|  |  |
| --- | --- |
| Document – Store | MongoDB |
| Key – value | Reddis |

Scalability ?

# Dataframe vs RDD (spark):

1. ניתן להתייחס לdataframe כאל רמה מופשטת יותר, בה אומרים לSpark מה ברצוננו לבצע, אבל לא אומרים לה איך בדיוק לבצע את זה (בניגוד לRDD).
2. ניתן להשמש בRDD, גם כאשר הנתונים אינם מסודרים בשום מבנה, בניגוד לdataframes שמצריך מבנה כלשהו (לרוב מבנה רלציוני).
3. Spark מתרגמת את מבנה הdataframe (מאחרי הקלעים) ומבצעת את הפעולות על RDD.
4. תמונה שמכילה שולחן

   התיאור נוצר באופן אוטומטי

# Normalization:

|  |  |
| --- | --- |
| Primary Keys | Unique Keys |
| Cannot be null | Can be null |
| Only one primary key | Multiple unique keys |
| Primary key creates clustered index by default | Unique creates a non-clustered index by default |
| Both can’t appear in two different lines | |

**Normalization:**

**1NF:**

1. each column should have a unique name
2. all data type in column should be the same
3. every column should hold atomic values
4. the order of the data doesn’t matter

**2NF:**

1. 1NF
2. No partial dependencies

**3NF:**

1. 2NF
2. Should not have transitive dependency

**3.5NF:**

1. 3NF

2. For any dependency A->B, A should be a super key,

Or in other words: A cannot be a non-prime attribute if B is a prime attribute.

כל קבוצה שגוררת קבוצה זרה אחרת היא super-key

**4NF:**

1. 3.5NF
2. Should not have multi-valued dependency

Student\_id | course | hobby

Hobby and course have no relation between them and can cause multiple rows of the same information

Primary key is a column or a group of columns.

We need the primary key to fetch any data from the table.

Example 1: ID will be a primary key.

A primary key can identify entire row.

Example 2:

To fetch the mark of a student, I need student\_id and subject\_id.

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

Teach partially depends on primary key, only depends on subject\_id and not student\_id

transitive dependency

An attribute in a table depends on a different non-prime attribute which is not a part of a primary key.

# JSON:

{ “BooksInLibrary”: [{ “bar\_code”: “123456”,

“copy\_Num” : “10”,

“name” : “ben”} , {…}, {…} ] }

# PySpark:

text\_file = sc.textfile(“myfile.txt”)

word\_count = text\_file.flatMap(lambda line: line.split(“ “)\

.map(lambda word: (word,1))\

.reduceByKey(lambda a,b: a+b)\

.collect()

Function to remember:

* The function **zip()** receives two or more lists and combines them together

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

* flatMap( *enter a function with one variable :* toLowerCase)
* GroupBy(lambda w: w[0:3])
* Filter(*function that returns true or false:* lambda x: x not in wrong\_words)
* Reduce(lambda x,y: x+y)
* reduceByKey(lambda a,b: a+b)
* groupBy(‘attribute’)
* sortByKey(true | false)
* str.upper\lower

**Need to remember:**

def BiGram(line):

words = line.split() //returns a list of word

return zip(words, word[1:])

def TrippleGram(line):

word = line.split()

return zip(word, word[1:], word[2:])

def NGram(list, N):

return zip(\*list[i:] for I in range(N)])

# MongoDB

Return specific values:

Find() – returns all values

findOne() – the first value

limit(n) – first N results

db.orders.find({

film\_name:'Aladin',

$or:[

{num\_of\_tickets:2},

{amount:50}

]

},

{

cinema:1,

amount:1,

num\_of\_tickets:1,

\_id:0

});

AND: {“attr1” : “name1”, “attr2”:”name2”}

OR: is used like the example above.

To return specific fields:

{“field1” : 1,

“field2”: 2 ,

**\_id:0** } 🡪 id always return in mongoDB so we erase it

# JAVASTREAMS

* mylist.stream().foreach((obj) -> { do something} ); - terminal operation, after this the function is closed
* Stream.of(mylist)
* Map(call\_function\_here) - produces a new stream after applying a function to each element of the original stream.collect(Collectors.toList())
* Stream.of(mylist).map(myfunc).filter(e -> e!=null).collect(Collector.toList())
* findFirst()
* distinct() – removing duplicates
* to perform the same action over and over we can create an int stream:

IntStream.range(0,n)

.mapToDouble(x -> a \* Math.pow(q,x))

.reduce(1, (x,y) -> x\*y )

boolean allEven = intList.stream().allMatch(i -> i % 2 == 0);

boolean oneEven = intList.stream().anyMatch(i -> i % 2 == 0);

boolean noneMultipleOfThree = intList.stream().noneMatch(i -> i % 3 == 0);

# Cassandra

ב Cassandra אין נורמליזציה ואפשר לשמור נתונים מספר פעמים. לכן, ועל מנת לאפשר שליפה מהירה, היינו מאחסנים את הנתונים בצורה שיישלפו מיידית בעת קריאה. למשל, טבלה של חדרים ומספרי טלפון של האחראיים על המדפסות באותו חדר.

# tf-IDF

1. כיוון שמדובר בערך שמשמש רק לדירוג אפשר להשתמש בכל בסיס שרוצים (כל עוד משתמשים תמיד באותו הבסיס). תוצאות הדירוג יצאו זהות לחלוטין בכל בסיס.

# Neo-4J

MATCH (m:Person {name:'Michal'})-[:Posted]->(po:Post)

WITH COLLECT(po) as posts

MATCH (p:Person)

Where ALL (post in posts WHERE (p)-[:Likes]->(post))

RETURN p2.name

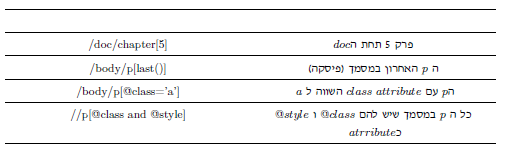
# XQuery

1. LET
2. FOR
3. WHERE
4. Order BY
5. RETURN
6. **f**or $x in /breakfast\_menu/food
7. **l**et $avg := avg(/breakfast\_menu/food /calories)
8. **w**here $x/calories > $avg
9. order by $x/price
10. **r**eturn $x/name

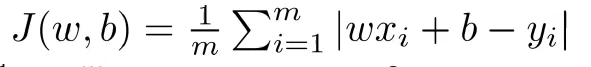
# XPath

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי



# Naïve Bayes

1. F – measures
2. Naïve bayes formula p(y|x)
3. Accuracy , recall, precision
4. Y = wx + b 🡪 line formula, y will be the result
5. We have a lot of features, so we need to train our equation
6. LOSS FUNCTION – 

m – num of features , (x,y) are the data we already have and training on , we want to find w and b that will give us the lowest result (minimize).



1. Gradient Descent

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

(h(x) – y) represents the difference between where my line think the answer is and where the answer actually is.

We’ll do that for every x , find out the average answer , multiply it with alpha .

* p(y|x) = p(y)\*p(x|y) / p(x)
* we don’t calculate p(x) in Naïve bayes because we are looking for the most likely probability (y),

hence there is no importance for distributing in p(x) (which is permenant)

and p(x) is permenant for each y.