German Medical Data Sciences: Shaping Change – Creative Solutions for Innovative Medicine
R. Röhrig et al. (Eds.)

doi:10.3233/SHTI190804

Implementation of a HL7-CQL Engine Using the Graph Database Neo4J

Georg FETTE[a,b,1], Mathias KASPAR[b], Leon LIMAN[a], Maximilian ERTL[b], Jonathan KREBS[a], Stefan STÖRK[b] and Frank PUPPE[a]
[a] *University of Würzburg, Chair of Computer Science 6*
[b] *University Hospital of Würzburg, Comprehensive Heart Failure Center*

Abstract. The Clinical Quality Language (CQL) is a useful tool for defining search requests for data stores containing FHIR data. Unfortunately, there are only few execution engines that are able to evaluate CQL queries. As FHIR data represents a graph structure, the authors pursue the approach of storing all data contained in a FHIR server in the graph database Neo4J and to translate CQL queries into Neo4J's query language Cypher. The query results returned by the graph database are retranslated into their FHIR representation and returned to the querying user. The approach has been positively tested on publicly available FHIR servers with a handcrafted set of example CQL queries.

Keywords. FHIR, query engine, graph database, Neo4J

1. Introduction

In recent years FHIR (Fast Healthcare Interoperability Resources) has become a widespread standard, not only for the exchange, but also for the storage of healthcare data. FHIR data storage servers are allowed to implement arbitrary technical storage layers, but the data access interface is clearly defined by the FHIR standard itself. The most common way to request data from a FHIR server is via FHIR search operations, which are part of the FHIR-REST-API. The FHIR search allows querying through resources of a server, filtered by parameters supplied in conjunction with the search operation. Although this search modality suffices for many simple request use cases, the expressiveness of FHIR search is limited in several aspects [1] (e.g. example 2 in Table 3 is not expressible in FHIR search). A more expressive approach to query FHIR data is possible by using the query language CQL (Clinical Quality Language; https://cql.hl7.org). CQL, like FHIR, is an HL7 standard. Its latest release (1 DSTU3) has been classified by maturity level 4 and is already in use in multiple prototype projects. Unfortunately, due to its novelty, execution engines for CQL are rare. To the best knowledge of the authors, only one CQL execution engine is publicly available (https://github.com/cqframework/clinical_quality_language).

CQL is a functional query language. A CQL script is defined as a so-called *library*, in which, besides the actual queries, also meta-information related to the queries can be

[1] Corresponding author: Georg Fette, Comprehensive Heart Failure Center (CHFC) Würzburg, University and University Hospital of Würzburg, Am Schwarzenberg 15, 97078 Würzburg, Germany; E-Mail: georg.fette@uni-wuerzburg.de

defined. The main functional elements in a CQL library are the *statements*. Each statement can be seen as an individual query. Usually, a statement defines the data sources that should be addressed by the query (e.g. *[Observation] B*), how the elements of the queried sources are constrained (e.g. *B.value = 'x'*), how the data sources relate to each other (e.g. *[Patient] A with [Observation] B such that B.subject = A*), and which elements of potential matches have to be returned as results (e.g. *Return Tuple {id:A.id, value:B.value}*). CQL provides a rich repertoire of operators (e.g. comparison, logical, arithmetic, list access, aggregation). These query definition paradigms are also essential to another field of computer science, namely graph databases and their query paradigms [2]. In order to investigate the applicability of graph databases and their query techniques, the authors implemented a CQL engine by using the graph database Neo4J as data sink and Neo4J's query language *Cypher* as the tool to express CQL queries.

2. Methods

In order to render a given FHIR server searchable using CQL the authors used the following approach. A Neo4J database server (Community Edition 3.5.3) was set up on a common desktop computer. All data from the FHIR server was exported to the file system into .csv files. The exported data was imported into the Neo4J database. All CQL queries to be executed were translated into Cypher and executed by the Neo4J server. The returned Neo4J data was retranslated into its FHIR representation and returned to the querying user. In the following chapters each of those steps will be explained in more detail and illustrated with the example resource from Figure 1.

```
{ "resourceType": "Patient", "id": "1438266",
  "birthDate": "2002-02-24", "name": [ { "given": [ "John" ] } ],
  "contact": [ { "name": [ { "given": [ "Mary" ] } ] } ] }
```

Figure 1. Example *Patient* resource encoded in JSON format.

In order to obtain the list of all profiles that are potentially hosted on the FHIR server, it is queried via FHIR search for all *StructureDefinition* resources. From each *StructureDefinition* the *snapshot* part is parsed for the initialization of all necessary .csv files into which the profile's resource data will get exported. To initialize the .csv files of FHIR's general-purpose complex types (*Duration*, *Range*, etc.), their *StructureDefinitions* were downloaded from the official FHIR website and processed in the same manner. Every profile is initialized with an individual .csv file. The columns in each .csv file store the primitive data type fields of the root layer of the respective profile. All *BackboneElement* fields within a profile's root layer are exported into individual .csv files, again with all primitive data type fields stored in the columns of their .csv files, and so on. The containments of *BackboneElements* within a parent profile or a different *BackboneElement* are stored in relationship .csv files that contain two columns for the IDs of the containers and their child structures. As *BackboneElements* in FHIR resources seldom contain an explicit ID, those IDs are generated as UIDs by the export process. All *Reference* resources are as well exported as relationship .csv files, containing the IDs of the referencing structures and those of their referenced resources. The actual resource data export requests all resources of

each profile via FHIR search without any further parametrization. If the FHIR server uses result paging, the URLs of all successive results are repeatedly followed until all resources have been processed. All fields of type *Reference* that reference the type *ANY* instead of a list of specific profiles were omitted, because untyped relations complicated the later import into Neo4J. Table 1 depicts the contents of the .csv files that are produced when processing the example from Figure 1.

Table 1. Contents of .csv files generated by the export process

Patient_nodes.csv		
:LABEL	**Patient:ID(Patient-ID)**	**birthDate**
Patient	1438266	2002-02-24
HumanName_nodes.csv		
:LABEL	**Patient:ID(HumandName-ID)**	**given:string[]**
HumanName	9aae43-eb86-4b01	John
HumanName	778e97-9e48-3ab2	Mary
Patient_contact_nodes.csv		
:LABEL	**Patient_contact:ID(Patient_contact-ID)**	
Patient_contact	e82707-1625-de8a	
Patient_to_HumanName_NAME_relations.csv		
:TYPE	**:START_ID(Patient-ID)**	**:END_ID(HumanName-ID)**
NAME	1438266	9aae43-eb86-4b01
Patient_to_Patient_contact_CONTACT_relations.csv		
:TYPE	**:START_ID(Patient-ID)**	**:END_ID(Contact-ID)**
CONTACT	1438266	e82707-1625-de8a
Patient_contact_to_HumanName_NAME_relations.csv		
:TYPE	**:START_ID(Patient_contact-ID)**	**:END_ID(HumanName-ID)**
NAME	e82707-1625-de8a	778e97-9e48-3ab2

All exported FHIR data is imported into the Neo4J database with a single call to the Neo4J import tool. All profile resources, all *BackboneElement* resources and all complex type resources are instantiated as individual Neo4J nodes. All containment relationships between profile or *BackboneElement* resources and contained *BackboneElement* or complex type resources as well as *Reference* resources are instantiated as Neo4J relationships. To supply all node and relationship .csv files to the import tool via the command line call, all parameters are written to a parameter file, which is given as the sole parameter for the import tool command call. The import tool is additionally parameterized to ignore relationships to resources that are not contained in the exported data. Figure 2 shows a screenshot of the Neo4J browser displaying the example from Figure 1 after import.

Figure 2. Screenshot of the Neo4J-browser displaying nodes after import of the example data from Figure 1.

The grammar of CQL is very similar to that of Cypher. A CQL query String is parsed and translated to a logical model [3, 4] containing all data elements, their paths and all constraints and operations applied on the data elements. The logical model is then used to generate the Cypher query. As CQL and Cypher differ in their supply of operators, the translator contains a list of one-to-one mappings between equivalent operators. For some operators (e.g. temporal operators) the authors implemented special mappings that perform more complex transformations. If a query contains

operators which are neither contained in the list of directly mappable operators nor in the list of operators with special implementations, the query was not translatable.

The CQL query Strings are parsed using a parser from the cql-to-elm project (https://github.com/cqframework/clinical_quality_language). The main parser from that project requires a data model file in order to perform type checking on the data elements contained in the CQL query. To avoid having to develop a generator for those model files, the authors decided to use the antlr4 parser from the CQL-project instead of the main parser.

For all Neo4J nodes returned by a Cypher query, the desired JSON String representing the original FHIR resource can be reconstructed from the property data of each returned node and the additional data of the nodes connected to that node via resource-internal relationships. The necessary relationships types that have to be resolved to reach all resource-internal nodes can be obtained from the *StructureDefinition* of the corresponding profile. As an additional caching mechanism, all Neo4J nodes representing profile resource roots receive an additional String property containing the resource's original JSON-String. This can be used instead of having to reconstruct the resource using all necessary node properties and having to resolve relationships.

3. Results

Using the approach described above, the authors indexed several publicly available FHIR server endpoints. Table 2 shows some information about the tested FHIR servers and their indexing process.

Table 2. Test FHIR servers that were indexed using the presented approach.

FHIR Server base URL	Export time / minutes	import time / seconds	Neo4J db size / MB	Node count / 1 000	Relation count / 1 000	Property count / 1 000
http://hapi.fhir.org/baseDstu3	50	15	257	1 490	1 292	3 145
http://vonk.fire.ly	12	10	69	358	390	818
http://test.fhir.org/r3	4	3	9	41	38	134

In the current state of the project the translator is capable of translating CQL queries with the following attributes: The CQL query may contain an arbitrary amount of arbitrary data sources. All data model element paths in the query can have arbitrary length. Data model elements can be constrained with arbitrary logical comparators. Multiple sources can be concatenated and their relationships can be constrained. Operators can be translated, when they exist with the same interface and the same semantics in both query languages or when they can be substituted by a combination of Cypher operators. During the implementation of the translator the following Cypher operators were detected missing: Cypher does not support CQL's arithmetic operators *log* (not to be confused with Cypher's *log* operator), *truncate*, *ceiling* and *div*. Cypher does not support CQL's String operators *Combine*, *ReplaceMatches* and *SplitOn*

Matches. Cypher does not allow an equivalent syntactical representation of CQL's list operators *first*/*last* parametrized with a *sort by*. The translation of *first/last* can be substituted with Cypher's *max/min* operators, if the path, which is accessed from the result of the list operator, is the same that has been used to sort the list (see example 3 in Table 3). The pattern could as well be translated with an additional nested Cypher query, but this approach has not yet been pursued.

The translation process for CQL queries to be executed on the Neo4J indices was tested with a list of handcrafted CQL statements, from which some are listed in Table 3. The automatic translation of each CQL statement into Cypher had to be equal to a predefined desired Cypher String.

Table 3. Example CQL queries and their corresponding Cypher queries.

No.	CQL query	Cypher query
1	[Encounter] A where A.length > 120 days	match (A:Encounter)-[:LENGTH]->(B:Duration) where B.value > 120 and B.unit = 'days' return A
2	[Observation] A with [Medication Administration] B such that A.subject = B.subject and A.valueString = 'blue' and B.effectiveDateTime > @2014	match (A:Observation)-[:SUBJECT]->(B), (C: MedicationAdministration)-[:SUBJECT]->(D) where B = D and A.valueString = 'blue' and C.effective DateTime > datetime('2014') return A
3	[Patient] A with [EpisodeOfCare] B such that B.patient = A and Last(B O sort by period.end).period.end < @2014	match (A:Patient), (B:EpisodeOfCare)-[:PATIENT]-> (C), (B)-[:PERIOD]->(D) with max(D.end) as E, A, B, C where C = A and E < datetime ('2014') return A

4. Discussion

FHIR data represents a graph structure, so what could be more appropriate than storing and querying FHIR data using a graph database? Neo4J is one of the most popular graph databases and it is a mature solution, in existence for more than 12 years and still actively developed. The reliability of the underlying Neo4J query engine has been proven in a multitude of projects.

For the evaluation of a CQL query the implementation guide of the CQL documentation proposes to evaluate all sources of a query and subsequently to filter the preliminary results by all constraints contained in the query. In the Neo4J database the resolution of sources and potentially additional constraints can be evaluated jointly, thus possibly enhancing query performance. No query speed evaluations have yet been performed, but it would be interesting to compare the query speed of the Neo4J approach with alternative CQL engine implementations. The current version of the engine implementation represents a snapshot of work in progress, covering only a portion of the full extent of CQL. There are still many CQL expressions that cannot yet be translated. It has to be analyzed up to which extend and depth this can be performed for arbitrary CQL statements using the language elements provided by Cypher. Some shortcomings of Cypher have already been addressed above.

The approach of translating a query into another query under preservation of the query's semantics is related to the research field of query rewriting [5]. In the presented approach the authors followed a more engineering oriented approach compared to the existing literature tackling query rewriting. Therefore, the inclusion of ideas from the field of query rewriting could be fruitful.

Currently, the export and import process has to be a full indexation of the whole data content of a FHIR server. Because full indexation is time consuming, it would be

desireable to have a delta export and import, processing selectively only that part of the data, with timestamps after the last update run [6]. Such a delta update could be run in fixed time intervals in order to keep the Neo4J database synchronized with its source FHIR server. A problem with the current export approach is that *delete* operations on the FHIR server could be undetectable, because deleted resources are unrequestable via FHIR search. The delete part of the delta could possibly be infered via requests to the *_history* part of the FHIR search API.

The export process, which is currently implemented using FHIR search, could be changed to a different FHIR access mode called *Bulk Data API*. This API was specified at the beginning of 2018 and is already implemented on several publicly available FHIR servers. Bulk exports might be a faster method to export FHIR resource data than FHIR search. However, not every FHIR server must necessarily implement the Bulk Data API; if absent, the regular FHIR search has to be used.

Neo4J contains a *Duration* data type, which is not yet used during indexing. The translation of example 1 from Table 3, in which the unit of the duration is currently explicitly compared via a String compare, could be processed using Cypher language elements that are more similar to the duration representation used in CQL.

5. Conclusion

An approach was presented to implement a CQL query engine by creating a Neo4J graph database index on top of an existing FHIR server and letting all queries be performed by the Neo4J server's Cypher query engine, which can be used by translating all CQL queries into Cypher queries. The approach is easily applicable on a FHIR server without modifications of existing structures. For various handcrafted test CQL queries the approach proved well feasible.

An implementation of the presented approach in Java is available at https://gitlab2.informatik.uni-wuerzburg.de/gef18bg/FHIR_Neo4J_Indexer.

Acknowledgements

This research was funded by the German Federal Ministry of Education and Research (Comprehensive Heart Failure Center Würzburg, grants #01EO1004 and #01EO1504).

References

[1] Zautke A, Evaluation of Methods for Querying, Filtering and Aggregating Complex Health Care Data with regard to their Applicability in FHIR, *Master Thesis*, (2018).
[2] Robinson I, Webber J, Eifrem E, *Graph Databases*, O'Reilly Media Inc., Sebastopol USA, (2013).
[3] Fette G, Kaspar M, Liman L, et al. Query Translation Between openEHR and i2b2. Stud Health Technol Inform.;258, 16-20 (2019).
[4] Fette G, Kaspar M, Liman L, et al, Query Translation Between AQL and CQL, To appear in: MedInfo Conference (2019).
[5] Calvanese D, De Giacomo G, Lenzerini M, et al, What is query rewriting? In Proceedings of the 7th International Workshop on Knowledge, Representation meets Databases, 17-27 (2000).
[6] Lindsay BG, Haas LM, Mohan C, Pirahesh H, Wilms PF, A snapshot differential refresh algorithm. In: SIGMOD Conference, 53–60 (1986).