> It’s a dangerous business, putting your product to market. You step onto the market, and if you don’t keep your data, there’s no knowing where you might be swept off to.[1]

Picture this – you start with the perfect use case for your data analytics product. You make a great pitch and you sell well. Maybe a little too well. Because now you have many clients. And all of them are asking hard questions: “Can you integrate my data, with my particular format?”, “How well can you scale?”, “How many visualizations do you offer?”.

Nowadays, data analytics doesn’t exist on its own. You have to take care of data extraction, transformation and loading, and of visualization. And you have to be able to automate the processes and absorb continuous updates. How to make sense of all that? How to scale properly?

Luckily, we are here to help.

Through this series of blog posts, we’ll discuss how to best scale and branch out an analytics

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

solution using a knowledge graph technology stack. Our main weapons when beating that beast will be GraphDB, Ontotext Platform, Kafka, Elasticsearch, Kibana and Jupyter. It may look like a daunting task at first, but we'll get through the process step by step, discussing the improvements as your solution naturally grows.

For the use case that this blog will explore, we have picked a perfect blend of the exciting and the fairly boring – building compliance. But with robots. Because everything is cooler with robots.

The core analytics task

At the core of this task, we have a simple question: is a building safe to use? After all, no one wants their new office collapsing on their heads. To that end, there are plenty of standards supported by the ICC, the Eurocodes and many others. Those standards are enforced by people "on the ground" who do the survey in person, from checking joint quality to making sure the building permits are all legitimate. The favourite tool for this would be a checklist or every analyst's first line of defense – a spreadsheet.

All that is well and good, but how do we establish a common format? We can kind of do it within one enterprise, agreeing on certain templates, but when we start going cross-enterprise, or when we start integrating legacy data, it will be a lot of work doing that by hand. Surely, this can be automated?

Well, our hypothetical company, "Large Analytics for Zealous Yields", LAZY, comes to the rescue. LAZY has a novel idea – get all that data and store it in a knowledge graph. They've read some of the many available resources on the topic and seen Ontotext's excellent product demos. But would the surveyors write SPARQL? Sounds unlikely. Ontotext Refine to the rescue.

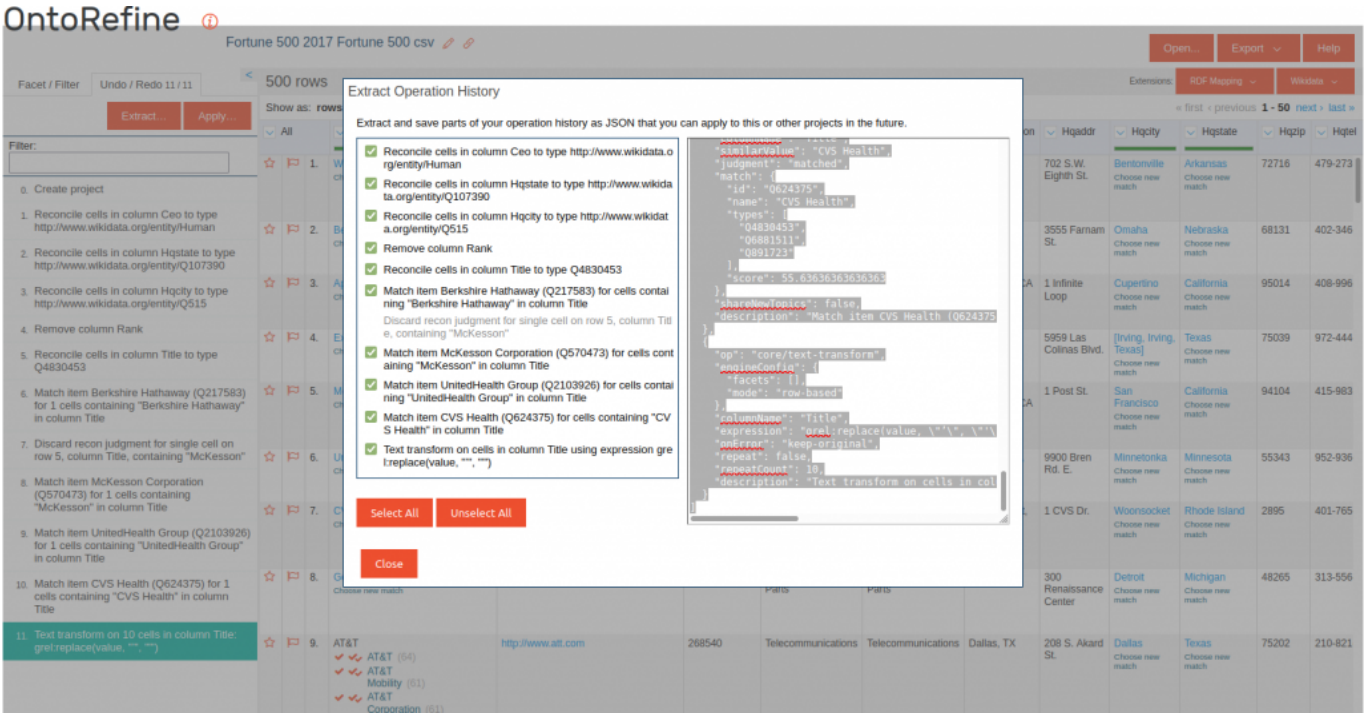
Dealing with spreadsheets via OntoRefine

Surveyors already use spreadsheets. The path of least resistance is to let them use spreadsheets[2] and translate these to your knowledge base. OntoRefine is a data transformation tool that lets you unite plenty of data formats and get them into your triplestore. It is built on the popular open source tool OpenRefine, which means it's always

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

improving.

So, everyone writes slightly different files? No problem, you can define a set of transformations using GREL functions. And the mapping can be made consistent with the intuitive mapping interface. Both of those actions can be exported to JSON and then automated. So, you no longer need to have a very specific template, you can coerce the data into the right format, then export the steps, so next time when you see the same template, you can just repeat the same actions.



OntoRefine in action

Now that the data is in the database, we can start benefiting from the RDF technology's strengths. One of the core upsides of storing your data in that format is inference. Beside survey information, the LAZY database can also contain an ontology. You can think about that as metadata about the data, describing its relationships.

Inferring new knowledge

There can also be inference rules. When an inference rule triggers, a new triple is added,

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

based on what we already explicitly know. Suppose that we have a simple rule for buildings: if two buildings share the same address, then they are the same building. We can express that with a custom rule. GraphDB has a very simple syntax for writing custom inference rules, coupled with a robust materialization engine, allowing you to profile and debug the rules in detail.

So, our rule would be something like this:

```
Prefices
{
     rdf      :  http://www.w3.org/1999/02/22-rdf-syntax-ns#
     lazy     :  http://lazy.org/compliance/
}

Rules {

Id: building_equality
  x rdf:type lazy:Building
  y rdf:type lazy:Building
  x lazy:address z
  y lazy:address z
  ----------------------------
  x rdf:sameAs y
}
```

Since this is a fairly basic example, we can leverage the already existent rules to achieve the same result as well. The standard ontology language OWL defines the so-called inverse functional properties, which can serve as unambiguous identifiers – if two object descriptions include the same value for such property, they are describing one and the same object in real life. This is the case with social security numbers, car plate numbers and the *lazy:address*. The semantics of owl:InverseFunctionalProperty is already supported in all preferined rulesets in GraphDB, whose name starts with “owl-”. The rule implementing its semantics looks like this:

a

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
b a c
d a c [Constraint b != d] [Cut]
-----------------------------------
b  d
```

This means we can just declare lazy:address as inverse functional property in our dataset:

```
lazy:address a owl:InverseFunctionalProperty .
lazy:Building1231 lazy:address lazy:SomeStreet .
lazy:BuildingABC lazy:address lazy:SomeStreet .
```

That way, we can simplify our lives in the future, so when we seek reports for one building (say Building123), we also get information about the “other” building (BuildingABC), which is at the same address.

Ensuring data quality with SHACL

Our previous step covers data format differences. But sometimes things are outright wrong. Everyone makes mistakes. It is possible that an inspector didn’t specify the building location. Or, something less obvious, like a particular building’s material. That would usually require a manual check or perhaps some clever spreadsheet magic – and spreadsheet magic tends to be fallible.

Fortunately, GraphDB has something better. It is called SHACL and you can reject bad reports with it. The report would simply not be imported and a constraint validation error would be displayed, forcing the inspector to get back to it and see what is missing.

Suppose that we have a very simple ontology. We want to enforce that each Compliance Report has an address and that it is made no earlier than 2000.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
@prefix lazy: <http://lazy.org/compliance/> .
@prefix sh:   <http://www.w3.org/ns/shacl#> .

lazy:ComplianceReportShape a sh:NodeShape ;
                           sh:targetClass lazy:ComplainceReport .
sh:property [
      sh:path lazy:address ;
      sh:minCount 1 ;
  ]
  sh:property [
      sh:path lazy:date ;
      sh:datatype xsd:date ;
      sh:minInclusive "2000-01-01"^^xsd:date ;
] .
```

SHACL in action

With RDF4J's advanced targeting, we can have even finer grained control over which constraints we want enforced.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
@prefix lazy: <http://lazy.org/compliance/> .
@prefix sh:   <http://www.w3.org/ns/shacl#> .

lazy:ComplianceReportShape a sh:NodeShape ;
                           sh:targetShape [
                                 sh:property [
                                                  sh:path sh:builtBy ;
                                                  sh:hasValue "ACME Corporation" .
]
        ]
sh:property [
      sh:path lazy:address ;
      sh:minCount 1 ;
  ]
  sh:property [
      sh:path lazy:date ;
      sh:datatype xsd:date ;
      sh:minInclusive "2000-01-01"^^xsd:date ;
] .
```

HACL with RSX RDF4J extensions – when we only want to validate ACME buildings

Wrapping it up – SPARQL queries

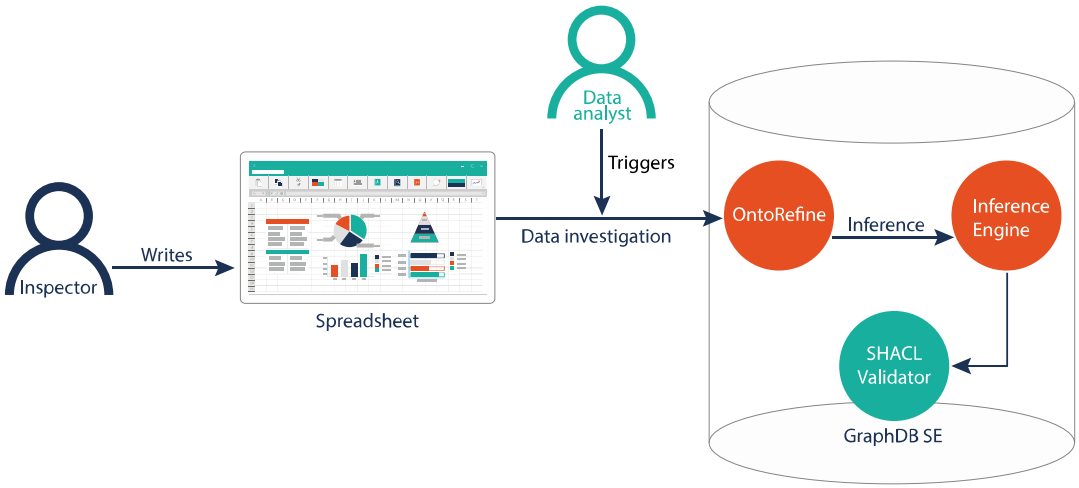
But how about solving the core problem? We have all this data in RDF format. All that we need now is a SPARQL query. So, we want to make sure that no building erected after 1970 uses asbestos? Easy enough.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?building {
    ?building a lazy:Building ;
              lazy:Constructed ?date ;
              lazy:Material ?materials .
    FILTER (?date > "01-01-1971"^^xsd:Date && ?materials IN ("asbestos"))
}
```

If we have a building erected post 1970 that uses asbestos, the query will return the building identifier. And we all agree asbestos is bad.

Conclusion



The LAZY Architecture in its first iteration

Ontotext GraphDB is more than a database. Sure, it keeps your triples and allows you to query them. But it can do so much more. It can help you unite different data sources, keep your data clean and infer new knowledge from your explicit triples.

With this tooling at hand, LAZY has solved its core problem. It can unite diverse models quickly, check for inconsistencies in the data itself and easily detect building code violations.

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com

All looks good for now!

Come back next Friday for our next blog post to see how growth can literally toss a spanner in the wheels – and what tools are already at your disposal to grow organically.

Do you want to solve similar problems specific to your enterprise use case?

Ontotext's GraphDB
Give it a try today!
Download Now

---

1. If that feels familiar, it's because it is rewording Tolkien's famous quote: "It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to."
2. Or "let them eat cake", if you will."

Copyright: Ontotext New York | Sofia | Basel
Reach out to us: info@ontotext.com