ontotext

ONTOTEXT ANSWER:

Inference is one of the big concerns when performance is considered. Not only does it have a big impact on load times, it also functions as a black box. You choose your inference ruleset, you ingest some data, it does **something**, for a seemingly arbitrary amount of time, and then inferred statements come out.

However, with GraphDB using forward-chaining – i.e., emitting inferred statements at ingestion time – it's actually rather easy to debug and profile your inference. We've described those steps in a long form in our documentation, but here's the short of it.

The first, most crucial step is to decide what you need inference for. A lot of users just turn on inference because it's there, but they don't actually use any inferred statements. In fact, with RDFS-plus being the default ruleset, some users might not even know they have inference enabled. The best option is to ask the hard modeling questions first, and worry about optimizations later. After all, there's no point optimizing something you never plan to use!

If you need customization, our recommendation is to start simple and build up one rule at a time. Avoid recursion and duplications. An important rule for writing your ruleset is to keep the most specific calls first, to optimize execution time. GraphDB evaluates the rules with the least amount of unbound variables first, but then follows your ordering.

```
b <rdf:type> a                          # This gets executed third, as it has 2 variables.
b <example:underType> c                 # This would be executed last.
a <rdf:type> <example:Rule>             # This gets executed first, as it has only 1 variable.
a <rdf:hasSpecific> <example:Awe>       # This gets executed second.
-----------------------------------------
a <example:alsoAppliesTo> c
```

Order of execution for a sample rule

The second step, once you have settled on your optimal ruleset, and want to explore how to optimize it, is to create a custom ruleset. Even if you are using a default one. This sounds counterintuitive at first. The explanation is that GraphDB offers a debugging and profiling solution, but only on compiled rulesets. And the default rulesets are hardcoded. So, if you want to debug a default ruleset, you have to go to the */configs* directory where the raw

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

source for it is located and tell the database to use the raw ".pie" file.

The next step would be to start your GraphDB instance with the debugging flag:

-Denable-debug-rules=true

You provide the flag as a Java argument. Once that is done, you can start ingesting your data. Use the serial pipeline, debugging doesn't work with parallel inference.

The end result would be logs that contain a lot of information. We have developed a script that breaks it down into a human-manageable tsv..

In this tsv, each rule would have a few variants. Those variants have statistics about how much time they have fired, how many triples they inferred and how fast they got executed. That's the moment to start optimizing your problematic rules.

The topic of how to write an optimal ruleset, though, is a very large one, way beyond the scope of our answer here. It would be more suited to a research paper. We hope that what is shared here would be enough to get you on the right track towards optimization. Just remember to turn off debugging once you are done as debugging rules takes some time and you don't want it in production.

Did this help you solve your issue? Your opinion is important not only to us but also to your peers.

Help us get better: review GraphDB on Gartner

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