Experimental performance evaluation of different data models for a reflection software architecture over NoSQL persistence layers

Sara Fioravanti Simone Mattolini Fulvio Patara Enrico Vicario

Department of Information Engineering
University of Florence, Italy

Delft, 16th March 2016
7th ACM/SPEC International Conference on Performance Engineering

- comparative experimental performance results
  - configurable pattern based domain logic (reflection and composite)
  - on different persistence layers (JPA/MySQL, MongoDB and Neo4j)

Outline

A growing variety of information

- Drivers Over the last decade, a growing digital universe of . . .
  - *human-sourced information*: unstructured/semi-structured data by human interactions
  - *process-mediated data*: structured data by traditional ICT systems
  - *machine-generated data*: (real-time) well-structured data by smart sensors and computer systems
- Trends Adoption of innovative forms of *data modeling*
- Goals
  - Enabling enhanced insight
  - Supporting decision making and process automation

Requirements

Challenges in terms of data modeling and information processing, closely related to the complex nature of data (five "Vs"):

- Volume
- Variety
- Variability
- Velocity
- Veracity

Support for variation is the key to sustainable architectures for long-lived applications

- in domains characterized by high variability and volatility
- where software requirements are not stable and evolve over time
  - e.g. healthcare domain

Requirements - 2

Variations can occur at any time and can affect structural and functional aspects of a system

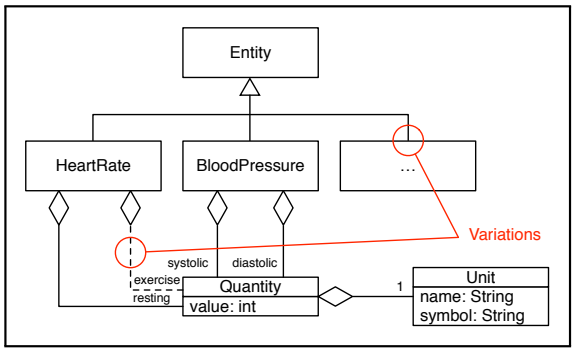
- Hard to forecast future requirements and their impact on the system

Aim: Adaptability and configurability as primary requirements

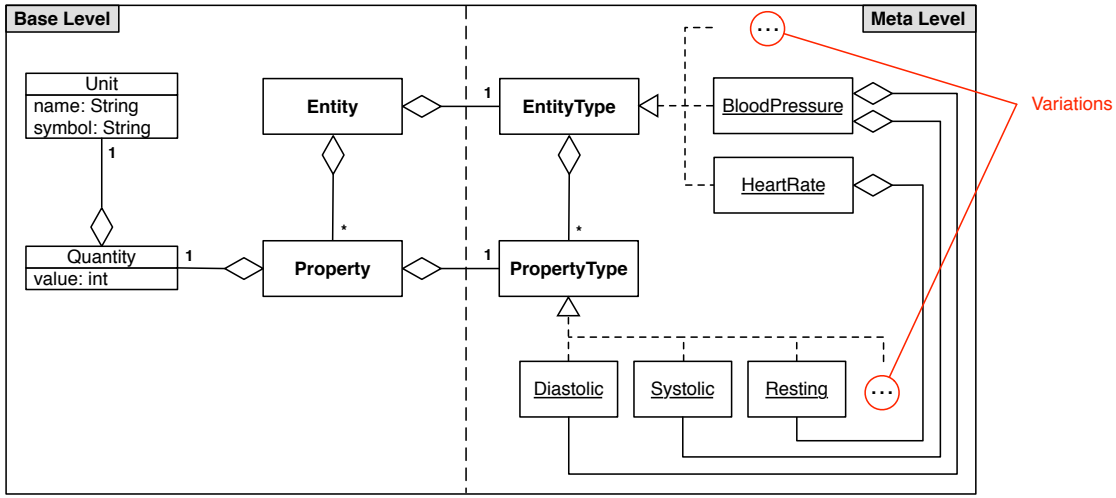
- Moving from *static single-level* systems
  - Unrealistic except in the case of expected low rate of change
  - Continuous cycle of re-coding, re-building, re-testing, and re-deploying after each new and neglected requirement
- . . . to *dynamic multi-level* meta-modeling architectures
  - Design systems easily adaptable to changing requirements on-demand represents a powerful alternative to consolidated but unsuccessful static solutions

The healthcare context

EHR system: is a software system for recording, retrieving, and manipulating repositories of retrospective, concurrent, and prospective information regarding the health status of a subject-of-care.



static single-level system



dynamic multi-level meta-modeling architecture

Reflection & Hierarchical composite

Two-level pattern-oriented meta-modeling architecture with a high degree of abstraction

- Reflection architectural pattern
- Observations & Measurements analysis pattern [1]
  - Real implementation of meta-modeling principles applied to the clinical process

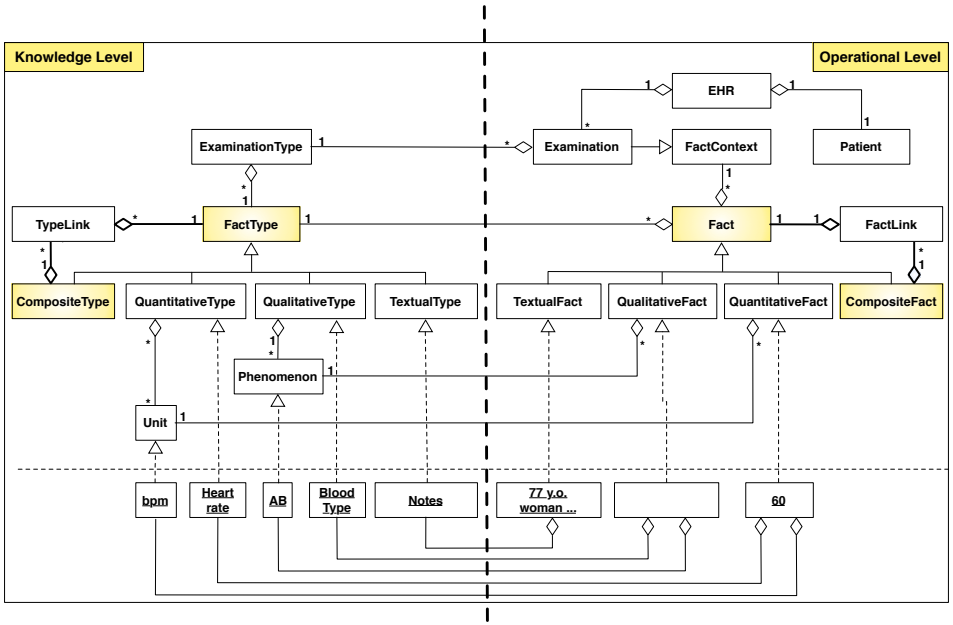
Combined with

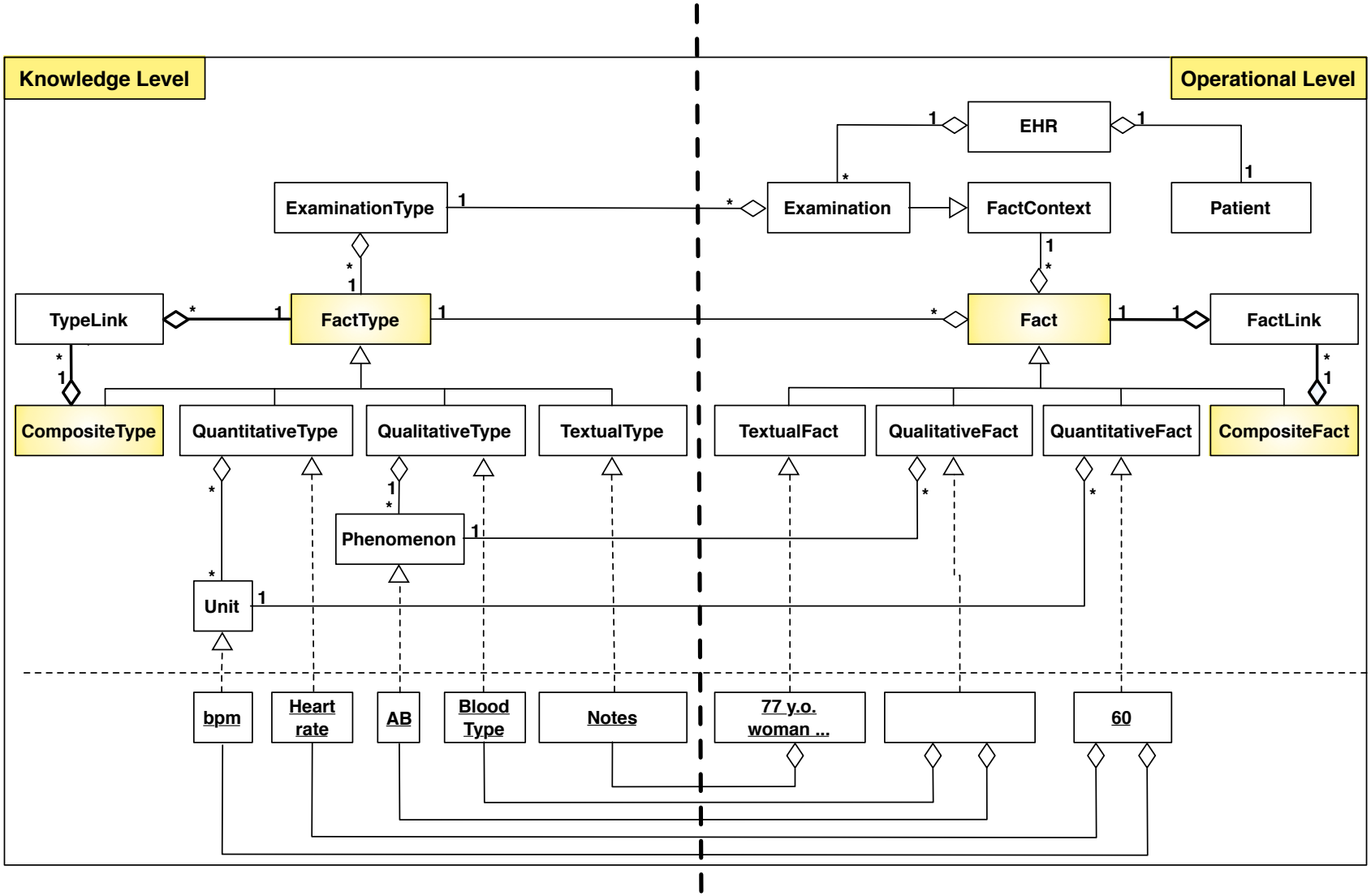
- Composite pattern to repeat aggregation of basic observations and measurements as a collection of references to other of them.

[1] Fowler, M. (1996). "Analysis patterns: reusable objects models". Addison-Wesley Longman Publishing Co., Inc.

Reflection & Composite on EHR System

- Separation between medical knowledge, at knowledge level
  - Semantics of medical phenomena
- . . . and clinical data, at operational level
  - Values assumed by medical phenomena
- composition to create tree structures




Knowledge Level
Operational Level
EHR
ExaminationType
Examination
FactContext
Patient
TypeLink
FactType
Fact
FactLink
CompositeType
QuantitativeType
QualitativeType
TextualType
TextualFact
QualitativeFact
QuantitativeFact
CompositeFact
Phenomenon
Unit
bpm
Heart rate
AB
Blood Type
Notes
77 y.o. woman ...
60

Benefits

- Pros
  - Open to modification, extension, and evolution over time
  - Improved overall system maintainability
  - High degree of flexibility and adaptability to different contexts
  - Run-time configuration and inversion of responsibility
  - Pattern-based solutions
    - Reflection pattern [2]
    - Dynamic Object Model pattern [3]

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. "Pattern-Oriented Software Architecture, Volume 1: A System of Patterns". Wiley, 1996.

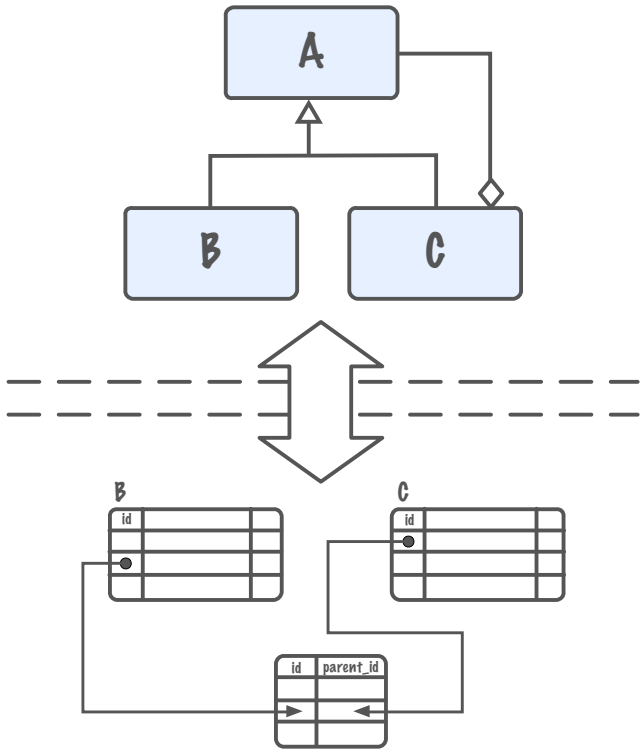
[3] D. Richle, M. Tilman, and R. Johnson. "Dynamic object model". Pattern Languages of Program Design, 5:3-24, 2000.

Limits

- Cons
  - Increased architectural complexity:
    - more indirect, less intuitive, harder to code, test, and maintain
  - Performance inefficiencies
    - design by patterns does not account for performance as first-class requirement, and naturally incurs in well-known performance anti-patterns
    - extra processing due to an increased number of objects (and relationships) needed for describing the whole domain

Performance from the interaction with persistence layer

- mismatch with the relational tier
- long time required for key persistence operations
- requires ad-hoc optimizations in the design of the relational database, pertaining to the choice of a particular representation for class inheritance, the use of auxiliary tables to store additional information, and the smart use of data fetching.



Hibernate and SQL solution

Benefits:

- reconciles mismatch between objects and relational data
- minimal boilerplate code, thanks to annotations
- provides full integration with the Java application stack

Drawbacks:

- JPA increases the degree of indirection with loss of design control
- structure organized in tables is not really compliant with the architectural model considered.

NoSQL Technologies

Various promised advantages:

- reduced access time through the clustering of similar data
- increased adaptability to the variety and variability of data over time through the use of a schemaless structure
- fit better domain model.

e.g. Neo4j, MongoDB.

Methodology and intent

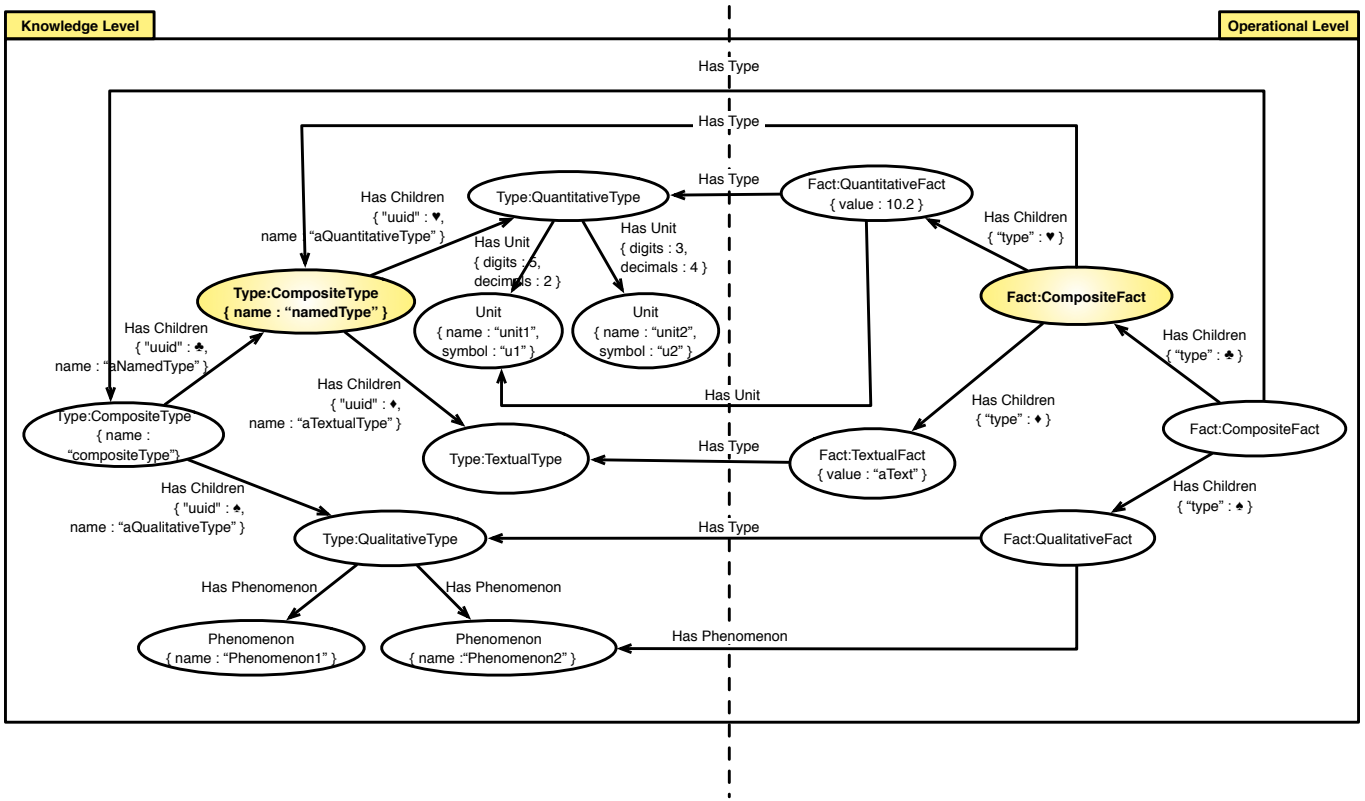
- Definition of new models that contains the same informations of the original one
- Persist on different choices of databases
- Measure performances
  - referred to a crucial application use case
  - applied on real data from the practice of use of the EHR system Empedocle, used at major hospital in Tuscany Region and on synthetic data generated stressing the most relevant dimensions of complexity.

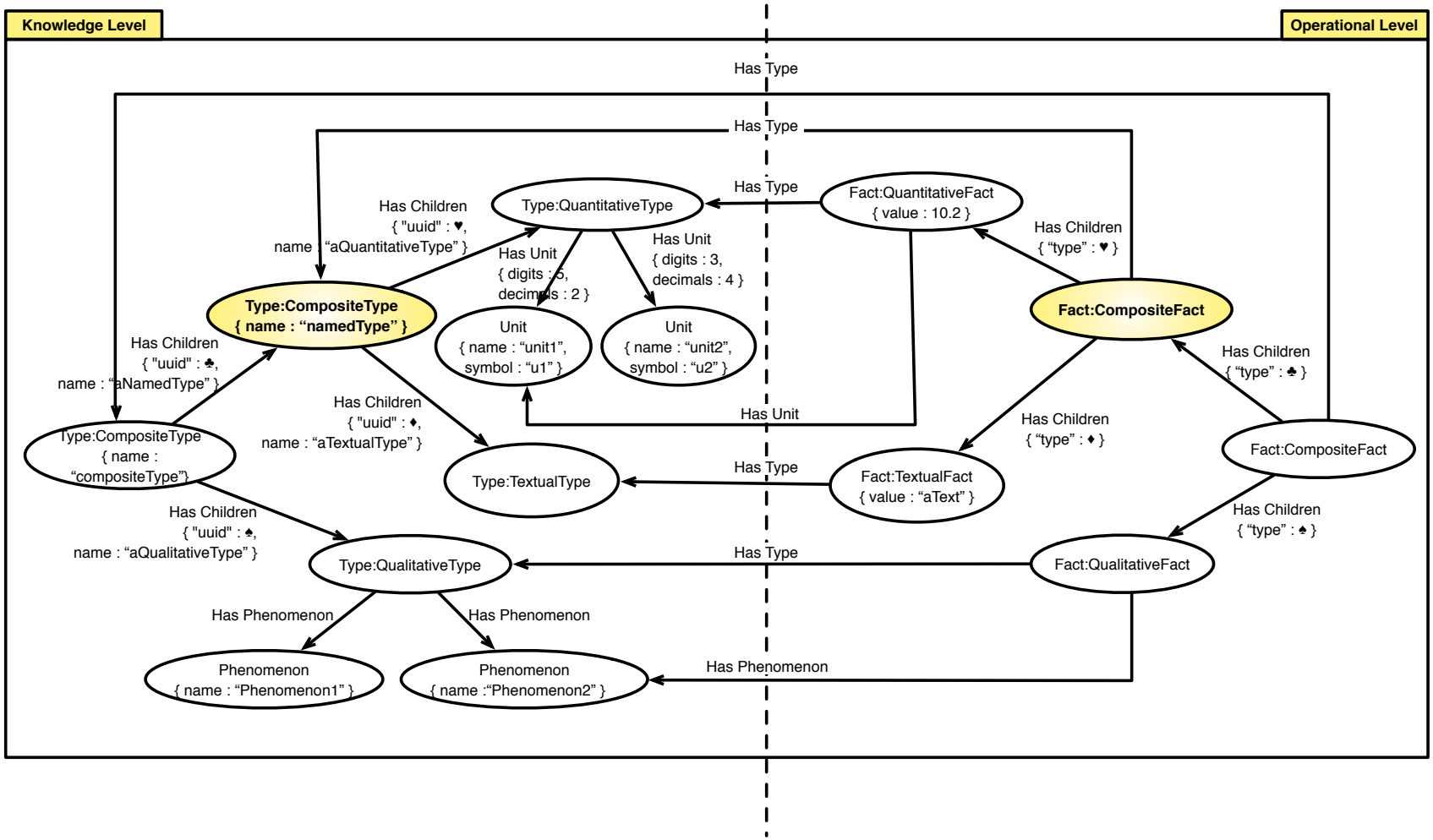
Methodological issues

- design efficient models
  - the absence of a fixed schema provides multiple options concerning the definition of the database structure
- demonstrate equivalence between data models
  - query equivalence of two models is defined as the possibility to extract the same information from both models through query operations.

Neo4j Model

- identify the node structure that forms the model;
- defining the relationships between nodes;
- defining the properties that characterize nodes and relationships
- labeling with the appropriate qualifiers.




Knowledge Level
Operational Level
Has Type
Has Type
Has Type
Has Type
Has Children
{ "uuid" : ♥,
name : "aQuantitativeType" }
Type:QuantitativeType
Fact:QuantitativeFact
{ value : 10.2 }
Has Children
{ "type" : ♥ }
Has Unit
{ digits : 5,
decimals : 2 }
Has Unit
{ digits : 3,
decimals : 4 }
Type:CompositeType
{ name : "namedType" }
Fact:CompositeFact
Unit
{ name : "unit1",
symbol : "u1" }
Unit
{ name : "unit2",
symbol : "u2" }
Has Children
{ "uuid" : ♣,
name : "aNamedType" }
Has Children
{ "type" : ♣ }
Has Children
{ "uuid" : ♦,
name : "aTextualType" }
Has Unit
Has Children
{ "type" : ♦ }
Type:CompositeType
{ name :
"compositeType"}
Type:TextualType
Has Type
Fact:TextualFact
{ value : "aText" }
Fact:CompositeFact
Has Children
{ "uuid" : ♠,
name : "aQualitativeType" }
Has Children
{ "type" : ♠ }
Type:QualitativeType
Has Type
Fact:QualitativeFact
Has Phenomenon
Has Phenomenon
Phenomenon
{ name : "Phenomenon1" }
Phenomenon
{ name :"Phenomenon2" }
Has Phenomenon

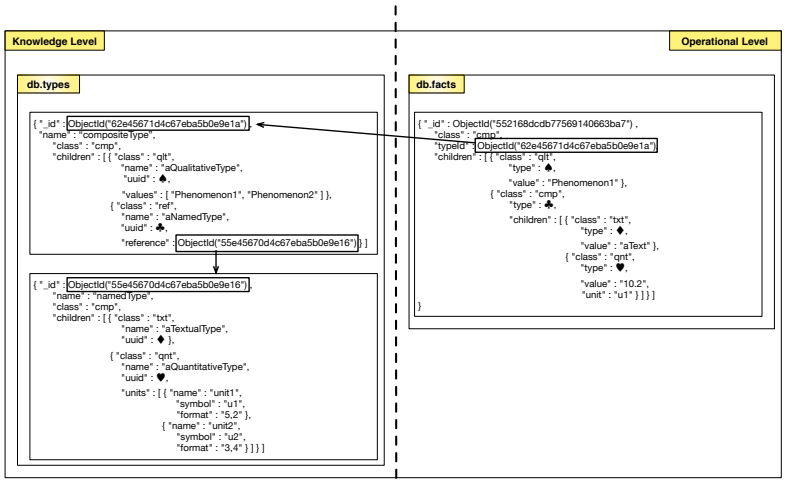
MongoDB Model

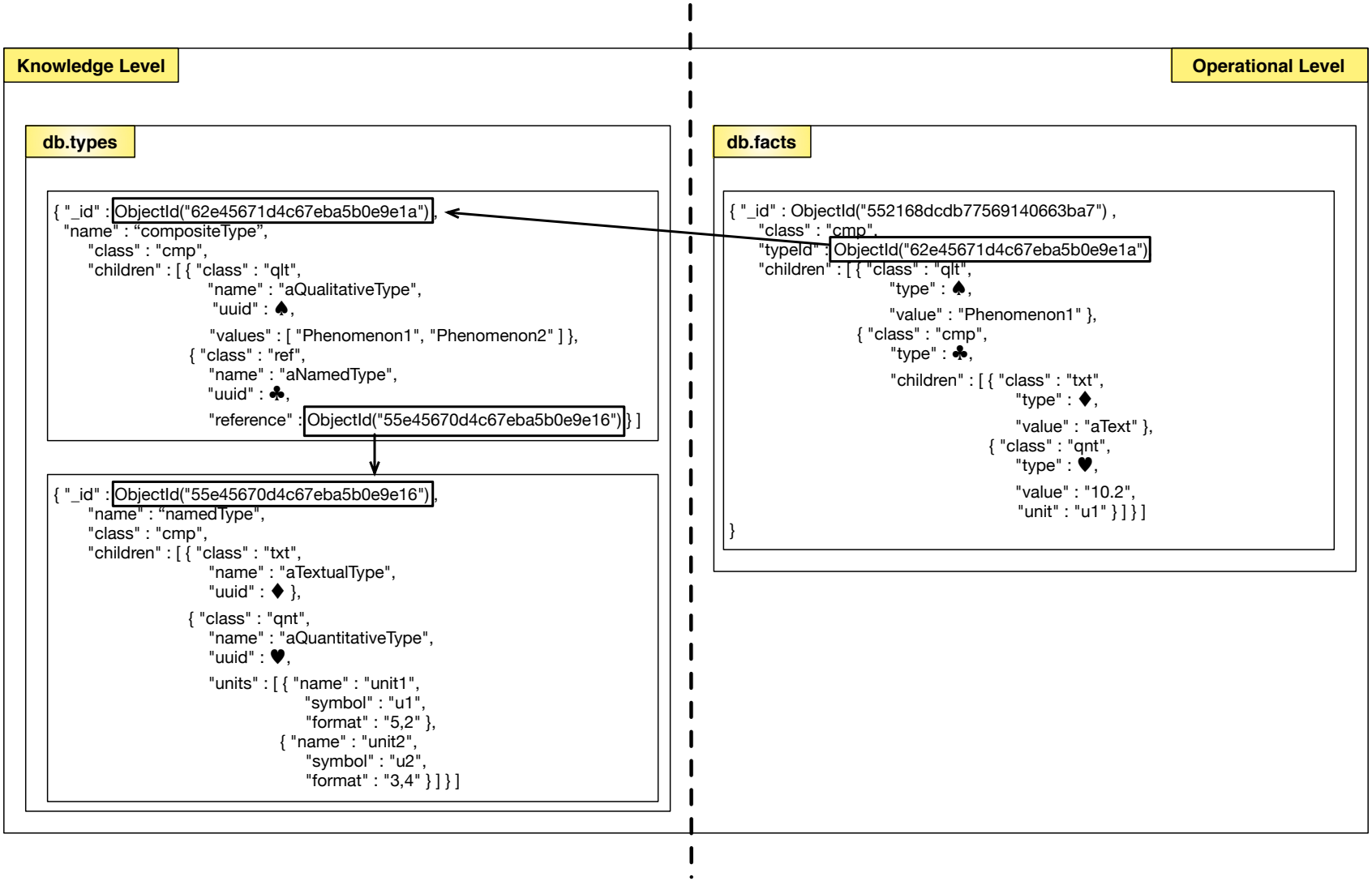
Solution uses documents embedding and references, mixing together.

- *named* FactType instances are persisted as documents, and are referenced by other documents using their ObjectId
- *anonymous* FactType instances are persisted as embedded documents inside the CompositeType in which they are defined.

For completely retrieving a Fact and its FactType, it is necessary to:

- query for the Fact;
- query for the related FactType using the ObjectId;
- link together the retrieved Fact and FactType instances;




Knowledge Level
Operational Level
db.types
db.facts
{ "_id" : ObjectId("62e45671d4c67eba5b0e9e1a"),
"name" : "compositeType",
"class" : "cmp",
"children" : [{ "class" : "qlt",
"name" : "aQualitativeType",
"uuid" : ♠,
"values" : ["Phenomenon1", "Phenomenon2"] },
{ "class" : "ref",
"name" : "aNamedType",
"uuid" : ♣,
"reference" : ObjectId("55e45670d4c67eba5b0e9e16")}]
{ "_id" : ObjectId("55e45670d4c67eba5b0e9e16"),
"name" : "namedType",
"class" : "cmp",
"children" : [{ "class" : "txt",
"name" : "aTextualType",
"uuid" : ♦ },
{ "class" : "qnt",
"name" : "aQuantitativeType",
"uuid" : ♥,
"units" : [{ "name" : "unit1",
"symbol" : "u1",
"format" : "5,2" },
{ "name" : "unit2",
"symbol" : "u2",
"format" : "3,4" }] }]
{ "_id" : ObjectId("552168dcdb77569140663ba7") ,
"class" : "cmp",
"typeId" : ObjectId("62e45671d4c67eba5b0e9e1a")
"children" : [{ "class" : "qlt",
"type" : ♠,
"value" : "Phenomenon1" },
{ "class" : "cmp",
"type" : ♣,
"children" : [{ "class" : "txt",
"type" : ♦,
"value" : "aText" },
{ "class" : "qnt",
"type" : ♥,
"value" : "10.2",
"unit" : "u1" }] }]
}

Equivalence of models

We have considered equivalent two data representations of the same domain logic using two different persistence models when the carried information can be serialized into equivalent strings of text.

- We chose a dataset with an arbitrary number of clinical information data, persisted in the relational model;
- we have retrieved all the Examinations and ExaminationTypes instances and serialized the information data in a string representation;
- we have converted information data from the relational model to the target NoSQL model, serializing again the information data, and comparing the resulting string with the string obtained from the relational model at the previous step.

The structure of the serialization is deliberately similar to a JSON document, due to its simple and readable syntax.

Experimentation - 1 (Case of use)

Experimentation's methodology

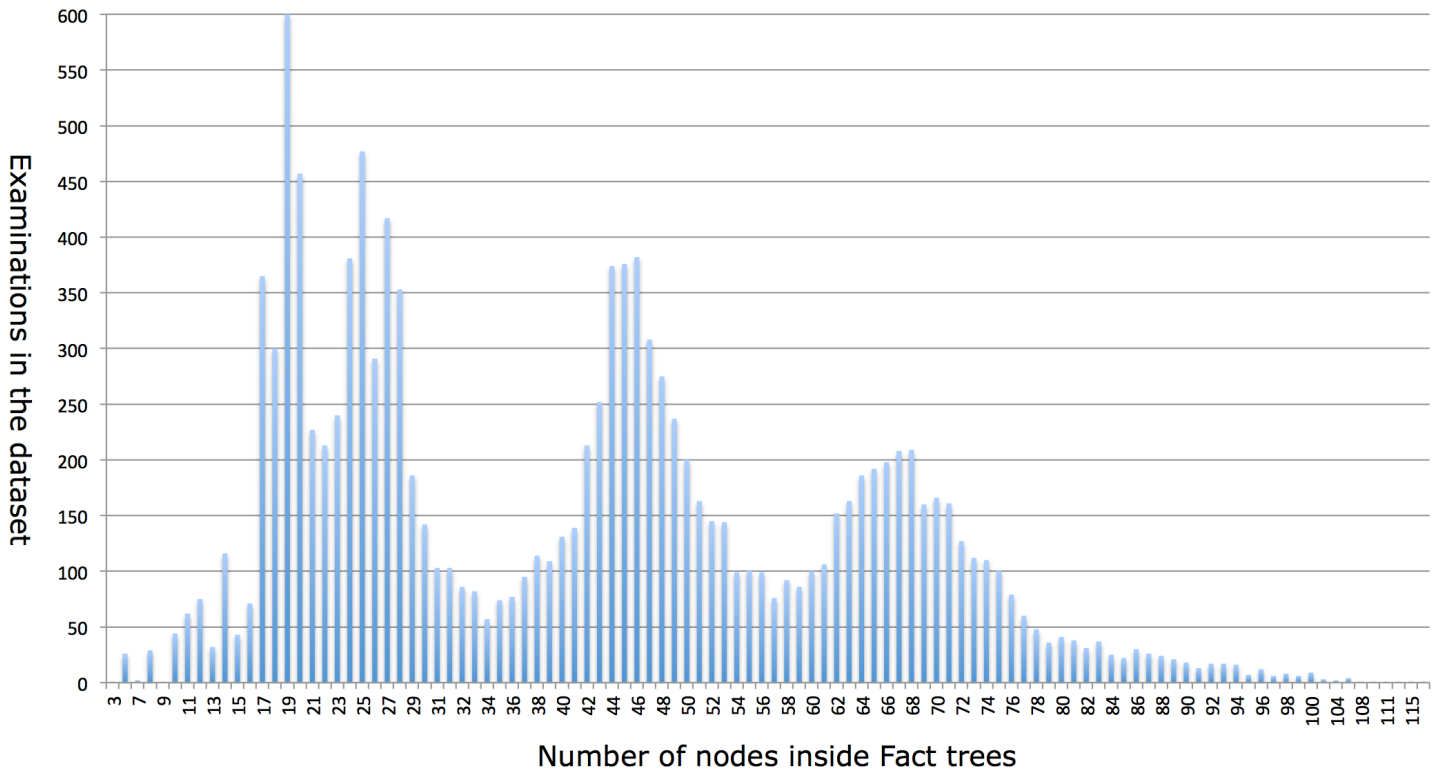
- Case of use under test
- Dataset for experimentation

Chosen major scenario of interaction: a health professional accesses a patient's EHR content in order to review past medical examinations and to read collected clinical information.

- we evaluated the total time from data retrieving to data serialization;
- retrieval also involves the associated ExaminationType since it contains the description of the Facts.

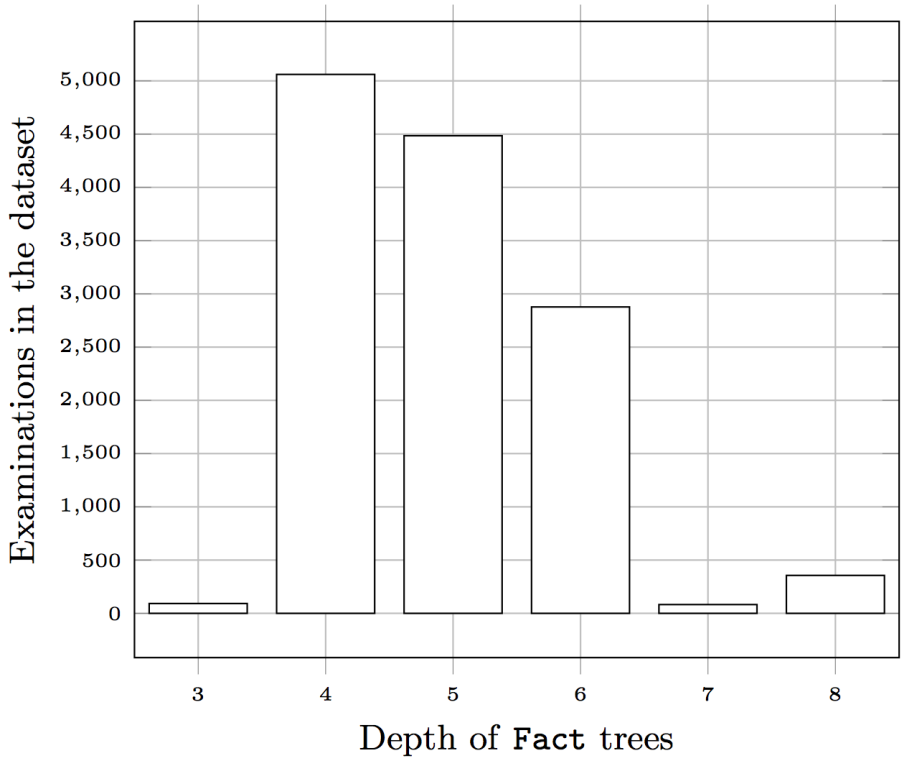
Experimentation - 2 (Real Dataset)

- a real dataset of clinical examinations acquired in our EHR system for which we provide a description of the statistics of the number of nodes and depth of the tree
  - 13000 examinations that belong to the same speciality and thus share the same structure.



Experimentation - 2 (Real Dataset)

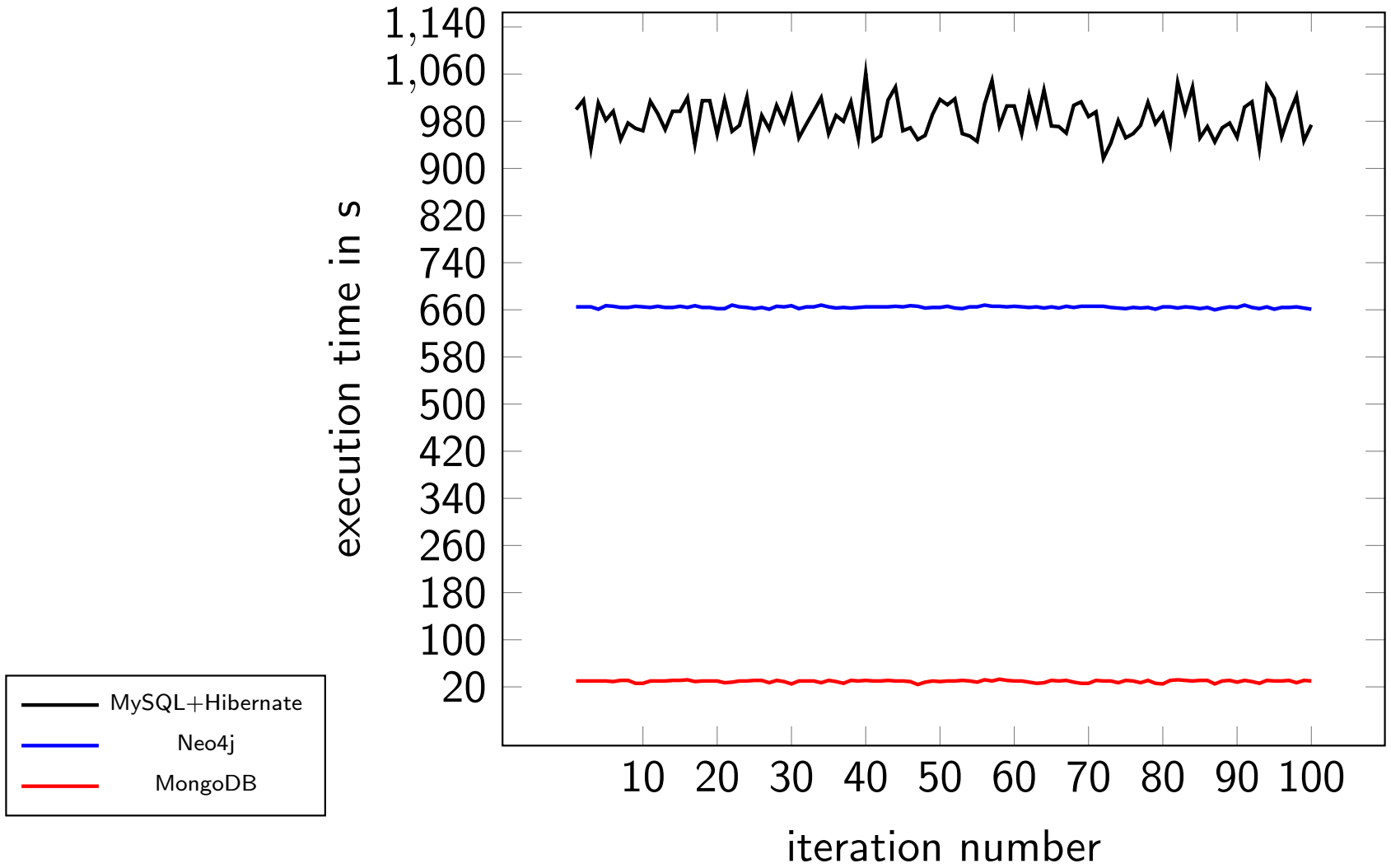
- a real dataset of clinical examinations acquired in our EHR system for which we provide a description of the statistics of the number of nodes and depth of the tree
  - 13000 examinations that belong to the same speciality and thus share the same structure.



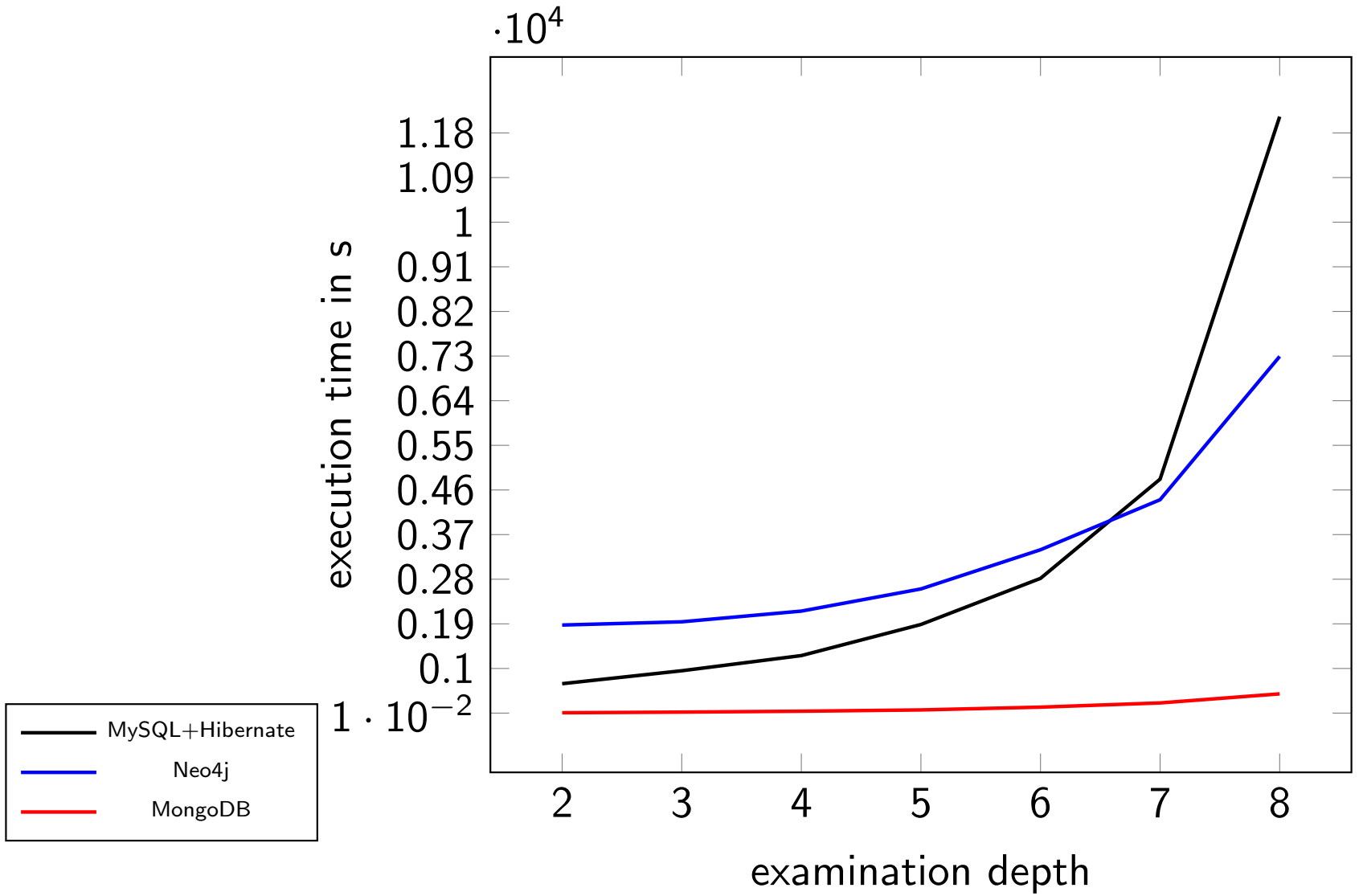
Experimentation - 3 (Synthetic Dataset)

- a synthetic dataset for which we can control the statistics so as to stress the indexes of complexity
  - built as a full binary tree, with depth ranging from 2 to 8
  - for each ExaminationType, a fixed number of Examinations with the same characteristics has been generated.

Results - 1 (Real Dataset)



Results - 2 (Synthetic Dataset)



Conclusions

- both tested NoSQL technologies offer advantages in terms of flexibility in the data model, scalability and reliability;
- the graph-oriented data model of Neo4j allows a more natural and direct data conversion
  - presents a performance increase of 1.5 times compares to MySQL + Hibernate
- the document-oriented data model of MongoDB produces better performance results
  - presents a gain of almost 33 times compares to MySQL + Hibernate.

Ongoing activities: Explore performances of NoSQL databases in other use cases, which impact on the application is less relevant but nonetheless interesting to have a full comparison between the various models.