MongoDB 2.4 and Text Search

Derick Rethans – derick@10gen.com – @derickr

10gen | the MongoDB company

mongoDB

Agenda

- What is MongoDB Text Search?
- How does text search work?
- Operational considerations
- Under the hood

What is Text Search?

What Does Text Search Do?

Examples

- Find all the blog postings that contain a comment mentioning “OpenStreetMap”
- Find all the profiles where the description is relevant to “databases”, return the profile date
- Find all the product metadata with description relevant to “Nikon S9500”

When to Use MongoDB Text Search

- Adding text search to an existing db application
- Simplify application search-db architecture
- Not a substitute for a dedicated search application (ElasticSearch)

MongoDB Text Search Strengths

- Consistent – integrated into db kernel
  - Works with sharding and replication
- Real-time
  - Index immediately reflects all changes
- Very easy to use - push of a button
- Basic European language support
- Covering index capabilities

Stemming and Stop Words

Stemming

Map different grammatical forms to a single form

Stemming Examples

- { walk, walked, walking, walks } ⇒ walk
- { magazine, magazines, magazine's } ⇒ magazine
- { runs, running, run, ran } ⇒ { run, ran }

Stop words

words that are too common to contribute to the document relevance

Examples of English Stop Words

{ am, themselves, of, before, here, while, what's, myself, ought, me, the, into, about, this, do, can't, a, ... }

Using Text Search

Enabling Text Search

Because it is **beta**, Text Search needs to be enabled:

Option 1: on the command line

```
$mongod --setParameter textSearchEnabled=true
```

Option 2: as admin command to mongod or mongos

```
> db.adminCommand( { setParameter: 1, textSearchEnabled: true } )
```

Create a Text Search Index

```
db.t.ensureIndex(
        { title: "text", post: "text" },
        { weights: {title: 10, post: 5} }
);
```

Run a Text Search Command

```
db.t.runCommand( "text", { search: "MongoDB" } );
```

Text Search Example

```
res = t.runCommand( "text" , { search: "first post" } );
```


```
{
        "queryDebugString" : "first|post||||||",
        "language" : "english",
        "results" : [
                {
                        "score" : 11.833333333333332,
                        "obj" : {
                                "_id" : 1,
                                "title" : "1st post",
                                "post" : "this is my first blog entry."
                        }
                },
                {
                        "score" : 7.5,
                        "obj" : {
                                "_id" : 2,
                                "title" : "2nd post",
                                "post" : "this is my second blog entry."
                        }
                }
        ],
        "stats" : {
                "nscanned" : 3,
                "nscannedObjects" : 0,
                "n" : 2,
                "nfound" : 2,
                "timeMicros" : 121
```

Indexing Options

Basic, default weights:

```
tc.ensureIndex( { title: "text", post: "text" } );
```

Explicit weights:

```
tc.ensureIndex( { title: "text", post: "text" },
                { weights: { "title": 10 } } );
```

Wildcard field: text at any depth, default weights:

```
tc.ensureIndex( { "$**": "text" } );
```

Wildcard field: override default weights and explicit weights:

```
tc.ensureIndex( { "$**": "text" },
                { weights: {"$**": 10, post: 5 } } );
```

Language-Specific Stemming

```
t = db.search; t.drop();
t.save( { _id: 1, title: "mi blog", post: "Este es un blog de prueba" } );
t.save( { _id: 2, title: "cuchillos son divertido", post: "Es mi tercer blog stemmed" } );
t.save( { _id: 3, title: "My fourth blog", post: "This stemmed blog is in english" } );
t.ensureIndex( { title: "text", post: "text" }, { default_language: "spanish" } );
```

Text Search Query Syntax

summer **OR** olympics (or strongly preferred: both)

```
t.runCommand( "text", "Summer Olympics" ) );
```

phrase "Summer Olympics"

```
t.runCommand( "text", "\"Summer Olympics\"" ) );
```

phrase "wild flowers" **OR** Sydney

```
t.runCommand( "text", "\"wild flowers\" Sydney" ) );
```

wild **AND** flowers

```
t.runCommand( "text", "\"wild\" \"flowers\"" ) );
```

industry **ANDNOT** Melbourne **ANDNOT** Physics

```
t.runCommand( "text", "industry -Melbourne -Physics" ) );
```

Text Search Query Options

```
db.collection.runCommand( "text",
    {
        // OR, AND, PHRASE, NEGATION
        search: "coffee",

        // post-search ‘find’ filter
        filter: { about: /desserts/ },

        // result limit
        limit: 2,

        // result projection, 1/0 = include/exclude
        project: { comments: 1, _id: 0 },

        // default stemmer override
        language: "english"
    }
)
```

Summary

Summary

- How does it work?
  - Specify as many fields as you like for text indexing
  - Each field can have an associated weight
  - Individual result documents are relevance-ordered by score
  - Supports basic boolean queries: OR, AND, NOT, PHRASE

Summary

- Operational considerations
  - Works with sharding and replication
  - You can have one text index per collection
  - The text index has an associated default language

Summary

- Under the hood
  - Optional search arguments include project and filter
  - It’s a command: results are wrapped in a single return document
  - Text strings generate stemmed, non-stop-word, weighted ‘terms’
- The text index acts like a multi-key index on terms

Slides:
http://derickrethans.nl/talks/mongo-text-manchester13

Derick Rethans—@derickr—derick@10gen.com