ТЕХНОТРЕК

Лекция 7

Поисковые движки. Elasticsearch

Найдется всё или хотя бы что-то

Поисковые платформы

Elasticsearch

- Open source (1185 контрибьюторов)
- Масштабируемость и отказоустойчивость
- Удобный API
- Гибкие настройки
- Динамический маппинг
- Геопоиск
- CJK

Много незнакомых слов

Морфорлогия

Стемминг

Нечеткий поиск

Лемматизация

N-грамма

Elasticsearch концепты сверху

- Нода
- Кластер
- Шард
- Реплика

Elasticsearch концепты внутри

- Индекс
- Тип
- Документ
- Поле
- Отображение (mapping)
- Query DSL

Анализаторы

Цель - из входной фразы получить список токенов, которые максимально отражают ее суть

Пример анализатора

```
PUT /your-index/_settings
{
  "index": {
    "analysis": {
      "analyzer": {
        "customHTMLSnowball": {
         "type": "custom",
          "char_filter": [
            "html_strip"
          ],
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "stop",
            "snowball"
          ]
}}}}}
```

Расстояние Левенштейна

(редакционное расстояние, дистанция редактирования) — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Цены операций могут зависеть от вида операции

w(a, b) — цена замены символа a на символ b
w(ε, b) — цена вставки символа b
w(a, ε) — цена удаления символа a

Частный случай задачи - Расстояние Левенштейна

w(a, a) = 0
w(a, b) = 1 при a≠b
w(ε, b) = 1
w(a, ε) = 1

Установка Elasticsearch

Prerequisites:
иметь установленную Java >= version 7

https://www.elastic.co/downloads/elasticsearch

bin/elasticsearch

http://localhost:9200/

pip install elasticsearch

Mappings

```
PUT my_index
{
  "mappings": {
    "_doc": {
      "properties": {
        "title":    { "type": "text"  },
        "name":     { "type": "text"  },
        "age":      { "type": "integer" },
        "created":  {
          "type":   "date",
          "format": "strict_date_optional_time||epoch_millis"
        }
      }
    }
  }
}
```

Создание и заполнение индекса

```
PUT http://localhost:9200/blogs

{
   "settings" : {
      "index" : {
         "number_of_shards" : 5, "number_of_replicas" : 3
      }
   }
}

POST http://localhost:9200/blogs/_bulk

{
   "index":{
      "_index":"blogs", "_type":"post", "_id":"10"
   }
}
{
   "title":"Test1", "description":"First test description"
}
```

Получение результатов

```
GET http://localhost:9200/schools/school/1

GET http://localhost:9200/index1,index2,index3/_search

{
    "query" : {
        "match" : { "title": "test" }
    }
}

GET http://localhost:9200/_search?q=name:central
```

Синтаксис запросов

```
+ signifies AND operation
| signifies OR operation
- negates a single token
" wraps a number of tokens to signify a phrase for searching
* at the end of a term signifies a prefix query
( and ) signify precedence
~N after a word signifies edit distance (fuzziness)
~N after a phrase signifies slop amount
```

Внедряем в приложение

```
# читаем описание клиента Elasticsearch

https://elasticsearch-py.readthedocs.io/en/master/api.html
```

Внедряем в приложение

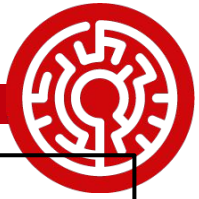
```
from elasticsearch import Elasticsearch

es = Elasticsearch('http://localhost:9200')

def index_instance(obj, index='index', doc_type='default'):
    es.index(
        index=index,
        doc_type=default,
        body=serialize_indexable(obj)
    )

def search_in_index(text, index='index', doc_type='default'):
    es.search(
        index=index,
        doc_type=doc_type,
        body={
            'query': {'match': {'text': text}}
        }
    )
```

Внедряем в приложение. Вариант 1

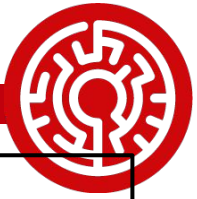


```
class Interval(db.Model):
    id = Column(Integer, primary_key=True)
    start = Column(Integer, nullable=False)
    end = Column(Integer, nullable=False)

    @hybrid_property
    def length(self):
        return self.end - self.start

    @hybrid_method
    def contains(self, point):
        return (self.start <= point) & (point <= self.end)
```

Внедряем в приложение. Вариант 2

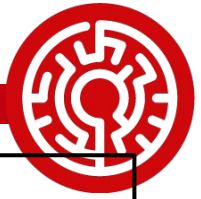


```
from flask import current_app

def add_to_index(index, model, doc_type):
    payload = {}
    for field in model.__searchable__:
        payload[field] = getattr(model, field)
   es.index(index=index, doc_type=doc_type, body=payload)

def query_index(index, doc_type, query, page, per_page):
    search = es.search(
        index=index, doc_type=doc_type,
        body={
             'query': {
                 'multi_match': {'query': query, 'fields': ['*']}
             },
             'from': (page - 1) * per_page, 'size': per_page
         }
    )
    ids = [int(hit['_id']) for hit in search['hits']['hits']]
    return ids, search['hits']['total']
```

Миксин

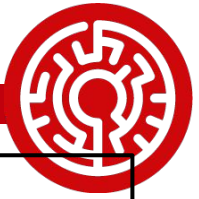


```
from app.search import add_to_index, remove_from_index, query_index

class SearchableMixin(object):
    @classmethod
    def search(cls, expression, page, per_page):
        ids, total = query_index(cls.__tablename__, expression, page,
per_page)
        if total == 0:
            return cls.query.filter_by(id=0), 0
        when = []
        for i in range(len(ids)):
            when.append((ids[i], i))
        return cls.query.filter(cls.id.in_(ids)).order_by(
            db.case(when, value=cls.id)), total

    @classmethod
    def before_commit(cls, session):
        session._changes = {
            'add': list(session.new),
            'update': list(session.dirty),
            'delete': list(session.deleted)
        }
```

Миксин



```
#class SearchableMixin(object):
    @classmethod
    def after_commit(cls, session):
        for obj in session._changes['add']:
            if isinstance(obj, SearchableMixin):
                add_to_index(obj.__tablename__, obj)
        for obj in session._changes['update']:
            if isinstance(obj, SearchableMixin):
                add_to_index(obj.__tablename__, obj)
        for obj in session._changes['delete']:
            if isinstance(obj, SearchableMixin):
                remove_from_index(obj.__tablename__, obj)
        session._changes = None

    @classmethod
    def reindex(cls):
        for obj in cls.query:
            add_to_index(cls.__tablename__, obj)

db.event.listen(db.session, 'before_commit', SearchableMixin.before_commit)
db.event.listen(db.session, 'after_commit', SearchableMixin.after_commit)
```

Домашнее задание

- Написать функцию, которая будет считать расстояние Левенштейна между двумя словами
- Развернуть и наполнить тестовыми данными Elasticsearch
- Реализовать поиск по пользователям, чатам и сообщениям (прикрутить к фронтенду)

ТЕХНОТРЕК

Спасибо за внимание!