Creating a Weighted Text Index

Axel Sirota
MACHINE LEARNING ENGINEER
@AxelSirota

Text Indexes in MongoDB

Creating a Text Index

```
> db.rent.createIndex({“name”: "text"})

> db.rent.getIndexes()
[
...
   {
...,
       "name" : "name_text",
       "ns" : "pluralsight.rent",
       "weights" : {
          "name" : 1
       },
       "default_language" : "english",
       "language_override" : "language",
       "textIndexVersion" : 3
   }
]
```

Analyzing the Index Usage

```
> db.rent.explain().find({name: "Loft"})
{
   "queryPlanner" : {
...,
       "winningPlan" : {
          "stage" : "COLLSCAN",
          "filter" : {
             "name" : {
                "$eq" : "Loft"
             }
          },
          "direction" : "forward"
       },
...
}
```

Analyzing the Index Usage

```
> db.rent.explain().find({$text: {$search: "Loft"}})
{
...,
        "winningPlan" : {
            "stage" : "TEXT",
            "indexName" : "name_text",
            "parsedTextQuery" : {
                "terms" : [
                    "loft"
                ],
                "negatedTerms" : [ ],
                "phrases" : [ ],
                "negatedPhrases" : [ ]
            },
...,
 }
```

Creating a Compound Text Index

Using two fields

```
> db.movies.createIndex({"name": "text", "space": "text"})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

Or just plug *every* text field!

```
> db.movies.createIndex({"$**": "text"})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

How Text Indexes Work?

{ "_id" : 1, "quote" : "is this a dagger which I see before me." }
{ "_id" : 2, "quote" : "Fortune favors the bold, you see" }

We want to create a text index based on **field quote**

How Text Indexes Work?

Text	Tokenization
is this a dagger which I see before me	[‘Is’, ‘this’, ‘a’, ‘dagger’, ‘which’, ‘I’, ‘see’, ‘before’, ‘me’]
Fortune favors the bold, you see	[‘Fortune’, ‘favors’, ‘the’, ‘bold’, ‘you’, ‘see’]
Mid-gadgets present safe punctuation as .? Right!	[‘Mid’, ‘Gadgets’, ‘present’, ‘safe’, ‘punctuation’, ‘as’, ‘Right’]

More on tokenization here

How Text Indexes Work?

Tokenization	Tokenization without stop words
[‘Is’, ‘this’, ‘a’, ‘dagger’, ‘which’, ‘I’, ‘see’, ‘before’, ‘me’]	[‘dagger’, ‘see’]
[‘Fortune’, ‘favors’, ‘the’, ‘bold’, ‘you’, ‘see’]	[‘Fortune’, ‘favors’, ‘bold’, ‘see’]

More on stop words per language here

How Text Indexes Work?

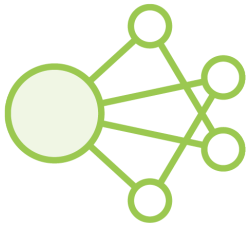
Tokenization without stop words	Stemming
[‘dagger’, ‘see’]	[‘dag’, ‘see’]
[‘Fortune’, ‘favors’, ‘bold’, ‘see’]	[‘fortune’, ‘favor’, ‘bold’, ‘see’]

How Text Indexes Work?

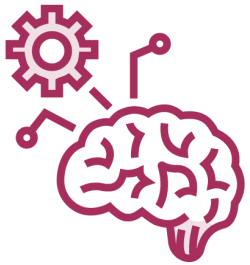
Word	ID
bold	2
dag	1
favor	2
fortune	2
see	[1, 2]

How Text Indexes Work?

TOKENIZE



STOP WORDS



STEMMING

SENSITIVENESS

How Text Indexes Work?

FINAL INDEX

How Text Indexes Work?

This final structure is a **hash table**

As this table grows the lookups **stop being O(1)**

As it grows, **inserting throughout** will be compromised!

Text indexes are a special type of multi key index

Text indexes are characterized by: fields, language, case and diacritic insensitive

One unique text index per collection

Text indexes grow exponentially as more fields are involved

Optimizing Our Indices, Partitioning, and Weighted Indices

Finding Lofts in Tribeca

```
db.rent.find({$text: {$search: "tribeca loft"}},
{_id:0, name:1, neighbourhood_cleansed:1})
{
   "name" : "Tribeca Loft",
   "neighbourhood_cleansed" : "Tribeca",
   "score" : 1.5
}
{
   "name" : "large Tribeca loft",
   "neighbourhood_cleansed" : "Chinatown",
   "score" : 1.3333333333333333
}
{
   "name" : "Artist Loft in Tribeca",
   "neighbourhood_cleansed" : "Tribeca",
   "score" : 1.3333333333333333
}
```

- **# We search "tribeca loft"**
- **# "Tribeca Loft" is first! Makes sense...**
- **# This one doesn't make sense! Chinatown??!**

Adding a Weighted Index

```
> db.rent.createIndex({"name": "text",
"neighbourhood_cleansed": "text",
"description": "text"}, {weights: {name:5,
neighbourhood_cleansed:10, description:1},
name: "my_awesome_index"})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

- **# Note {weigths: {FIELD: VALUE}}**
- **# We can specify names!**

Finding Lofts in Tribeca

```
> db.rent.find({$text: {$search: "tribeca loft"}},
{_id:0, name:1, neighbourhood_cleansed:1,
score: {$meta: "textScore"}}).sort({score:
{$meta: "textScore"}}).limit(5).pretty()
{
    "name" : "Tribeca Loft",
    "neighbourhood_cleansed" : "Tribeca",
    "score" : 19.54
}
{
    "name" : "Stunning Sundrenched Tribeca
Loft",
    "neighbourhood_cleansed" : "Tribeca",
    "score" : 19.225
}
```

◀ **# Much better!**

How Do Weighted Indexes Work?

```
{
   "name" : “Authentic and Open Tribeca Loft",
   "description" : "Awesome loft in the heart of NY",
   "neighbourhood_cleansed" : "Tribeca"
}
```

HOW DO WE CALCULATE THE SCORE?

How Do Weighted Indexes Work?

Indexed Field	Internal Score (TF-IDF)
Name	0.32
Description	0.43
Neighborhood	0 (No match)

Normal Index result: 0.32 + 0.45 + 0 = 0.75

How Do Weighted Indexes Work?

Indexed Field	Internal Score (TF-IDF)	Field Weight	Relative Score
Name	0.32	10	3.2
Description	0.43	5	2.15
Neighborhood	0 (No match)	1	0

Weighted Index result: 3.2 + 2.15 + 0 = 5.35

A Performance Tale

```
db.rent.find({$text: {$search: "Loft"}})
```

Another Approach: Partitioning

```
> db.rent.find({neighbourhood_cleansed: "Tribeca"}, {_id:0, "name":1,
“neighbourhood_cleansed":1}).limit(5)

{ "name" : "Apartment in Tribeca, NYC Panoramic VIEWS~",
"neighbourhood_cleansed" : "Tribeca" }
{ "name" : "Bright Luxury in Tribeca & Views!",
"neighbourhood_cleansed" : "Tribeca" }
{ "name" : "TriBeCa Amazing River View Loft 3BR",
"neighbourhood_cleansed" : "Tribeca" }
{ "name" : "Everyone who stays leaves happy!",
"neighbourhood_cleansed" : "Tribeca" }
{ "name" : "2 BED TriBeCa, Beautiful-Renovated!",
"neighbourhood_cleansed" : "Tribeca" }
```

```
> db.rent.find({$text: {$search: "\"tribeca\" loft"}}, {_id:0, name:1, neighbourhood_cleansed:1}).limit(5)
{ "name" : "Loft", "neighbourhood_cleansed" : "Tribeca" }
...
{ "name" : "Designer Loft", "neighbourhood_cleansed" : "Chinatown" }
{ "name" : "Artist Loft in Tribeca", "neighbourhood_cleansed" : "Tribeca" }

> db.rent.find({"neighbourhood_cleansed": "Tribeca", $text: {$search: "loft"}}, {_id:0, name:1, neighbourhood_cleansed:1}).limit(5)
{ "name" : "Loft", "neighbourhood_cleansed" : "Tribeca" }
...
{ "name" : "Loft with View of Duane Park", "neighbourhood_cleansed" : "Tribeca" }
{ "name" : "Deluxe Tribeca Loft", "neighbourhood_cleansed" : "Tribeca" }
```

Another Approach: Partitioning

- **# Instead of trying to force Tribeca via full text**
- **# We can do exact match on neighbourhood_cleansed.**

Note this query will scan the whole collection!

Another Approach: Partitioning

WITHOUT PARTITION

```
db.rent.explain("executionStats").find({"neighbourhood_cleansed": "Tribeca", $text: {$search: "loft"}}, {_id:0, name:1, neighbourhood_cleansed:1}).limit(5)
{
...
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 5,
        "executionTimeMillis" : 0,
        "totalKeysExamined" : 85,
        "totalDocsExamined" : 170,...
}
```

WITH PARTITION

```
db.rent.explain("executionStats").find({"neighbourhood_cleansed": "Tribeca", $text: {$search: "loft"}}, {_id:0, name:1, neighbourhood_cleansed:1}).limit(5)
{
...
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 5,
        "executionTimeMillis" : 1,
        "totalKeysExamined" : 5,
        "totalDocsExamined" : 5,...
}
```

30x DIFFERENCE!

Index Partitioning

This is called **index partitioning**

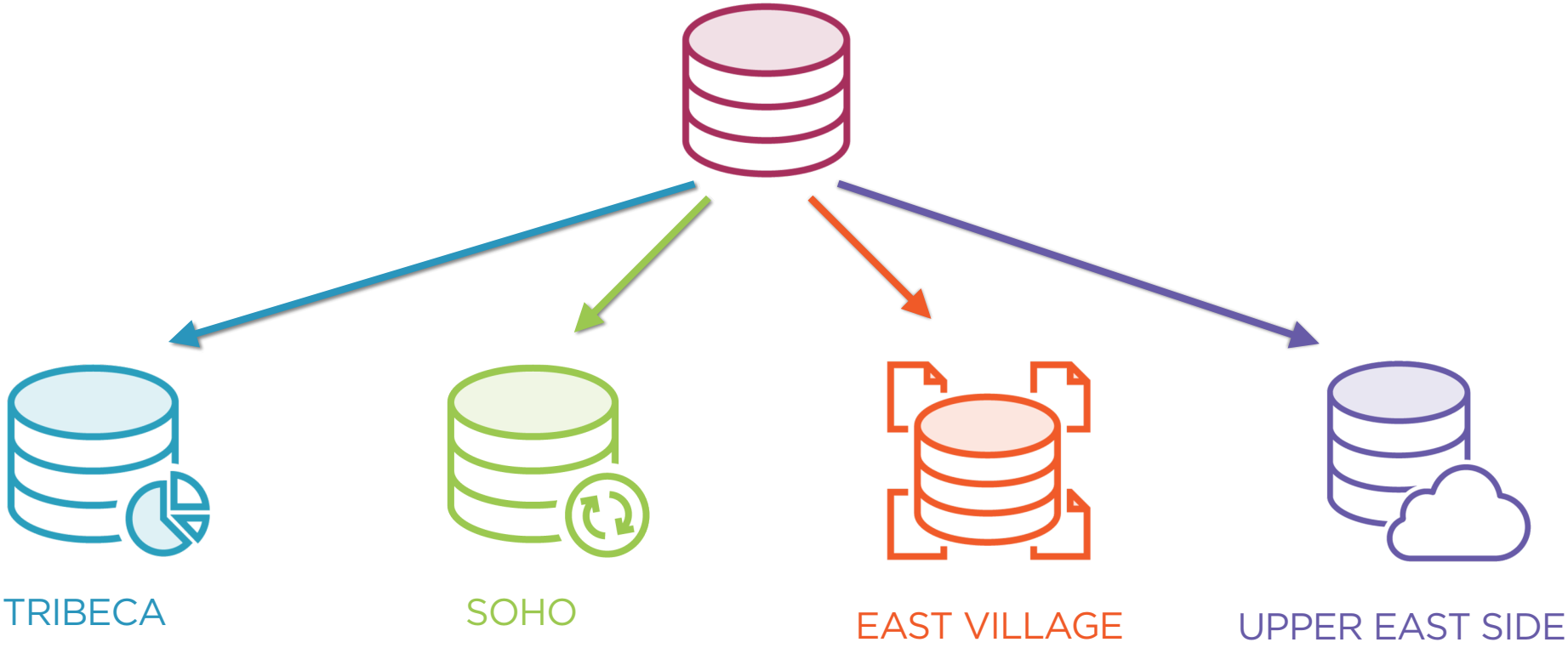
If text searches can be supedited to non text queries in order to reduce the space

Example: We don't want just the name of the rental, but that name in CERTAIN NEIGHBORHOODS

Partitioning by Neighborhood

Value	IDs
Tribeca	[1,4]
SoHo	[2]
East Village	[3,6]
Upper East Side	[5]

Partitioning by Neighborhood



Partitioning by Neighborhood: SoHo

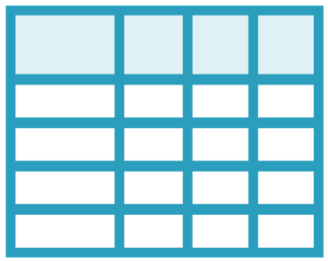
Word	IDs
america	[1,4]
see	[1]
tail	[1]

Partitioning by Neighborhood

```
find({"neighbourhood_cleansed":
"Tribeca", $text: {$search: "loft"}})
```

Tribeca

NEIGHBORHOOD TABLE

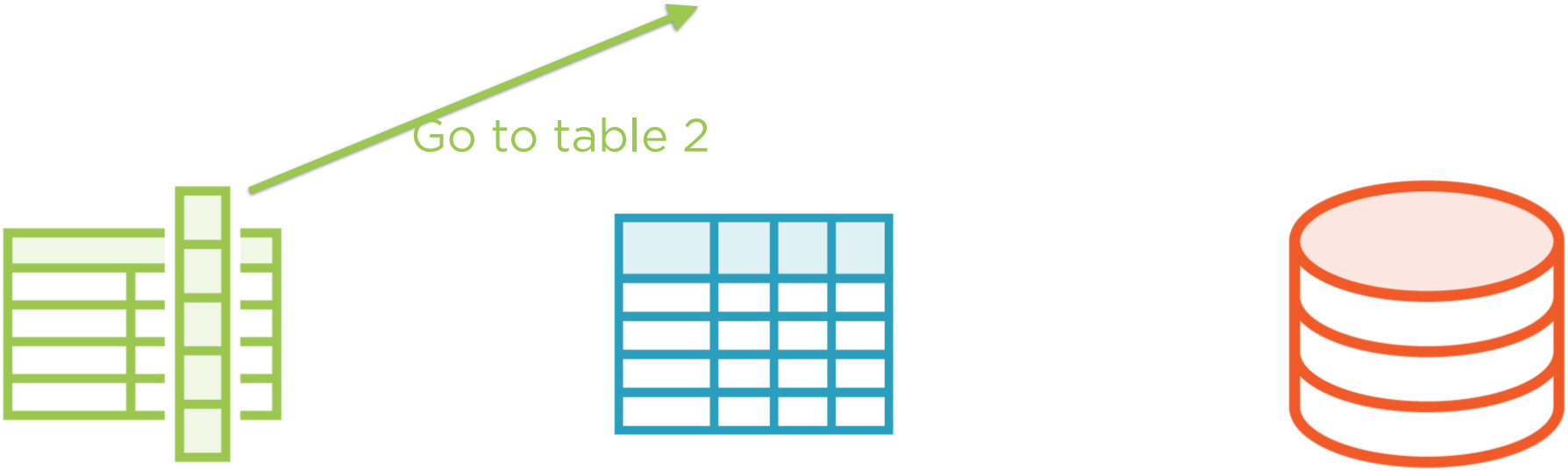


INDEX FOR TRIBECA

DB

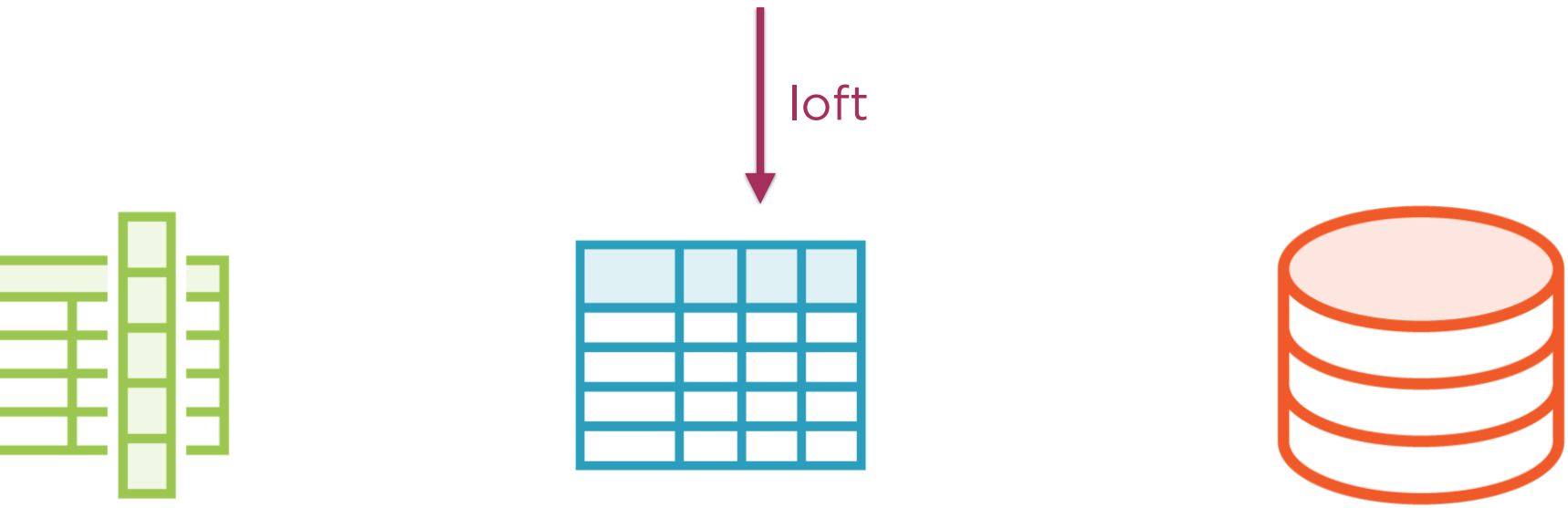
Partitioning by Neighborhood

```
find({"neighbourhood_cleansed":
"Tribeca", $text: {$search: "loft"}})
```



Partitioning by Neighborhood

```
find({"neighbourhood_cleansed":
"Tribeca", $text: {$search: "loft"}})
```

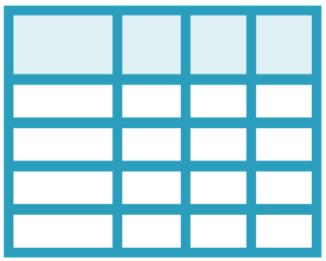


Partitioning by Neighborhood

```
find({"neighbourhood_cleansed":
"Tribeca", $text: {$search: "loft"}})
```

Fetch
IDs
[1,4]

NEIGHBORHOOD TABLE



INDEX FOR TRIBECA

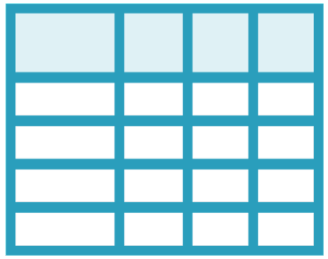
DB

Partitioning by Neighborhood

```
find({"neighbourhood_cleansed":
"Tribeca", $text: {$search: "loft"}})
```

Fetch IDs [1,4]

NEIGHBORHOOD TABLE



INDEX FOR TRIBECA

DB

Demo

We will run a search query over a collection

We will create a weighted index

We will optimize our index with partitioning

Summary

Text indexes score the documents by adding up the internal scores

Weighted indexes change the scoring

Partition the text index by exact match fields in order to optimize queries