ontotext

ONTOTEXT ANSWER:

Query optimization can be quite problematic. Performance depends on a lot of factors and it's difficult to give a one-size-fits-all response. We'll try to give a few helpful pointers.

Inference vs querying. As a forward-chaining database, GraphDB materializes triples at load time. If you know you have a complex statement pattern that you will be querying often, why not precompute it with inference?

Extra indexes. Enable the context index if you are using a lot of named graphs. Add the predicate index if you have many different predicates or often use wildcated predicates.

Limiting data sizes. There is an inverse correlation between query speed and data size (including inferred triples). Querying can be compared to searching for a labeled item in a shop. It's a lot easier to find that item if the shop size is smaller.

Limiting response set sizes. It's easier to return 100 responses than it is to return 10,000. This is useful when doing quick exploratory queries. You can make them more representative by using the SAMPLE function.

Response batching. Return your data in batches. This is exactly what the workbench SPARQL UI does – returning the first 1,000 results, then the next and so on. This is controlled by two query parameters, limit and offset.

Binding set control. Each query has different bindings. For example: *?s ?p ?o* has three unbound variables and would return the whole database. *?s ex:predicate ?o* has two variables and so on. Keep the number of unbound variables as low as possible.

Redundant operations. Very often, you may have filters that never execute, or you may use the MINUS keyword instead of FILTER NOT EXISTS (the difference here is that MINUS computes all the triples, then excludes them, FILTER NOT EXISTS cuts off the moment a binding is observed). Reduce those redundant operations – and if you don't know that they are redundant, check the explain and measurement plans, as explained below.

Explain plan and onto:measure. The explain plan shows you how the engine would

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

ontotext

evaluate your query, with an estimated number of operations (complexity) per each step. If you use the named graph *FROM onto:measure*, the database would actually execute the query as well, and give you timings for each separate step of the execution process.

More resources. Especially if you are being capped by RAM or CPU, or seeing a lot of slow disk reads, simply giving more processing power to the database and moving it to a quick SSD may resolve your problem. You can explore this via the JMX interface and your own OS' monitoring tools.

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