirror_ob.select= 1  
modifier_ob.select=1  
context.scene.objects.active  
("Selected" + str(modifier_ob.  
mirror_ob.select = 0  
= bpy.context.selected_object  
data.objects[one.name].select  
print("please select exactly  
-- OPERATOR CLASSES -----  
types.Operator):  
on X mirror to the selected  
object.mirror_mirror_x"  
mirror X"  
context):  
context.active_object is not
```

# 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

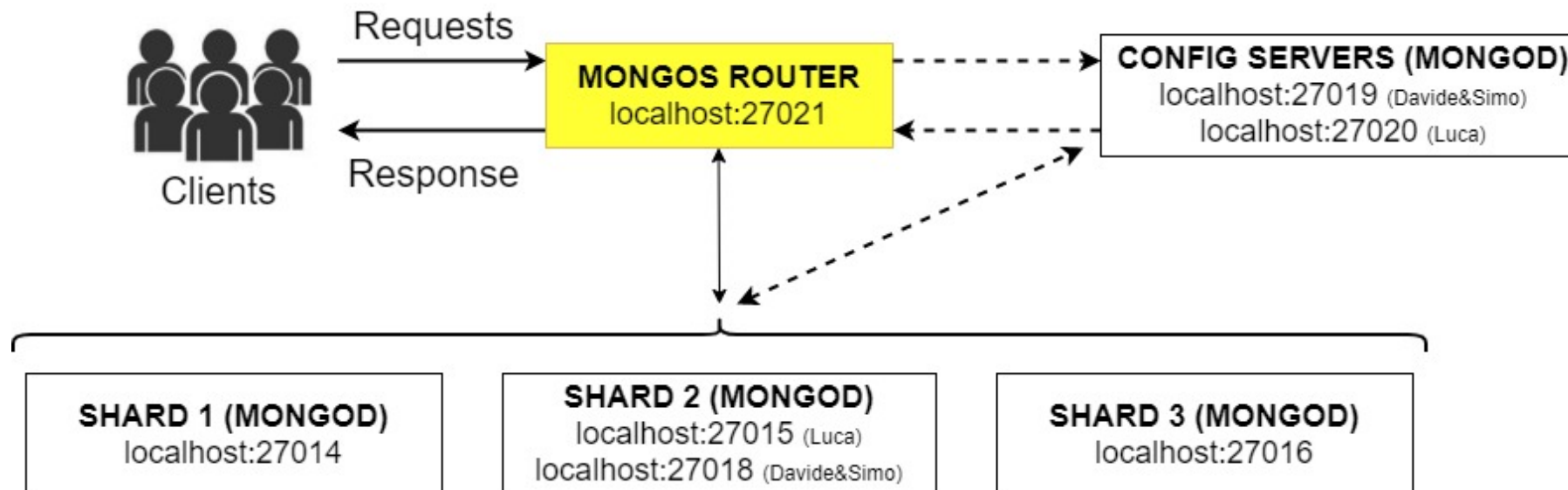
The background features a series of overlapping, semi-transparent green triangles and polygons that create a dynamic, layered effect. The colors range from a light, pale green to a deep, forest green. The shapes are oriented diagonally, with some pointing towards the top right and others towards the bottom left. The overall composition is modern and minimalist.

# MongoDB

# CLUSTER ARCHITECTURE (1/3)

## MONGOS ROUTER

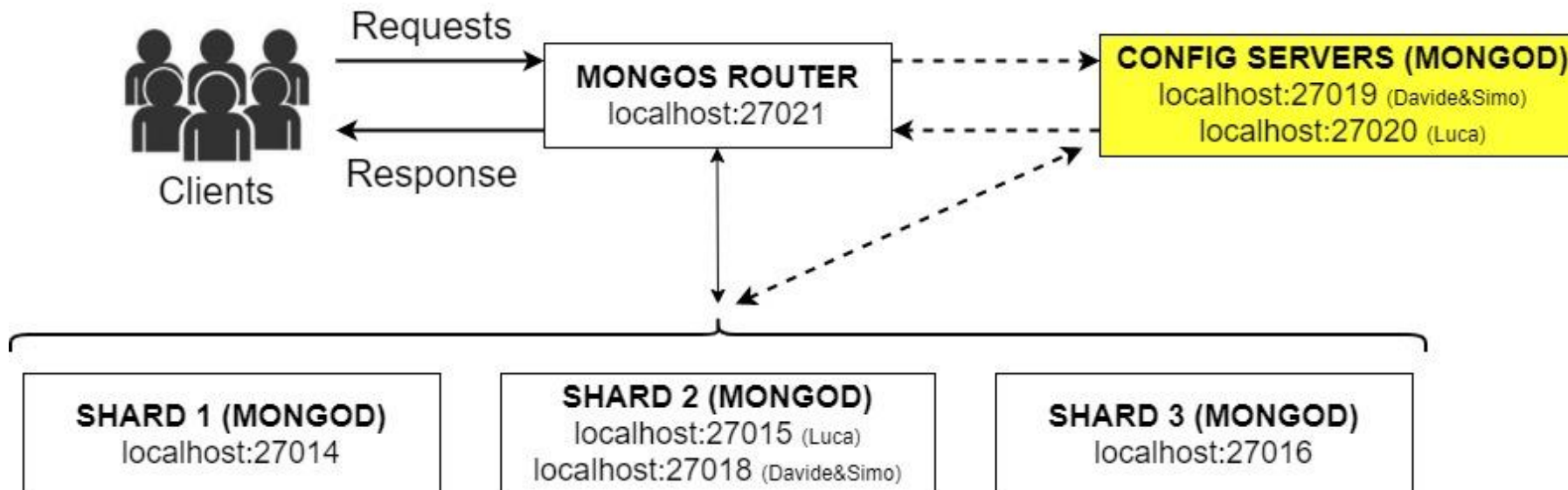
- ▶ Router/balancer;
- ▶ Interfaccia alle applicazioni client;
- ▶ Le applicazioni esterne non devono preoccuparsi dell'architettura del sistema, ma possono solamente connettersi a questo nodo come se fosse un semplice database MongoDB;
- ▶ Tutte le query e le operazioni di scrittura vengono quindi inviate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

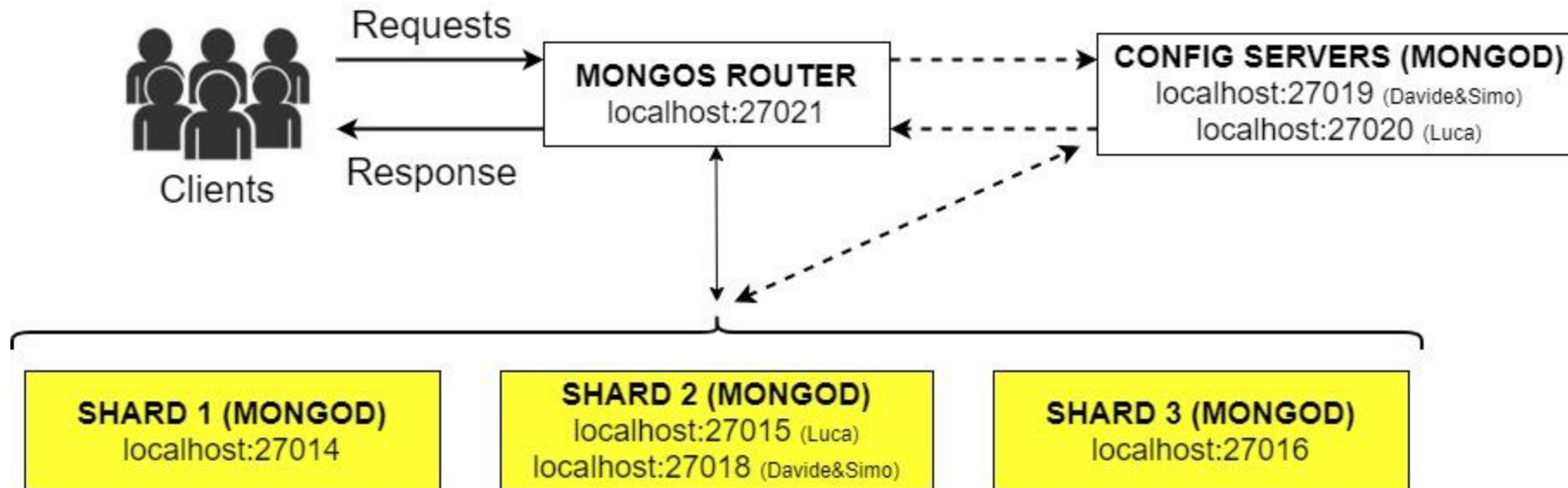
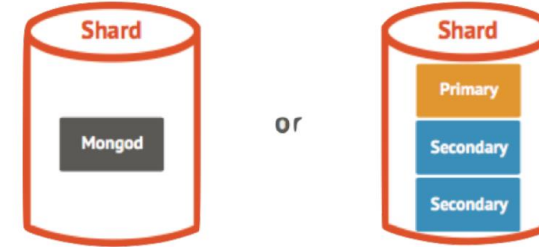




# CLUSTER ARCHITECTURE (3/3)

## 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
```

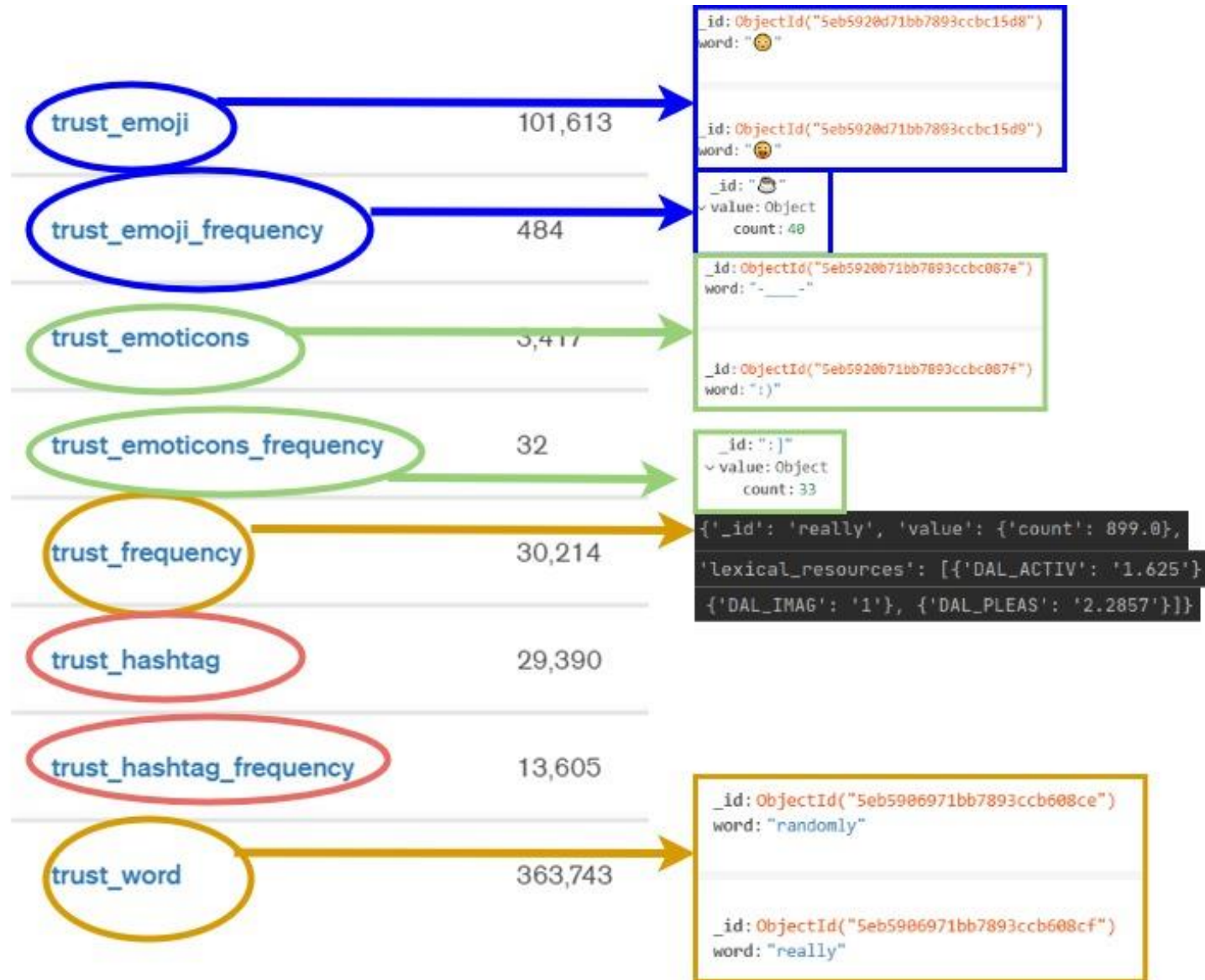
```
Shard shard0000 at localhost:27014  
data : 5.55MiB docs : 152474 chunks : 2  
estimated data per chunk : 2.77MiB  
estimated docs per chunk : 76237
```

```
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)

```
1 import pymongo
2 from bson import Code
3
4 client = pymongo.MongoClient("mongodb://localhost:27021/?readPreference=primary&appname=MongoDB%20Compass%20Community"
5                               "&ssl=false&retryWrites=false&w=majority")
6 db = client['emotion']
7
8 dataset_sentiment = ["anger", "anticipation", "disgust", "fear", "joy", "sadness", "surprise", "trust"]
9
10 for emotion in dataset_sentiment:
11
12     emotion_word = db[emotion + '_word']
13     emotion_emoji = db[emotion + '_emoji']
14     emotion_emoticons = db[emotion + '_emoticons']
15     emotion_hashtag = db[emotion + '_hashtag']
16
17     # Load map and reduce functions
18     map = Code(open('wordMap.js', 'r').read())
19     reduce = Code(open('wordReduce.js', 'r').read())
20
21     results = emotion_word.map_reduce(map, reduce, emotion + "_frequency")
22     emotion_emoji.map_reduce(map, reduce, emotion + "_emoji_frequency")
23     emotion_emoticons.map_reduce(map, reduce, emotion + "_emoticons_frequency")
24     emotion_hashtag.map_reduce(map, reduce, emotion + "_hashtag_frequency")
```



# map\_reduce (2/5)

## wordMap.js

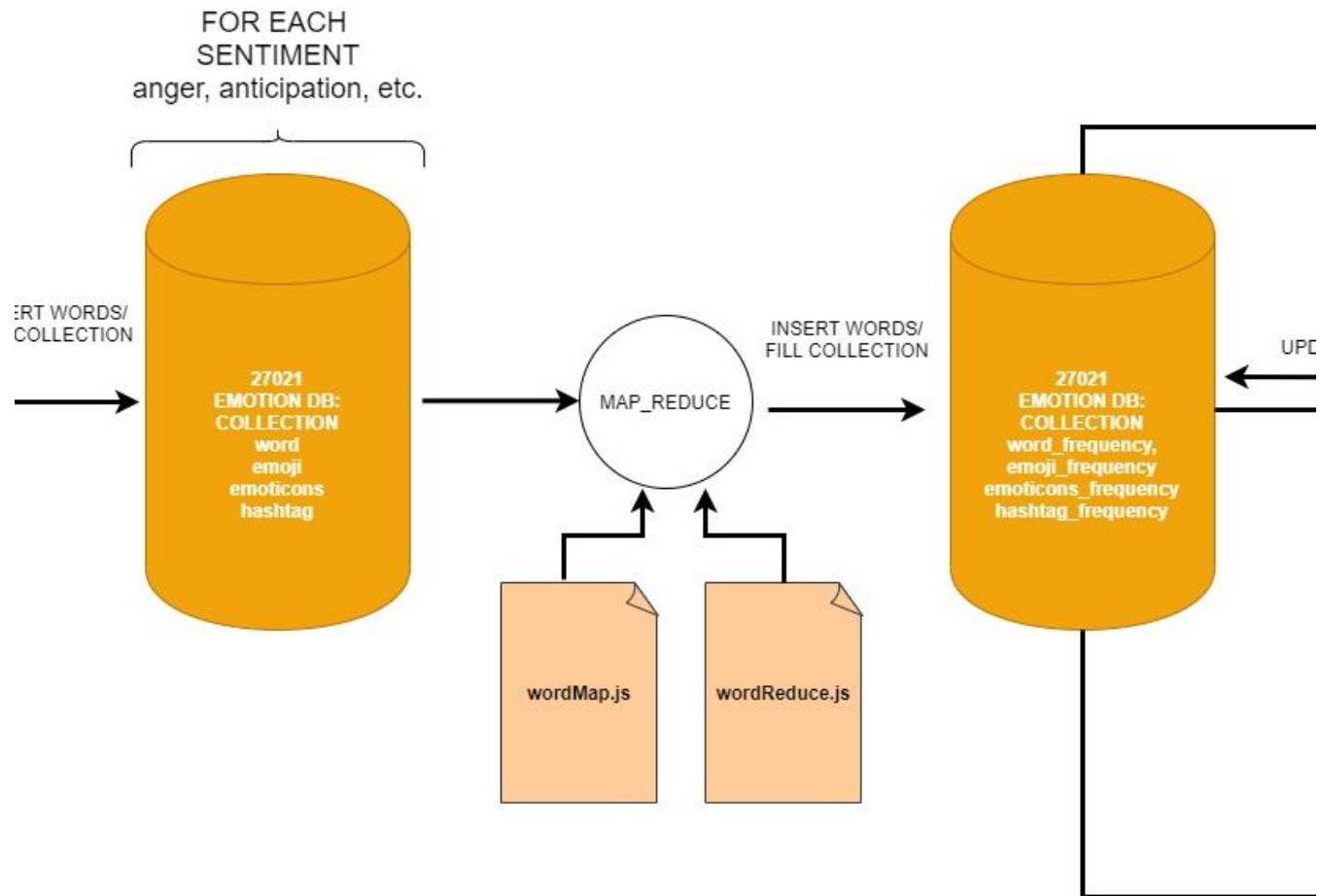
```
1 function wordMap(){
2
3     var words = this.word.match(/^\s*\S+(\s?\S)*\s*$/g);
4
5     if(words == null) {
6         return;
7     }
8
9     for (var i = 0; i < words.length; i++){
10         emit(words[i], {count: 1});
11     }
12 }
```

# map\_reduce (3/5)

## wordReduce.js

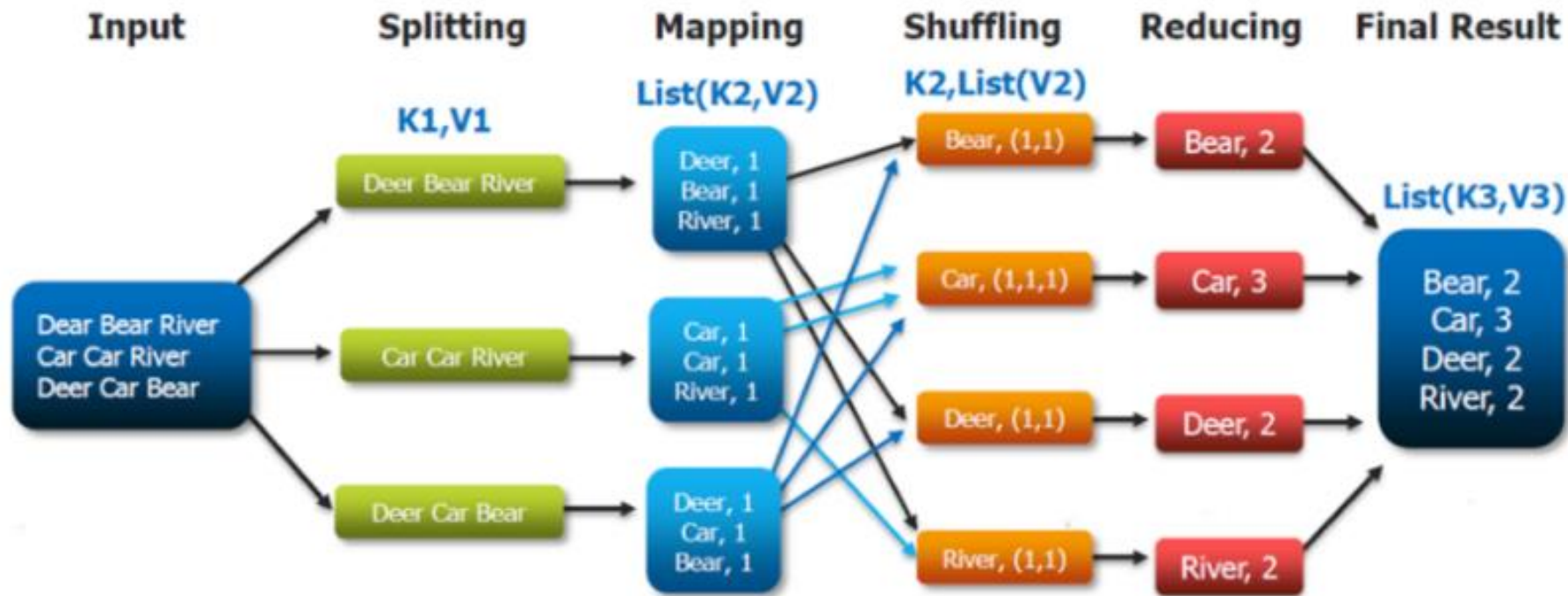
```
1 function wordReduce(key, values){  
2     var total = 0;  
3     for (var i = 0; i < values.length; i++){  
4         total += values[i].count;  
5     }  
6     return {count: total};  
7 }
```

# map\_reduce (4/5)



# map\_reduce (5/5)

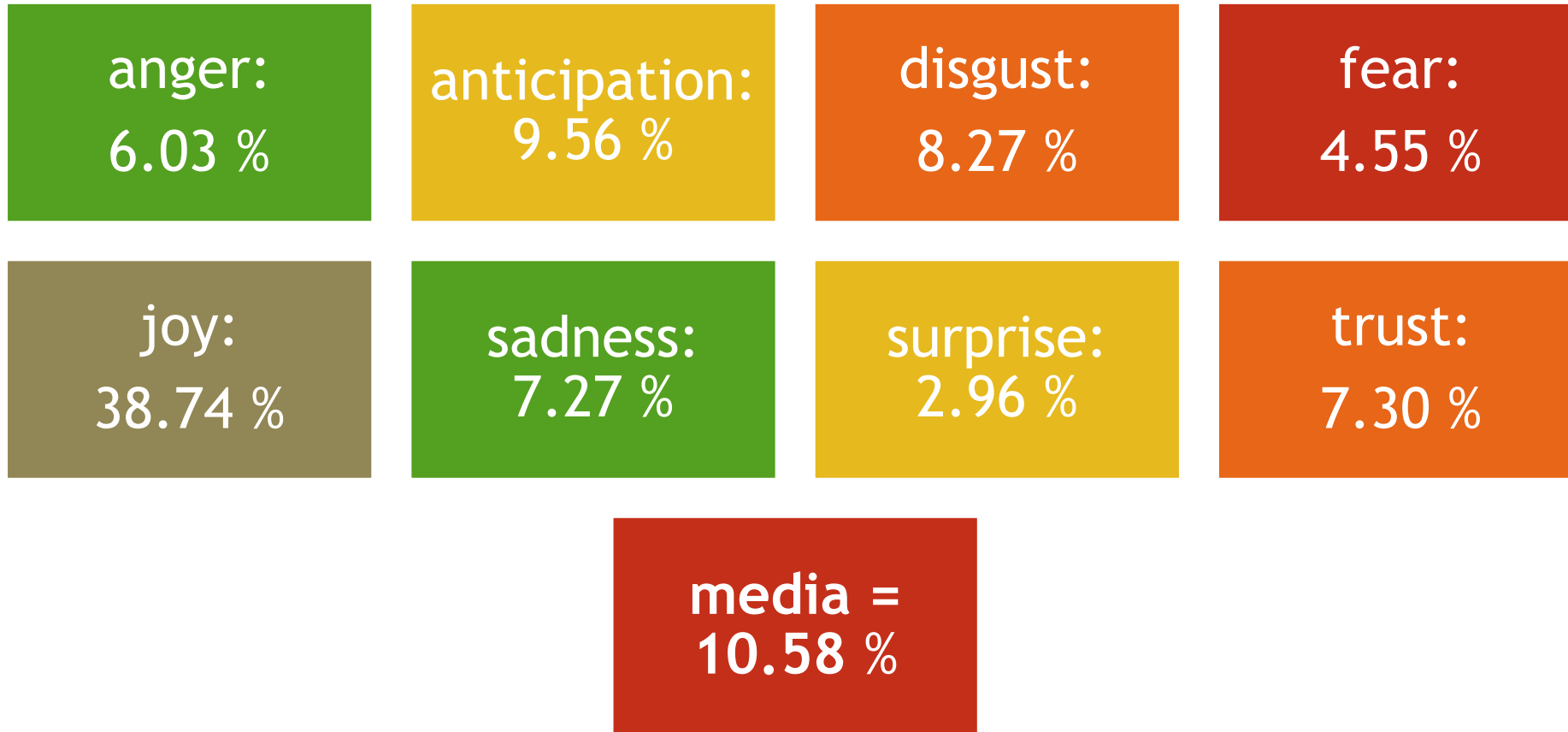
## The Overall MapReduce Word Count Process





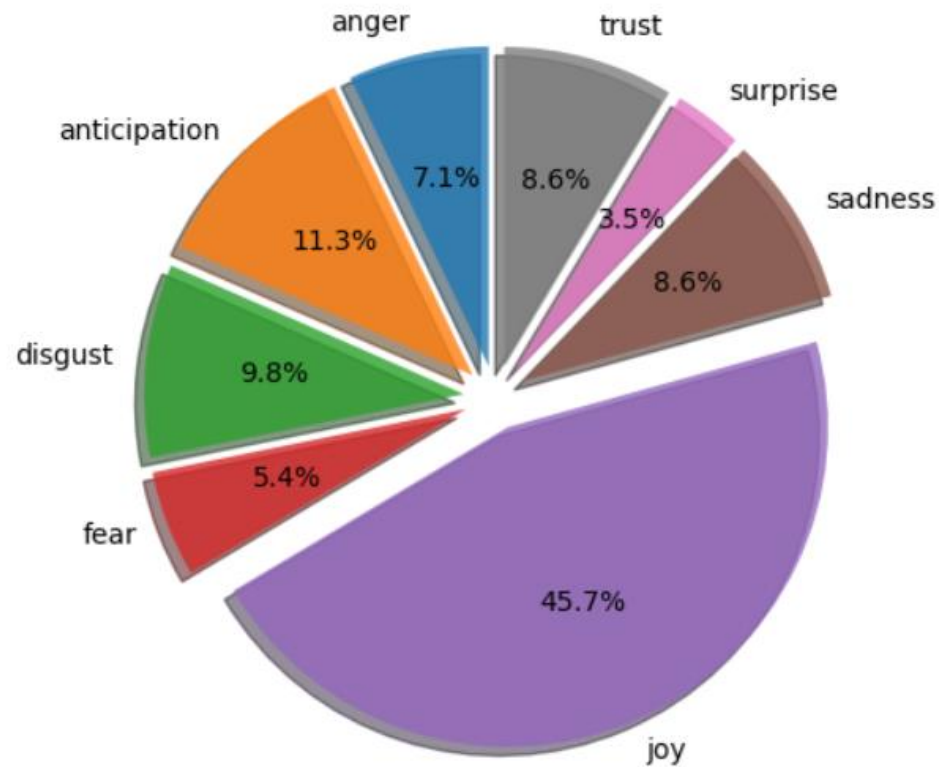
# Results

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

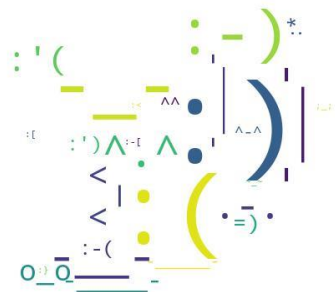




# Statistics of use of lexical resources (MySQL/Mongo) (2/2)



WORDCLOUD



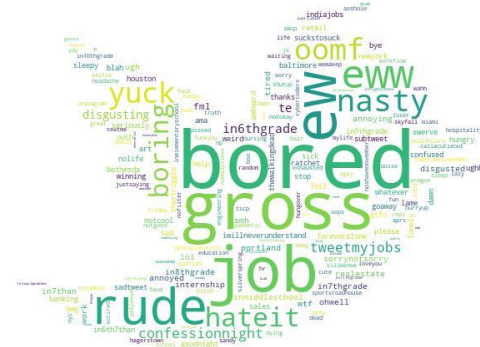
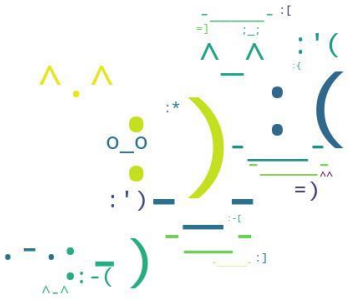
# Anger

- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS



# Anticipation

- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS



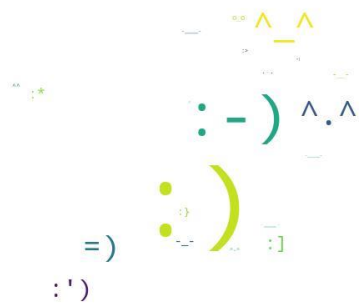
# Disgust

- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS



- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS

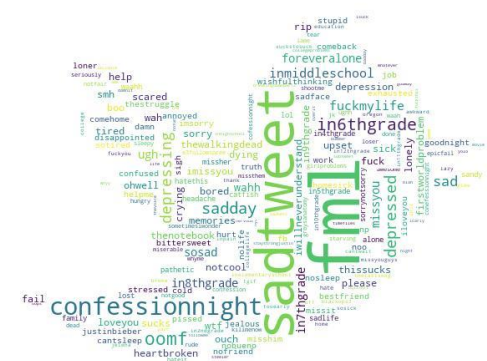
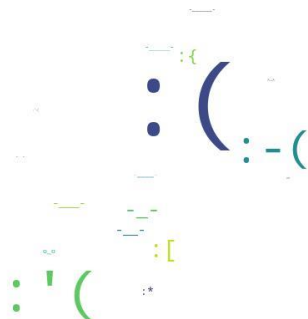
- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS



# Joy

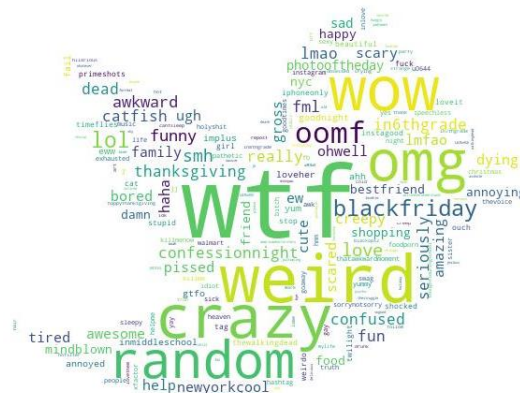
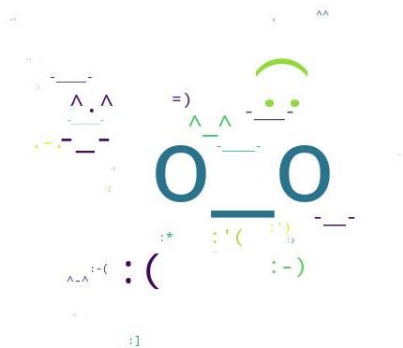
- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS





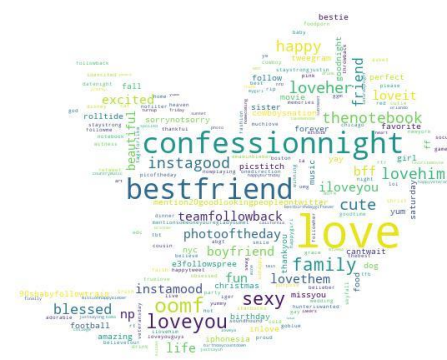
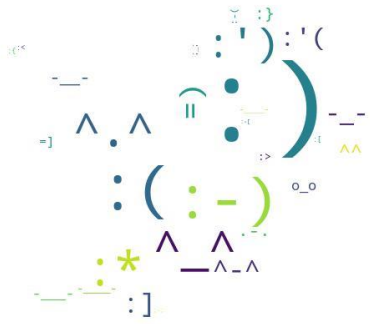
# Sadness

- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS



# Surprise

- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS



# Trust

- ▶ EMOTICONS
- ▶ EMOJI
- ▶ HASHTAG
- ▶ WORDS

# Conclusions

# Conclusions (1/2)

PROPERTY	MySQL	MongoDB
Modello dei dati	<ul style="list-style-type: none"><li>• Struttura dati fissa;</li><li>• Adatto per una struttura di dati che non cambierà nel tempo;</li><li>• Si creano le tabelle richieste, le colonne e si specifica il tipo di dati per ogni colonna;</li><li>• Lo schema fisso comporta la presenza di valori NULL;</li><li>• Prima di poter memorizzare i dati, è necessario definire tabelle e colonne.</li></ul>	<ul style="list-style-type: none"><li>• Flessibilità e dinamicità dello schema;</li><li>• Struttura KEY-VALUE;</li><li>• Possibilità di annidamenti (nested);</li><li>• 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.</li></ul>
Ricerca dei dati	<ul style="list-style-type: none"><li>• Utilizzo dell'operatore JOIN che permette di combinare dati da più tabelle;</li><li>• Le FOREIGN KEY consentono di creare relazioni tra tabelle;</li><li>• Una ricerca efficace richiede una conoscenza approfondita del modello dei dati di riferimento.</li></ul>	<ul style="list-style-type: none"><li>• Grazie all'annidamento (nested) tutti i dati necessari sono presenti in un solo documento.</li><li>• L'assenza di operazioni di JOIN tra i documenti comporta performance più alte per i tempi di risposta.</li></ul>

# Conclusions (2/2)

PROPERTY	MySQL	MongoDB
Vincoli e integrità dei dati	<ul style="list-style-type: none"><li>Non accetta alcun tipo di dato che non rispetti la struttura imposta dal programmatore.</li></ul>	<ul style="list-style-type: none"><li>Non esiste un vincolo sul tipo di dato.</li><li>Svantaggio: non essendoci dei controlli sull'integrità dei dati, il compito ricade totalmente sull'applicativo che dialoga col database.</li></ul>
Scalabilità	<ul style="list-style-type: none"><li>Scalabilità verticale;</li><li>A fronte di un maggiore carico di lavoro si procede a incrementare l'hardware sulle macchine server;</li><li>La replica e il clustering sono disponibili, ma comportano complessità implementativa.</li></ul>	<ul style="list-style-type: none"><li>Scalabilità orizzontale → caratteristica importante per i Big Data;</li><li>Consente di distribuire i dati e le operazioni su macchine differenti al fine di parallelizzare le operazioni;</li><li>È possibile configurare più nodi che si replicano automaticamente senza un singolo punto di errore, così da evitare che il crash di un server porti all'interruzione del servizio;</li><li>Elevata scalabilità e disponibilità dei dati.</li></ul>



GRAZIE PER  
L'ATTENZIONE

The background features a series of overlapping, semi-transparent green triangles and polygons that create a dynamic, abstract composition. The colors range from a light, pale green to a deep, forest green. The geometric shapes are primarily located on the right side of the image, with some extending towards the center, leaving the left side mostly white.