

# ***Il database diventa smart con Alexa. Come abilitare i conversational analytics con un time series database open source***

Marco Riva - Software Engineer - Radicalbit



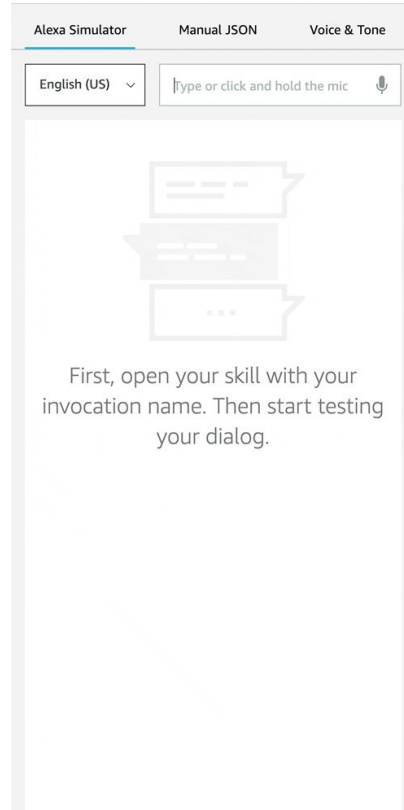
***Online Tech Conference***

***- Italian edition -***

**9-10-11 Novembre, 2021**



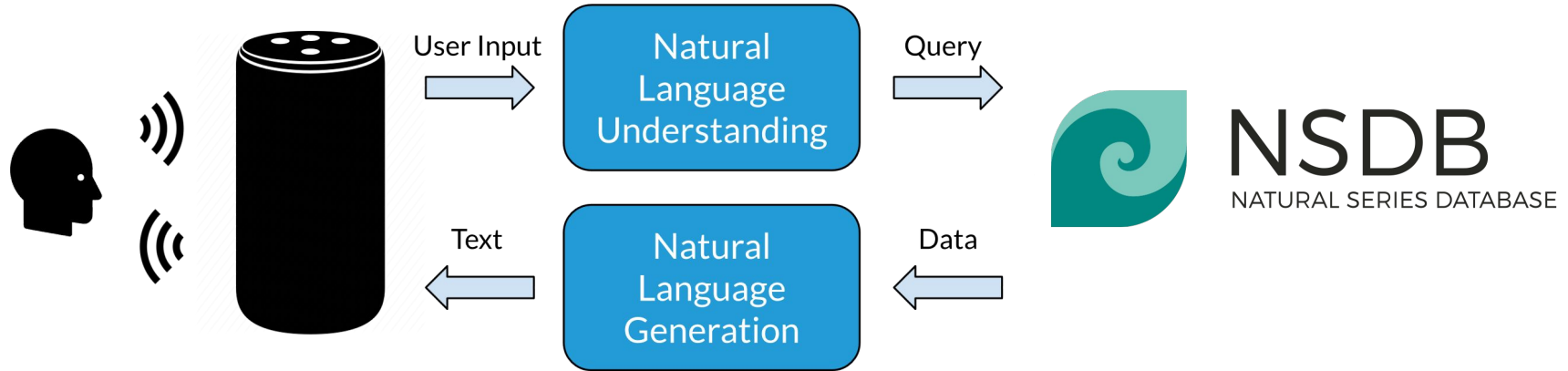
# Alexa, what we will talk about today?



# Alexa, say hi to NSDb!

End-to-end system that is capable of:

- understanding human language through Alexa to query a database,
- provide a coherent response in natural language based on the gathered data.

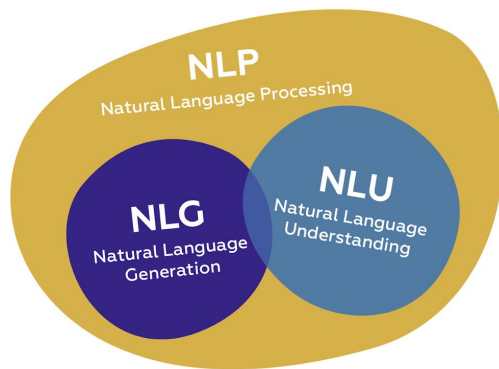


# When a machine meets an human

## NLP Natural Language Processing

**NLU** Natural Language  
Understanding → extract the  
meaning from a sentence in  
human language

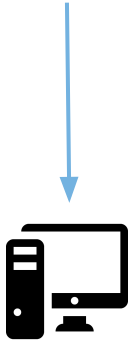
**NLG** Natural Language  
Generation → generate human  
language starting from data



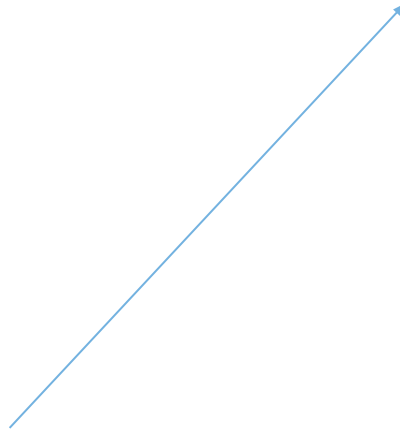
# Yes, but, concretely?



*“Find me the number of tweets since yesterday”*



*select count(value) from tweets  
where timestamp < now - 24h*



count(value)

1250



*“The number of tweets is 1250 since yesterday”*

# The core components



**NSDB**  
NATURAL SERIES DATABASE



amazon alexa



AWS Lambda



RosaeNLG

# Time is precious

Optimized for time-series, SQL-like language

Time-series record → Bit

- timestamp: Long
- value: Numeric
- dimensions: Set((key: String → value: T))

Bit → Tweet

- timestamp → the timestamp of the tweet
- value → the number of retweet
- dimensions → hashtag, username, likes, ecc -...



**NSDB**  
NATURAL SERIES DATABASE

NSDb	Traditional RDBMS PostgreSQL
metric (set of Bits)	table
namespace (set of metrics)	schema
database (set of namespaces)	database

# Alexa, our interface



**Alexa Skills:** apps for Alexa, they are a way to customize the behavior of the device to fulfill user's custom requests.

We have to implement these two parts





# Lambdas come in handy



We can run the code that will be used by Alexa (the Skill Logic) through AWS Lambdas, for an easy and fast implementation.

The behavior of a Lambda is simple, we call the function, there is an arbitrary complex processing phase, and we have the result back.





RosaeNLG.org

Open-source, template-based,  
Natural Language Generation  
JavaScript library for node.js or  
client side execution with  
advanced features for  
linguistic resources.

```
-
  var numberOfPears = 2
  if (numberOfPears > 1)
    setRefNumber(numberOfPears, 'P')
  else
    setRefNumber(numberOfPears, 'S')

mixin example
| I have eaten #[+value(numberOfPears, {'TEXTUAL':true })]
| #[+value("pear", {number:numberOfPears})]

| #[+example]
// I have eaten two pears.
-
  numberOfPears = 1

| #[+example]
// I have eaten one pear.
```

```
- var data = ['apples', 'bananas', 'apricots', 'pears'];
  eachz fruit in data with { separator: ',', last_separator: 'and', begin_with_general: 'I love', end:'!' }
  | #{fruit}
  // I love apples, bananas, apricots and pears!
```

# Why RosaeNLG?

- Open source
- Easy to use
- No beforehand data is needed
- Reliability
- Syntactic variations



RosaeNLG.org

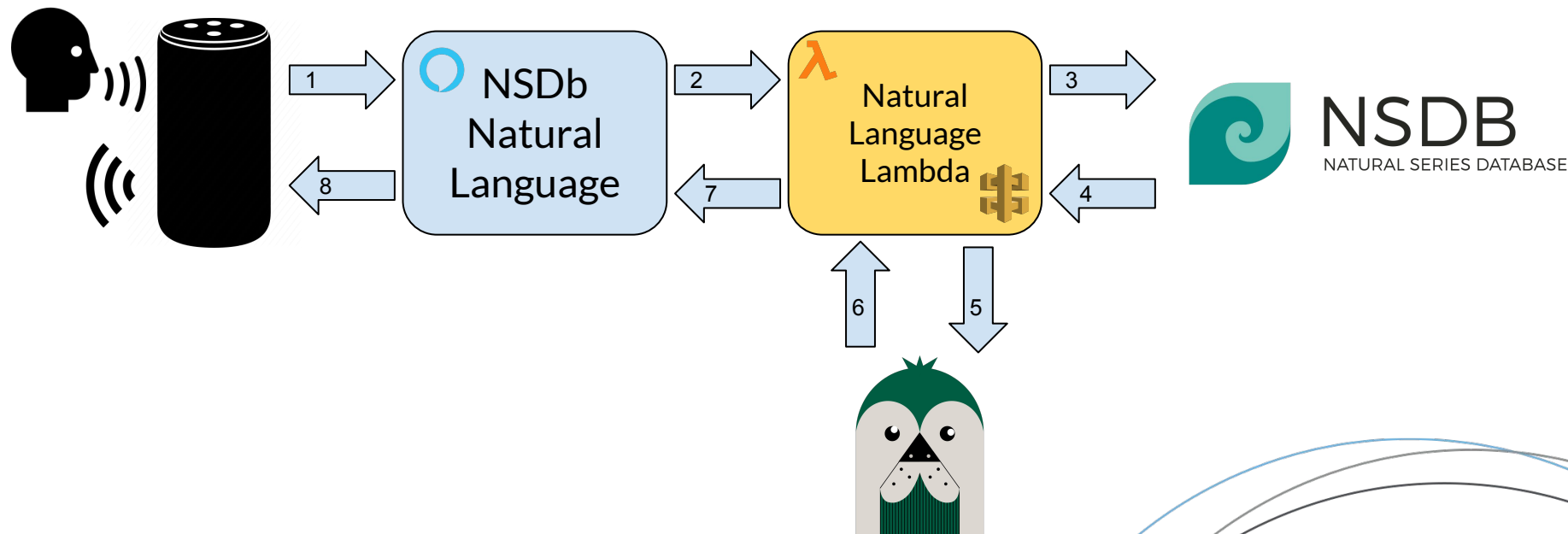
## Alternatives?

Neural Networks (especially **RNN** and **Seq2Seq**) are cool for language generation but they are not reliable and require a lot of data that **we don't have** to train the network.

# Neural Networks hallucinations and issues

Data	name[Cotto], eatType[italian restaurant], near[The Bakers]
1	Cotto is an italian restaurant located near The Bakers
2	Cotto is an italian restaurant and is located near The Bakers
3	Cotto is an italian restaurant with a <u>low price range</u> . It is located near The Bakers
4	Cotto is a <u>place</u> near The Bakers
5	Cotto is a <u>pub</u> near The Bakers

# Put everything together



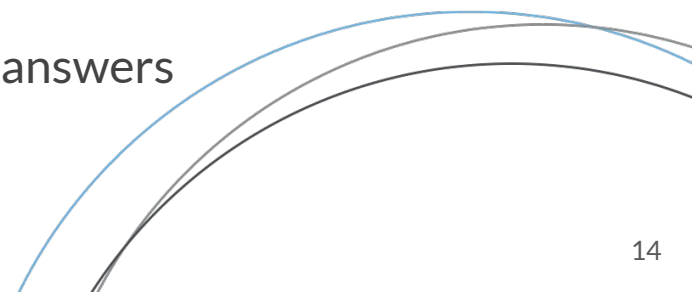
# Towards modularity

Thanks to the modular architecture, we can potentially:

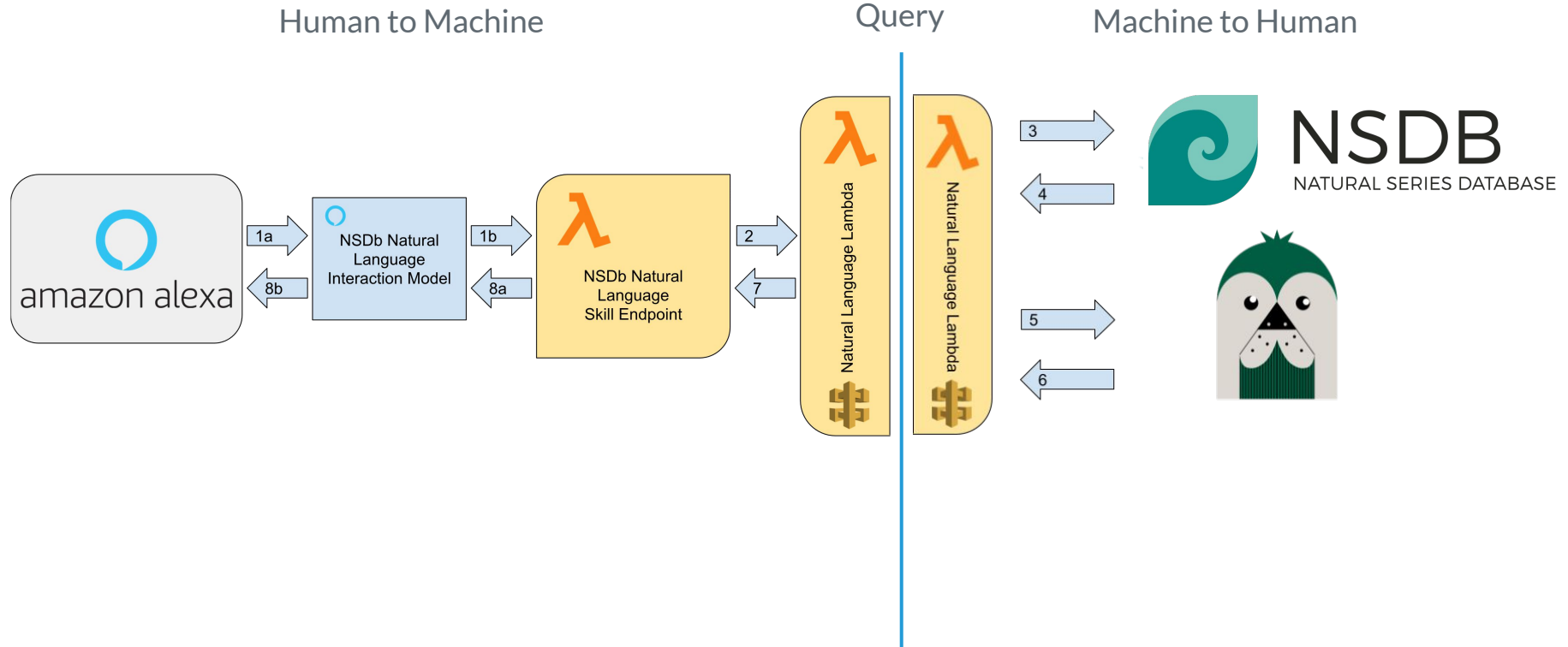
- Change the NLU (Google Home, Siri, chatbot, ...)
- Change the NLG (different tool, Neural Networks, ...)
- Change database (MySQL, PostgreSQL, ...)
- Change language (Italian, French, German, ...)

We can also use the two parts “separately”:

- Natural language queries with raw data
- Standard queries with natural language answers

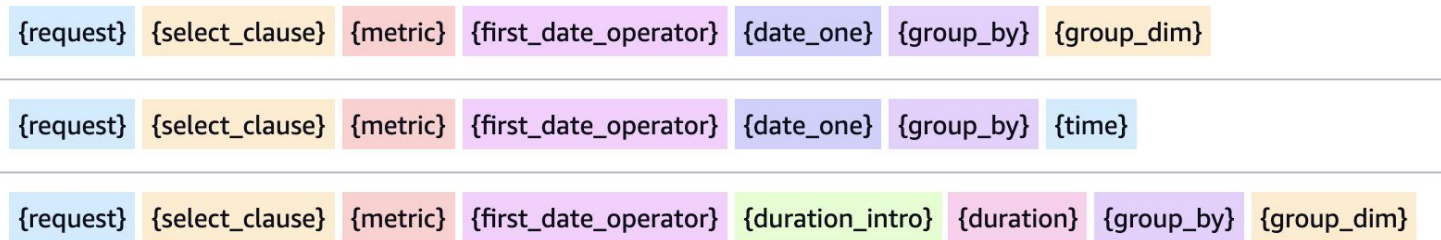


# Let's split the architecture



# Make Alexa understand better

1. **Intents:** An action that fulfills a user's spoken request.
2. **Slots:** Used to catch values coming from the spoken request.
3. **Sample utterances:** A set of likely spoken phrases (also with slots) mapped to the intents.



Triggered Intent



# Alexa, put all the pieces together

{request} {select\_clause} {metric} {first\_date\_operator} {duration\_intro} {duration}

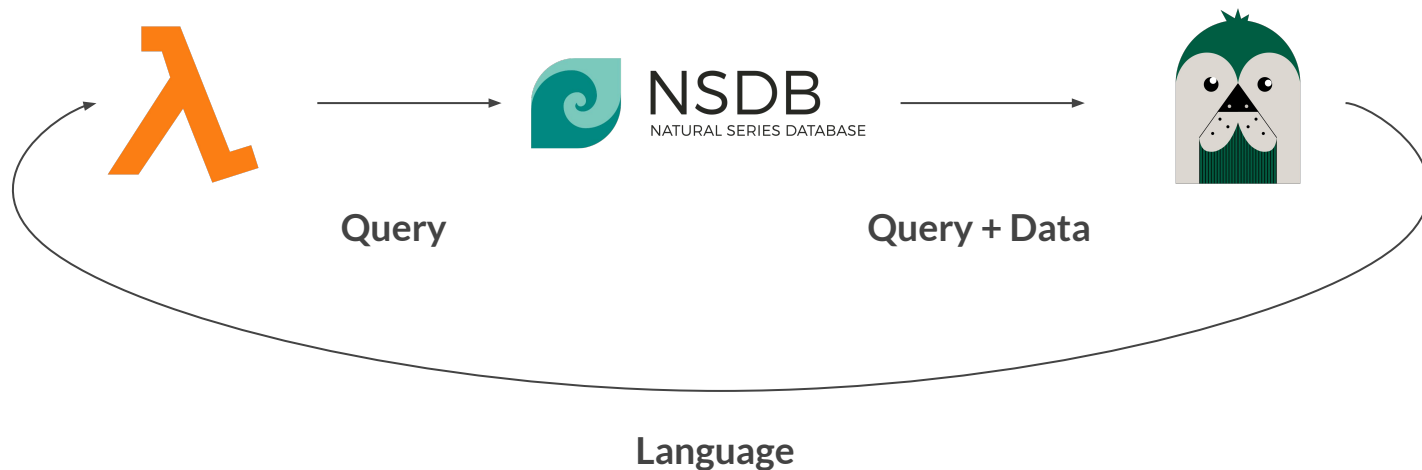


find me the number of tweets in the last twenty-four hours



```
SELECT COUNT(*) FROM tweets WHERE timestamp < now() - 24h
```

# Let the Lambda do the hard work



The communication interface between the NLU (Alexa) and the NLG (RosaeNLG + NSDb) part.

It's a mediator between Alexa, NSDb and RosaeNLG.

# It's time to generate the language

Different templates to handle different types of queries:

- *metadata queries*
  - used to know which databases, namespaces, metrics, dimensions we have inside the instance of NSDb we are referring to
- *plain queries*
  - standard one shot SQL queries, they can be plain, temporal or grouped
- *descriptive queries*
  - aggregation of some queries that are used to build a phrase that describe the *global view* of the metric (for example the minimum and maximum value, ...)

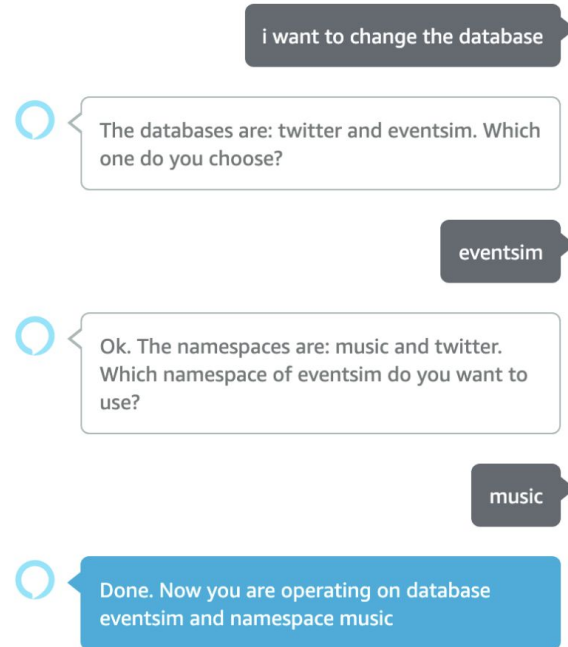
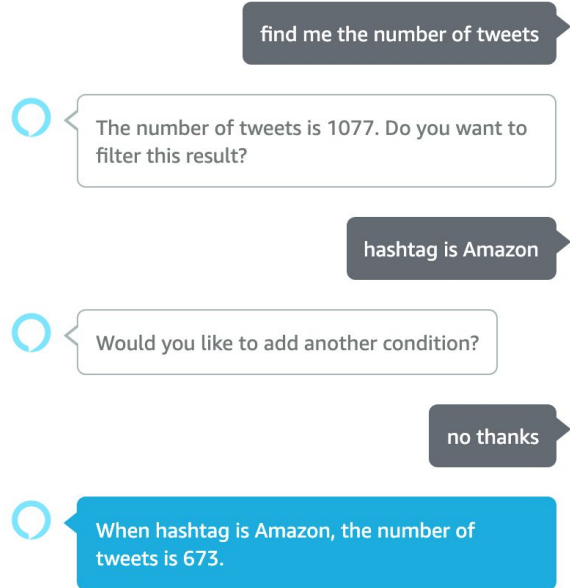
# The template way to generate language

**synz** and **syn** are used to handle synonyms to avoid repetitions.

```

mixin order_by
  if nsdb.parsed.order.direction == 'desc'
    if nsdb.parsed.order.order_by == 'value'
      synz
      syn
      | ranking by #{nsdb.parsed.order.order_by}
      syn
      | from the greatest to the lowest #{nsdb.parsed.order.order_by}
    else
      synz
      syn
      | ordering #{nsdb.parsed.order.order_by} descending
      syn
      | descending order according to #{nsdb.parsed.order.order_by}
      syn
      | ordering the #{nsdb.parsed.order.order_by} in a descending way
      syn
      if nsdb.parsed.order.order_by == 'value'
        | ranking by #{nsdb.parsed.order.order_by}
      syn
      if nsdb.parsed.order.order_by == 'timestamp'
        | from the #[+syn('latest', 'oldest')] to the most recent
    
```

# Let's talk with Alexa



`select count(value) from tweets where hashtag = "Amazon"`

Database selection and namespace description

# Other conversations

find me the plan of listened songs



I've found 2 different values of plan, plan Paid occurred 12971 times, that is the 70 percent, while plan Free occurred 30 percent, with 5414 listened Songs, and in the last 24 hour instead, plan Paid was at 68 percent, with 3652 listened Songs and Free at 32 percent, with 1700.

for each gender find me the number of listened songs



From the greatest to the lowest value, for each different gender, we obtain 2 groups: the number of listened Songs is 10,222 for gender Male and 8,163 for Female. Do you want to filter this result?

plan is paid



Would you like to add another condition?

no thanks



When plan is Paid from the greatest to the lowest value, there are 2 genders: the number of listened Songs is 7,299 when gender is Male and 5,672 for Female.

Descriptive query with useful informations

`select count(value) from listenedSongs`

`where plan = "paid" group by gender`



<thanks/>

marco.riva@radicalbit.io