Search-based business intelligence and reverse data engineering with Apache Solr

Mario-Leander Reimer
Chief Technologist
This talk will ...

- Give a brief overview of the AIR system’s architecture
- Show reverse data engineering using Solr and MIR
- Talk about the fight for our right to Solr
- Describe solutions for the problem of combinatorial explosion
- Outline a flexible and lightweight ETL approach for Solr
Let’s go back to when it all began ...

Source: http://www.october212015.com/images/timecircuits.jpg
The project vision: find the right information in less than 3 clicks.

The situation:
- Users had to use up to 7 different applications for their daily work.
- Systems were not really integrated nicely.
- Finding the correct information was laborious and error prone.

The idea:
- Combine the data into a consistent information network.
- Make the information network and its data searchable and navigable.
- Replace existing application with one easy to use application.
But how do we find the originating system for the desired data?

Where to find the vehicle data?
60 potential systems and 5000 entities.

- System A
- System B
- System C
- System D

Vehicle data
Other data
And how do we find the hidden relations between the systems and their data?

How is the data linked to each other?

400,000 potential relations.
Meta Information Research (MIR)

Source: http://www.thewallpapers.org/photo/31865/Mir_space_station_12_June_1998.jpg
MIR is a simple and lightweight data reverse engineering and analysis tool based on Solr.

- MIR manages meta information about the source systems (the data models and record descriptions)
- MIR allows to navigate and search in the metadata, you can drill into the metadata using facets
- MIR also manages the target data model and Solr schema description
Search results

Tree view of systems, tables and attributes

Facetted drill down

Wildcard queries

Found potential synonyms for the chassis number

- Faceted drill down
- Wildcard queries
- Tree view of systems, tables and attributes
- Search results

Found potential synonyms for the chassis number.
EAT YOUR OWN DOG FOOD.
The AIR Solr schema definition is modelled and defined within MIR.
def sourceGenerator = MIR + Solr + Maven;

```xml
<plugin>
  <groupId>com.bmwi.ispi.air.central</groupId>
  <artifactId>mir-generator-maven-plugin</artifactId>
  <version>${project.version}</version>
  <configuration>
    <systemName>${mir.systemName}</systemName>
    <solrSolrHome>${basedir}/../..//air-tools/mir-generator-maven-plugin/src/main/resources</solrSolrHome>
    <artefacts>
      <javaArtefact>
        <className>{0}Et</className>
        <packageName>com.bmwi.ispi.air.central.fahrzeug.core</packageName>
        <templateName>SolrEntity.ftl</templateName>
        <tableName>Fahrzeug</tableName>
        <strategy>SINGLE_TABLE</strategy>
      </javaArtefact>
    </artefacts>
  </configuration>
  <executions>
    <execution>
      <id>generate-sources</id>
      <phase>generate-sources</phase>
      <goals>
        <goal>generate-sources</goal>
      </goals>
    </execution>
  </executions>
</plugin>

@Generated("MIR Generator")
public class FahrzeugEt extends AbstractEt {

  @Field(SolrFieldNames.VIN)
  private java.lang.String vin;

  // more fields ...
```
But Solr is a full text search engine. You have to use an Oracle DB for your application data!

**NO!**
Some of the AIR requirements were ...

- Focus is on search. Transactions are not required.
- High demands on request volume and performance.
- Free navigation on data model and content.
- Support for full text search and facetted search.
- Offline capabilities.
- Scalability from low-end device to server to cloud.
Apache Solr outperformed Oracle significantly in query time as well as index size.

SELECT * FROM VEHICLE WHERE VIN='V%' | 383 ms | 384 ms | 383 ms
INFO_TYPE:VEHICLE AND VIN:V* | 38 ms | 0 ms | 0 ms

SELECT * FROM VEHICLE WHERE VIN='%X%'
INFO_TYPE:VEHICLE AND VIN:*X*

SELECT * FROM MEASURE WHERE TEXT='engine'
INFO_TYPE:MEASURE AND TEXT:engine

Test data set: 150.000 records
Disk space: 132 MB Solr vs. 385 MB Oracle
Dirt Race Use Case:

- Low-end devices
- No Internet

Running Solr and AIR-2-Go on Raspberry Pi Model B worked like a charm.

Model B Hardware Specs:
- ARMv6 CPU at 700Mhz
- 512MB RAM
- 32GB SD-Card

Running Debian Linux + JDK8
Jetty Container with the Solr and AIR WARs deployed
Reduced Solr data set with approx ~1.5 Mio documents

And now try this with Oracle!
YOU GOTTA FIGHT FOR
YOUR RIGHT TO SOLR!

BEASTIE BOYS
No silver bullet. A careful schema design is crucial for your Solr performance.

```xml
<fieldtype name="text" class="solr.TextField" positionIncrementGap="100">
  <analyzer type="index">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.WordDelimiterFilterFactory" splitOnNumerics="1" preserveOriginal="1" generateWordParts="1" generateNumberParts="1"
    catenateWords="1" catenateNumbers="1" catenateAll="1" />
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_de.txt"
    format="snowball" enablePositionIncrements="true"/>
    <filter class="solr.ReversedWildcardFilterFactory" withOriginal="true"
    maxPosAsterisk="3" maxPosQuestion="2" maxFractionAsterisk="0.33"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.WhitespaceTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymFilterFactory" synonyms="lang/synonyms.txt"
    expand="true" ignoreCase="true"/>
    <filter class="solr.WordDelimiterFilterFactory" splitOnNumerics="0" preserveOriginal="1" generateWordParts="0" generateNumberParts="0"
    catenateWords="0" catenateNumbers="0" catenateAll="1" />
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="lang/stopwords_de.txt"
    format="snowball" enablePositionIncrements="true"/>
  </analyzer>
</fieldtype>
```
Naive data denormalization can quickly lead to combinatorial explosion.
Multi-valued fields can efficiently store 1..n relations but may result in false positives.

```
{
  "INFO_TYPE": "AWPOS_GROUP",
  "NUMMER": ["1134190", "1235590"],
  "BAUSTAND": ["1969-12-31T23:00:00Z", "1975-12-31T23:00:00Z"],
  "E_SERIES": ["F10", "E30"]
}

Index 0

q=INFO_TYPE:AWPOS_GROUP AND NUMMER:1134190 AND E_SERIES:F10

Index 1

q=INFO_TYPE:AWPOS_GROUP AND NUMMER:1134190 AND E_SERIES:E30

In case this doesn’t matter, perform a post filtering in your application.

**Note:** latest Solr versions support nested child documents. Use instead.
Technical documents and their validity were expressed in a binary representation.

- Validity expressions may have up to 46 characteristics.
- Validity expressions use 5 different boolean operators (AND, NOT, ...)
- Validity expressions can be nested and complex.
- Some characteristics are dynamic and not even known at index time.

**Solution:** transform the validity expressions into the equivalent JavaScript terms and evaluate these terms at query time using a custom function query filter.
Binary validity expression example.

AND(CHARACTERISTIC(53078923), 53086475),

Type(53078923) = 'Brand', Value(53086475) = 'BMW PKW'

CHARACTERISTIC(53088651), 53161483)

Type(53088651) = 'E-Series', Value(53161483) = 'F10'

CHARACTERISTIC(64555275), 64558475)

Type(64555275) = 'Transmission', Value(53161483) = 'MECH'
Transformation of binary validity terms into their JavaScript equivalent at index time.

AND(Brand='BMW PKW', E-Series='F10', Transmission='MECH')

((BRAND=='BMW PKW')&&(E_SERIES=='F10')&&(TRANSMISSION=='MECH'))

{
"INFO_TYPE": "TECHNISCHES_DOKUMENT",
"DOKUMENT_TITEL": "Getriebe aus- und einbauen",
"DOKUMENT_ART": "reparaturanleitung",
"VALIDITY": "((BRAND=='BMW PKW')&&(E_SERIES=='F10')&&(...))",
"BRAND": ['BMW PKW'],
...
}
The JavaScript validity term is evaluated at query time using a custom function query.

&fq=INFO_TYPE:TECHNISCHES_DOKUMENT
&fq=DOKUMENT_ART:reparaturanleitung
&fq={!frange l=1 u=1 incl=true incu=true cache=false cost=500}

```javascript
jsTerm(VALIDITY,eyJNT1RPUl9LUkFGVFNUT0ZGQVJUX01PVE9SQVJCRUlUU1ZFUkZBSFJFTiI6IkIiLCJFX01BU0NISU5FX0tSQUUU1RPRkZBUlQiO5m51bGwsIlNJQ0hFUkhFSVRTRkFIUlpFVUciOiIwIiwiQU5UklFQiI6IkFXRCIsIkVkJBVVJFSUhFIjoiWCciQ==)
```

Base64 decode

```javascript
{
  "BRAND":"BMW PKW",
  "E_SERIES":"F10",
  "TRANSMISSION":"MECH"
}
```

(((Boolean) SCRIPT ENGINE.eval(javaScriptTerm, bindings)) ? VALID : INVALID);

http://qaware.blogspot.de/2014/11/how-to-write-postfilter-for-solr-49.html
How often do we load data? How do we ensure data consistency?
A traditional approach using a DWH and ETL: too inflexible, heavy weight and expensive.

ETL jobs would usually be implemented with Informatica

Significant business logic required depending on the source database
Flexible and lightweight ETL combined with Continuous Delivery and DevOps.
Let’s go
BACK TO THE FUTURE
Apache Solr has become a powerful tool for data analytics applications. Be creative.

Our next big project using Apache Solr is already on its way. High performance application to predict and calculate the bill of materials for all required parts and orders.

Apache Solr as a compressed, scalable and high performance time series database.

*FOSDEM’15 – Florian Lautenschlager, QAware GmbH*

Leveraging the Power of SOLR and SPARK

*Apache: Big Data 2015 – Johannes Weigend, QAware GmbH*
Business intelligence is about asking the right questions about your data.
And with Apache Solr you can search and find the answers you are looking for.
Mario-Leander Reimer
Chief Technologist, QAware GmbH

https://twitter.com/leanderreimer/
https://slideshare.net/MarioLeanderReimer/
https://speakerdeck.com/lreimer/