Informatics for Health: Connected Citizen-Led Wellness and Population Health
R. Randell et al. (Eds.)

doi:10.3233/978-1-61499-753-5-406

Querying Archetype-Based Electronic Health Records Using Hadoop and Dewey Encoding of openEHR Models

Erik SUNDVALL[a,b,1], Fang WEI-KLEINER[a], Sergio M FREIRE[c], and Patrick LAMBRIX[a,d]
[a]*Linköping University, Linköping, Sweden* [b]*Region Östergötland, Linköping, Sweden*
[c]*Universidade do Estado do Rio de Janeiro, Rio de Janeiro, Brazil*
[d]*Swedish e-Science Research Centre, Sweden*

Abstract. Archetype-based Electronic Health Record (EHR) systems using generic reference models from e.g. openEHR, ISO 13606 or CIMI should be easy to update and reconfigure with new types (or versions) of data models or entries, ideally with very limited programming or manual database tweaking. Exploratory research (e.g. epidemiology) leading to ad-hoc querying on a population-wide scale can be a challenge in such environments. This publication describes implementation and test of an archetype-aware Dewey encoding optimization that can be used to produce such systems in environments supporting relational operations, e.g. RDBMs and distributed map-reduce frameworks like Hadoop. Initial testing was done using a nine-node 2.2 GHz quad-core Hadoop cluster querying a dataset consisting of targeted extracts from 4+ million real patient EHRs, query results with sub-minute response time were obtained.

Keywords. Medical Record Systems, Computerized; Database Management Systems, Dewey encoding, Archetypes, openEHR, Hadoop, Epidemiology, XML

1. Introduction

The adoption of standardized “archetype”-based Electronic Health Record (EHR) systems is increasing globally. Such systems use a fixed *reference model* (from e.g. openEHR, ISO 13606 or CIMI) that provides basic building blocks that are then assembled and constrained (primarily by clinicians) into clinically relevant structures using modeling layers consisting of *archetypes* and *templates* [1]. This partly resembles how XML building blocks can be assembled and constrained by layers of schemas. “Archetyped” instance data, i.e. conforming to archetypes, templates and the corresponding reference model (RM), often form deep tree structures where path-based querying is useful for both single-patient and epidemiological multi-patient use cases.

Existing deep tree storage and retrieval mechanisms (for example XML- and JSON-databases) can be reused for archetype-based systems [2]. There are open source XML-database solutions fully capable of handling single-patient use-cases for databases with millions of archetype-based records [3]. However, for good performance for large population epidemiological queries, other approaches or optimizations are needed [4]:

[1] Corresponding Author: erik.sundvall@liu.se. IMT, Linköping University, 581 85 Linköping, Sweden

- If the queries are well known beforehand and regularly re-run, like for recurring statistical needs, then common solutions are a) to have database administrators (DBAs) create query-specific index optimizations or b) to export relevant parts to a data warehouse (OLAP) system.
- In exploratory epidemiological research, on the other hand, where query results often lead to new questions, the delay caused by waiting for a DBA or other constrained (often human) resources can become a major bottleneck. Thus explorations of automated optimizations also handling ad-hoc population queries have been published [4]. In this publication we describe *Dewey encoding* as another method to add to the arsenal of solutions for these ad-hoc query use cases.

Storage of archetype-based data can be implemented using various database approaches [2][3][4][5] but are often queried using the implementation-neutral AQL (Archetype Query Language) [6] that can be translated into the database's ”native” query language.
Many kinds of tree-shaped data can be *Dewey encoded* and stored in many kinds of persistence solutions. This paper contributes an outline of how Dewey indexing can be customized by including *archetype_node_id* in the index paths in order to enhance performance when applied to openEHR models and associated *AQL* queries; non-enhanced XML Dewey coding would instead store *archetype_node_id* as attribute+value. In our implementation we used Hadoop as a scalable persistence and query execution environment. Adapting the approach for other persistence platforms and non-XML-formats should be rather straightforward.

2. Background

Based on the Dewey Decimal Classification [7], which was originally developed for general knowledge classification, Dewey order encoding is among the popular methods for encoding XML documents [8]. With Dewey order, each node in an XML tree is labeled with a vector (for example the string 1.8.3.1) representing the path from the root to the node following the rules: (1) The root is labeled as an empty string; and (2) For a non-root element *u, label(u)=label(v).x*, where *u* is the *x*th child of *v*.
With the Dewey order encoding, each node in the XML tree is labeled with a distinct string. This enables Axis operations in an XPath-style query such as ::child, ::descendant and ::sibling to be simply formulated with common string operations. Based on the Dewey order coding, all the distinct paths of the underlying XML document can then be enumerated and to each path expression, the set of the corresponding nodes (in the form of Dewey encoding) is referred. The steps above are in fact the essential operations used in the so-called XML document shredding, with the ultimate goal of storing and querying the XML data in Relational Database Systems.

3. Method

Since the collection of all archetype-based data instances in an EHR system is a giant logical tree with paths, we follow the general routine of Dewey order encoding [8] to label all the nodes in the tree. One index table (*pathindex* in Figure 1) assigns a consecutive path id number to each unique path found in the data, thus if the path has already been seen in a previous document (EHR extract) then the path is not added to pathindex. As an example of this, consider Table 1 that shows that there were 2820 different paths in the EHR extracts of 4.2 million patients (labeled sus4200k). The Dewey id of the document node containing a path is then appended to a "TP"-table (Figure 1) that has a name based on the *Path id* number corresponding to the path in the pathindex table. If it is a branch node then the corresponding table is of type Branch, if there instead is a leaf/data/text value at the node it is of type Leaf, see Figure 1. This design using a large number of tables/files suits the distributed Hadoop implementation when data and computational load is to be distributed over several servers.

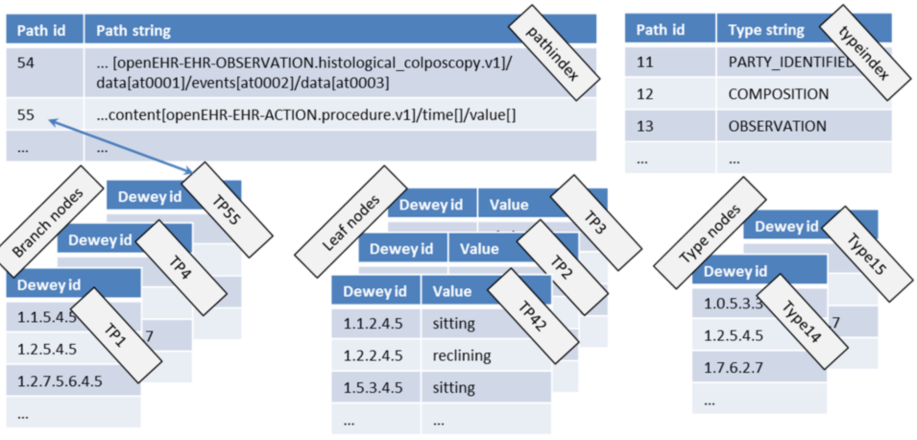

Figure 1. Simplified table examples. The number of TP-tables corresponds to the number of entries in the pathindex table that grows when new kinds of clinical information (using new archetypes etc) is added to the system. The number of rows within the TP-tables increases with the number of patients and notes per patient.

archetype_node_id: In an archetype-based tree structure, many node instances have an attribute named archetype_node_id that via archetypes identify the clinical meaning of the node [1]. In an AQL query [6], archetypes play important roles by occurring in the form of archetype predicates such as e.g. [openEHR-EHR-COMPOSITION.encounter.v1], and node predicates such as e.g. items[at0001]. Therefore, we make the archetype_node_id a first class citizen by including archetype_node_ids in the enumerated paths. An example path expression is: `...data[at0001]/events[at0002]/data[at0003]/items[at0004]/value[]...`

This is in contrast to the conventional XML transformation method [8] of storing the archetype-id and values in corresponding attributes in a table, thus a considerable amount of join operations on the attribute tables and the path tables can be avoided.

Reference Model types. RM types such as COMPOSITION, OBSERVATION occur in the FROM clause of an AQL query as class expressions to scope the data source for the query. We collect all the RM types and for each type *t*, the set of Dewey_IDs whose type is tagged *t* is stored in a table (see "Type nodes" in Figure 1).

AQL query processing. A formal description of the translation from AQL to SQL is beyond the scope of this paper. In the following we outline general rules with the help of a running query example (taken from a Brazilian epidemiology context described in [3] and [4]) which returns the records in a certain date interval where a certain examination is missing (indicated by the presence of a null-flavour value):

```
SELECT e/ehr_id/value as ehr_id FROM Ehr e
CONTAINS COMPOSITION c [openEHR-EHR-COMPOSITION.citologic_exam_form.v1]
CONTAINS OBSERVATION obs [openEHR-EHR-OBSERVATION.siscolo_anamnesis.v1]
WHERE EXISTS
obs/data[at0001]/events[at0002]/data[at0003]/items[at0004]/value
AND EXISTS
obs/data[at0001]/events[at0002]/data[at0003]/items[at0022]/null_flavour
AND (c/context/start_time/value >= $beginTime AND
c/context/start_time/value < $endTime)
```

The translation takes the following steps:

1. Identify paths. All paths can be extracted from the SELECT and WHERE clauses. Moreover, we retrieve the archetype predicates from the FROM clause and combine them together to produce the desired path expressions. For the example query we generate four paths ending with these substrings (suffixes):

P1	e/ehr_id/value
P2	[openEHR-EHR-OBSERVATION.siscolo_anamnesis.v1] /data[at0001]/events[at0002]/data[at0003]/items[at0004]/value
P3	[openEHR-EHR-OBSERVATION.siscolo_anamnesis.v1] /data[at0001]/events[at0002]/data[at0003]/items[at0022]/null_flavour
P4	[openEHR-EHR-COMPOSITION.citologic_exam_form.v1] /context/start_time/value

Note that if the retrieved path is not unique, we will retrieve a set of paths. To simplify the presentation, we assume all above paths are unique.

2. Retrieve tables. For each path from the last step, we retrieve the tables T_{P1}, T_{P2}, T_{P3} and T_{P4} respectively. Note that T_{P1} and T_{P4} contain Dewey_ID and value attributes (leaf tables), and T_{P2} and T_{P3} contain only Dewey_IDs (branch tables).

3. Filters. The filter (`value >= $beginTime AND value < $endTime`) is applied to T_{P4}.

4. Join operation. We conduct a join operation on T_{P2} and T_{P3} with the Dewey_ID of the variable `obs` as the join attribute. Note that the Dewey_ID of `obs` can be conveniently obtained by trimming a suffix of certain length on the Dewey_ID of T_{P2} and T_{P3} respectively. We name the result table as T_{P23}.

5. Containment checking. Analogously to `obs`, we can obtain the Dewey_ID of `e` and `c` from T_{P1} and T_{P4} respectively. Then the containment checking is conducted by the join operation on T_{P1}, T_{P23} and T_{P4} with the join condition that `e` is the ancestor of `c` and `c` is the ancestor of `obs`. We name the result table as T_{P1234}.

6. Type checking. We check whether `c` is of type `COMPOSITION` and `obs` is of the type `OBSERVATION` by joining T_{P1234} with the type table $T_{COMPOSITION}$ and $T_{OBSERVATION}$.

4. Results and Discussion

The data sets used for tests are sus42k, sus420k and sus4200k, which contain information about 42,428, 424,270 and 4,242,500 patients, respectively. They have been described and used for other performance tests in [4]. The data is shredded into the branch and leaf type tables. The characteristics of the datasets are given in the following table.

Table 1. Characteristics of the datasets used in this study. Size is the total size of all the shredded tables.

Shredded dataset	Size	# Paths	#Branch tables	#Leaf tables	#Type tables	Average table size
sus42k	4.5 GB	2625	1819	806	24	1.7 MB
sus420k	45 GB	2784	1929	855	24	16 MB
sus4200k	450 GB	2820	1953	867	24	158 MB

The experiments were conducted in a nine-node cluster Hadoop server; each node with quad 2.20 GHz AMD Opteron(TM) 6274 Processors running Debian GNU/Linux 6.0.10 with 12 GB RAM and 1 TB hard disk. The Hadoop block size was 64 MB. The query, also used in [4], was manually translated and coded in Java and then executed using Hadoop map-reduce processing on the datasets described in Tables 1 and 2.

Table 2. Datasets, table sizes and results from query performance testing on a Hadoop cluster. The last column shows the running time values of the query processing. For each dataset, the query was run 10 times and an average running time value was calculated.

Table size:	T_{P1}	T_{P2}	T_{P3}	T_{P4}	Total	Running time
sus42k	2.1 MB	0.7 MB	0.1 MB	1.6 MB	4.5 MB	21 s
sus420k	22 MB	6 MB	1 MB	13 MB	42 MB	29 s
sus4200k	218 MB	60 MB	7 MB	130 MB	415 MB	59 s

The results of this initial test indicate that archetype-aware Dewey indexing can be a useful tool to support ad-hoc querying in large datasets on systems with relational capabilities (join and containment operations etc.) The indexing is related to *inverted indexes* (like Apache Lucene) that have been applied in other openEHR settings [2].

Comparisons to previous work: Performance tests [3][4][5] of different archetype-based systems are hard to compare with each other due to different data and hardware. In previous studies [3][4], using this data, Couchbase gave really fast results for recurring queries, but required indexing time for every new (unseen) query; for really small datasets BaseX was faster than this more scalable Dewey+Hadoop approach.

Future work: a) Benchmarking studies should compare several solutions on the same hardware setup using the same data and queries.

b) The single query translated manually and tested is just an initial feasibility test, but we were encouraged by other researchers to publish it so that other teams can explore it. In any realistic production system the AQL to Hadoop-querying translation should be automated instead of hand-coded in order to support the exploratory epidemiological research use case described in the introduction. Automating such translation is also a reasonable prerequisite to ease preparation of further performance tests with multiple different AQL queries. Other automated AQL translations have been done earlier [3], and AQL grammars are available [6].

References

[1] openEHR architecture overview, Release-1.0.3, 2015, http://www.openehr.org/releases/BASE/Release-1.0.3/architecture_overview.html

[2] Frade S, Freire S, Sundvall E, Patriarca-Almeida J, Cruz-Correia R. Survey of openEHR storage implementations. *26th IEEE International Symposium on Computer-Based Medical Systems* 2013 June 20–22; Porto, Portugal. p. 303–307.

[3] Freire S, Sundvall E, Karlsson D, Lambrix P. Performance of XML Databases for Epidemiological Queries in Archetype-Based EHRs. *Scandinavian Conference on Health Informatics*. 2012 October 2–3; Linköping, Sweden. Linköping Electronic Conference Proceedings 70, p. 51–57.

[4] Freire S, Teodoro D, Wei-Kleiner F, Sundvall E, Karlsson D, Lambrix P. Comparing the Performance of NoSQL Approaches for Managing Archetype-Based Electronic Health Record Data. *PLoS ONE* 11(3): e0150069, 2016.

[5] Wang L, Min L, Wang R, Lu X, Duan H. Archetype relational mapping - a practical openEHR persistence solution. *BMC Medical Informatics and Decision Making* 15:88, 2015.

[6] Archetype Query Language (AQL), http://openehr.org/releases/QUERY/latest/docs/AQL/AQL.html

[7] Dewey Decimal Classification, https://en.wikipedia.org/wiki/Dewey_Decimal_Classification

[8] Tatarinov I, Viglas SD, Beyer K, Shanmugasundaram J, Shekita E, and Zhang C. Storing and querying ordered XML using a relational database system. *ACM SIGMOD International Conference on Management of Data.* 2002 June 2-6; New York, NY, USA. ACM, p. 204-215.