Search and Nosql for an information management application

@LucianPrecup
@nosqlmatters #nosql14
2014-04-30
whoami

• CTO of Adelean
• Integrate search, nosql and big data technologies to support ETL, BI, data mining, data processing and data visualization use cases.
Poll - How many of you are ...

- Familiar with Elasticsearch or Solr?
- Using Hibernate or other ORM framework? Hibernate search?
- Familiar with NoSQL?
- Developing information management applications?
- Familiar with SOA? Developing web services?
- Familiar with ETL and BI?
Conventions used in the slides

- Agent / Component
- Human Agent
- Cluster
- Storage

Read or Write
Read or Write Asynchronous
Update (Modification)

Communication through a communication channel
Communication with request and data direction

Protocol Boundary
Information management application?

- Backed by a relational database
- CRUD operations
- The user is generally an employee of the company
Enterprise information management

- ORM Mapping (like Hibernate)
- The relational database is the central point
SOA and information management

- The service layer becomes the central point
Information management, ETL and BI

- Batch extraction, transformation and loading during non-peak periods
User interface

User interface

http://www.marvelsoft.co.in/images/stories/screenshot.png
Adding "Google like" functionality

• Operators are using the applications and
  – Are asking customers for information
  – Need to be fast
  – Sometimes make typing errors
• Customers
  – Give information to operators (sometimes over the phone)
  – Do not know their id (contract id, customer id) by heart
  – Expect operators to be fast and accurate
Auto-completion (result suggestion) and highlighting

"Full text" search

Order all the results on different criteria

Display the total number of results

Fuzzy searches and spelling suggestions

Display heterogeneous results

Advanced search

Faceting for additional search criteria

Display heterogeneous results

Server side result pagination

2014-04-30 @LucianPrecup @nosqlmatters #nosql14
SELECT * WHERE FIRST_NAME LIKE ...% OR ...
if (text contains searchTerm) then replace text with <b>text<b>

Split searchText into tokens
SELECT * WHERE (FIRST_NAME = token1 AND LAST_NAME = token2 AND ZIP_CODE = token3) OR (FIRST_NAME = token2 AND LAST_NAME = token1 AND ZIP_CODE = token3) OR ...

SELECT COUNT(*) FROM ...

SELECT * FROM ...
WHERE ...
ORDER BY ...

O.K. Simpler

SELECT COUNT(*) FROM ...

SELECT NULL as COMPANY_ID, ...
FROM PERSON
UNION ALL
SELECT NULL as BIRTH_DATE, ...
FROM COMPANY

SELECT CUSTOMER_STATUS, COUNT(*) FROM ...
GROUP BY CUSTOMER_STATUS

SELECT CUSTOMER_TYPE, COUNT(*) FROM ...
GROUP BY CUSTOMER_TYPE

SELECT ... FROM ... WHERE ... AND ROWNUM > 0 AND ROWNUM <= 10
Indexing with EdgeNGram → very performing "auto-complete" search
Highlighting handled by the search engine

"Full text" search available by default
"Structured" search also available
Faster (cached) sort
Fast and pertinent fuzzy searches and suggesters

Hits count available with the results
"Schema free" right!
Search facets calculated with the results. Almost no additional cost.

Pagination handled natively ("I am feeling lucky" !)
A search engine for information management?

- User experience
  - "Full text" search and "Google like" navigation
  - Direct access to data
  - Facetted search and navigation
  - "out of the box" pagination
  - Performance → fluidity
  - Search improved with auto-completion and search suggestions

- Additional functionality
  - Fuzzy search
  - Phonetic analysis
  - Spell checking
  - Synonyms and technical terms handling
  - Compound words handling

- Faster than SQL for
  - Sorting
  - Pagination
  - Facets and aggregations
  - Filtering
Auto-completion with EdgeNGram

Indexing

<table>
<thead>
<tr>
<th>Id</th>
<th>Nom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Céline</td>
</tr>
<tr>
<td>2</td>
<td>Celia</td>
</tr>
</tbody>
</table>

Ascii folding → Celine, Celia

Lowercase → celine, celia

EdgeNGram → ce cel celi celin celine

Searching

Nom

Célin

← Search term

Celin ← Ascii folding

celin ← Lowercase

EdgeNGram → ce cel celi celin celine

ce cel celi celia

Index

<table>
<thead>
<tr>
<th>Key</th>
<th>Document id</th>
</tr>
</thead>
<tbody>
<tr>
<td>ce</td>
<td>1, 2</td>
</tr>
<tr>
<td>cel</td>
<td>1, 2</td>
</tr>
<tr>
<td>celi</td>
<td>1, 2</td>
</tr>
<tr>
<td>celin</td>
<td>1</td>
</tr>
<tr>
<td>celine</td>
<td>1</td>
</tr>
<tr>
<td>celia</td>
<td>2</td>
</tr>
</tbody>
</table>

2014-04-30
Auto-completion with EdgeNGram

Indexing

<table>
<thead>
<tr>
<th>Id</th>
<th>Nom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Céline</td>
</tr>
<tr>
<td>2</td>
<td>Celia</td>
</tr>
</tbody>
</table>

Ascii folding: Celine, Celia
Lowercase: celine, celia

EdgeNGram: ce cel celi celin celine
ce cel celi celia

Searching

Search term: Célin
Ascii folding: Celin
Lowercase: celin

Index

<table>
<thead>
<tr>
<th>Key</th>
<th>Document id</th>
</tr>
</thead>
<tbody>
<tr>
<td>ce</td>
<td>1, 2</td>
</tr>
<tr>
<td>cel</td>
<td>1, 2</td>
</tr>
<tr>
<td>celi</td>
<td>1, 2</td>
</tr>
<tr>
<td>celin</td>
<td>1</td>
</tr>
<tr>
<td>celine</td>
<td>1</td>
</tr>
<tr>
<td>celia</td>
<td>2</td>
</tr>
</tbody>
</table>
Information management and search

- Enter "polyglot persistence"
- How to synchronize the two databases?
- Triggers? Hard to implement
  - Especially for external triggers
  - And for large transactions (when many tables are concerned)
- How to provide seamless integration for the application?
- Luckily we have the Service Layer ...
SOA and a search engine

- The Service Layer ensures seamless integration for applications
- The Service Layer can help with the synchronization: Creates, Updates, Deletes are passing through
- But how to deal with legacy applications and batch processes?
Search integration in enterprise environment

- Hooks in the Service Layer to intercept Creates, Updates, Deletes
- Batch indexing to catch up with legacy applications
- The Search Services Layer ensures seamless integration and security
Search integration in enterprise environment

- **Hooks into the Services Layer**
  - Send messages asynchronously to Elasticsearch

- **Real time indexing for everything that passes through the Services Layer**
  - If messages are self contained proceed with indexing, if not use a Read operation on the Service Layer

- **Batch indexing to catch up with legacy applications**
  - Delta extraction
  - Full database extraction

- **Search Services Layer is important**:  
  - Constructs Elasticsearch queries and supplies a high level interface, ready to integrate into applications
  - Acts as a proxy to Elasticsearch, ensuring authentication and authorizations
The world of a search engine is flat

Normalized database

<table>
<thead>
<tr>
<th>film</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
</tr>
<tr>
<td>id</td>
</tr>
<tr>
<td>title</td>
</tr>
<tr>
<td>published</td>
</tr>
<tr>
<td>genre</td>
</tr>
<tr>
<td>language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>person_film</th>
</tr>
</thead>
<tbody>
<tr>
<td>FK1</td>
</tr>
<tr>
<td>filmId</td>
</tr>
<tr>
<td>role</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>personId</td>
</tr>
<tr>
<td>id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>person</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK</td>
</tr>
<tr>
<td>id</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>birthPlace</td>
</tr>
</tbody>
</table>

Elasticsearch document

{"film": {
    "id": "183070",
    "title": "The Artist",
    "published": "2011-10-12",
    "genre": ["Romance", "Drama", "Comedy"],
    "language": ["English", "French"],
    "persons": [
        {"person": {"id": "5079", "name": "Michel Hazanavicius", "role": "director" }},
        {"person": {"id": "84145", "name": "Jean Dujardin", "role": "actor" }},
        {"person": {"id": "24485", "name": "Bérénice Bejo", "role": "actor" }},
        {"person": {"id": "4204", "name": "John Goodman", "role": "actor" }}
    ]
}}
How to create the "flattened" document

• Joining data from the previous example:

<table>
<thead>
<tr>
<th>FILMID</th>
<th>TITLE</th>
<th>PUBLISHED</th>
<th>GENRE</th>
<th>LANGUAGE</th>
<th>PERSI</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>183070</td>
<td>The Artist</td>
<td>2011-10-12</td>
<td>Romance,Drame,Comédie</td>
<td>Anglais, Français</td>
<td>5079</td>
<td>Michel Hazanavicius</td>
</tr>
<tr>
<td>183070</td>
<td>The Artist</td>
<td>2011-10-12</td>
<td>Romance,Drame,Comédie</td>
<td>Anglais, Français</td>
<td>84145</td>
<td>Jean Dujardin</td>
</tr>
<tr>
<td>183070</td>
<td>The Artist</td>
<td>2011-10-12</td>
<td>Romance,Drame,Comédie</td>
<td>Anglais, Français</td>
<td>24485</td>
<td>Bérénice Bejo</td>
</tr>
<tr>
<td>183070</td>
<td>The Artist</td>
<td>2011-10-12</td>
<td>Romance,Drame,Comédie</td>
<td>Anglais, Français</td>
<td>4204</td>
<td>John Goodman</td>
</tr>
</tbody>
</table>

• Solr DataImport Handler : Entities and referencing ${parent.ID}
  • "Crawls" the database → very bad performance
• WM_CONCAT, LISTAGG, GROUP_CONCAT
  • Nice workaround if your database allows it
• Elasticsearch jdbc river
  • https://github.com/jprante/elasticsearch-river-jdbc/wiki/Structured-Objects
• Use your own batch !
  • Use your own code to create the document sent to the search engine
  • If you are using Java, you can integrate SpringBatch for example
Solr’s DataImportHandler

```xml
<dataConfig>
  <dataSource driver="org.hsqldb.jdbcDriver" url="jdbc:hsqldb:/temp/example/ex" user="sa" />
  <document name="products">
    <entity name="item" query="select * from item">
      <field column="ID" name="id" />
      <field column="NAME" name="name" />
      <field column="MANU" name="manu" />
      <field column="WEIGHT" name="weight" />
      <field column="PRICE" name="price" />
      <field column="POPULARITY" name="popularity" />
      <field column="IN STOCK" name="inStock" />
      <field column="INCLUDES" name="includes" />
    </entity>
    <entity name="item_category" query="select CATEGORY_ID from item_category where item_id='${item.ID}'">
      <field column="description" name="cat" />
    </entity>
    <entity name="category" query="select description from category where id = '${item_category.CATEGORY_ID}'">
      <field column="description" name="cat" />
    </entity>
  </document>
</dataConfig>
```

"Crawling" the database → very bad performance
Solr’s DataImportHandler

OK for HTML pages but KO for db relations

---

http://wiki.apache.org/solr/DataImportHandler

2014-04-30 @LucianPrecup @nosqlmatters #nosql14
The issue with joins :-) 

• Let’s say you have two relational entities: Persons and Contracts 
  – A Person has zero, one or more Contracts
  – A Contract is attached to one or more Persons (eg. the Subscriber, the Grantee, …)

• Need a search services :
  – S1: getPersonsDetailsByContractProperties
  – S2: getContractsDetailsByPersonProperties

• Simple solution with SQL:
  SELECT P.* FROM P, C WHERE P.id = C.pid AND C.a = 'A'
  SELECT C.* FROM P, C WHERE P.id = C.pid AND P.a = 'A'
The issue with joins - solutions

- **Solution 1**
  - Index Persons with Contracts together for S1
    ```json
    {"person" : { "details" : ... , "contracts" : ["contract" :{"id" : 1, ...} , ... ] }
    ```
  - Index Contracts with Persons together for S2
    ```json
    {"contract" : { "details" : ... , "persons" : ["person" :{"id" : 1, "role" : "S", ...}]}
    ```

- **Issues with solution 1:**
  - A lot of data duplication
  - Have to get Contracts when indexing Persons and vice-versa

- **Solution 2**
  - Elasticsearch’s Parent/Child

- **Issues with solution 2:**
  - Works in one way but not the other (only one parent for n children, a 1 to n relationship)

- **Solution 3**
  - Index Persons and Contracts separately
  - Launch **two Elasticsearch queries** to get the response
  - For S1: First get all Contract ids by Contract properties, then get Persons by Contract ids (terms query)
  - For S2: First get all Persons ids by Person properties, then get Contracts by Person ids (terms query)
  - The response to the second query can be returned “as is” to the client (pagination, etc.)
Let’s say you have a database containing lines of monthly payments by customer companies.
Let’s say you need to display company information together with the aggregated income for each company.

With SQL:
```
SELECT Company.id, Company.A, Company.B SUM(Payments.amount) FROM Company, Payments
WHERE Company.id=Payments.cid GROUP BY Company.id, Company.A, Company.B
```

With Elasticsearch:
```
Launch

"facets" : {
"Total" : {
"terms_stats" : {
"key_field" : "Payments.cid",
"value_field" : "Payments.amount"
}
}
}

then
GET /index/Company/_mget

{
"ids" : ["cid1", "cid2", ...]
}

Or, you could use the relational database itself if it is fast enough (IN query)
Other tips and tricks

• Define your own mapping for each field (even if you are
tempted by the “schema free”)
  – For example: you might not want to use regular stop words for
    fields representing person names

• Beware of idf
  – Values that are rare are boosted by default. This can lead to
    some “unexpected” behavior.

The query

```json
{"bool": {
    "should": [
        {
            "text": {
                "name": "john"
            }
        },
        {
            "text": {
                "place": "sunnyvale"
            }
        }
    ]
}}
```

Could result in

<table>
<thead>
<tr>
<th>Name</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Doe</td>
<td>Sunnyvale</td>
</tr>
<tr>
<td>Alice O’Malley</td>
<td>Sunnyvale</td>
</tr>
<tr>
<td>John Dean John</td>
<td>Palo Alto</td>
</tr>
</tbody>
</table>
More tips and tricks

• Use index alias when launching the full batch indexing to keep the service running 24/7

• When searching structured information you may have many identical scores. To keep the order and pagination deterministic, use a sort by default. For example first by _score, then by id.

• For user defined sorting (there are many criteria in this kind of use cases: name, user_id, zipcode, location, etc.) be sure to have enough memory. For example sorting on a 34 million records database by user_id (unique, of size 8 bytes) you must count about 260 MB of fielddata cache.

customers_1
size: 1.13G (1.13G)
docs: 1,399,791
(1,399,791)

customers_2
size: 99B (99B)
docs: 0 (0)
Search and BI

- Elasticsearch is already in sync with the main database.
- Kibana is capable of displaying useful information about the data.
- Why not having some BI dashboards in real time?
Example of Kibana dashboard

-- [http://www.elasticsearch.org/overview/kibana/](http://www.elasticsearch.org/overview/kibana/)
Datawarehouse

- Elasticsearch is conceived for Big Data so using it as a Data Warehouse makes sense
NoSQL, Search and RDBMS

- Real time synchronization
- Better performance, better user experience, more functionality
- Direct access to pertinent data
- Facets (simple or complex)
- Auto-completion, suggestions, fuzzy searches, ...
- Traditional BI alternative
Elasticsearch integration – global view
Why not NoRDBMS?

- Legacy applications
- The need for transactions and consistency
- The need for data normalization
- The need for structure
- Polyglot persistence is better
- NoSQL databases are lacking features for now (see keynote from @ted_dunning)
Thank you

Q & A