TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

Valeriia Shpychka 134455

ANALYSIS OF PERFORMANCE AND COMPLEXITY OF BUILDING A WEB APPLICATION BASED ON COUCHBASE AND POSTGRESQL WITH JSONB TYPE

Bachelor's thesis

Supervisor: Erki Eessaar
PhD

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Valeriia Shpychka 134455

JÕUDLUSE ANALÜÜS NING VEEBIRAKENDUSE LOOMISE KEERUKUS COUCHBASE JA JSONB TÜÜBIGA POSTGRESQL PÕHJAL

Bakalaureusetöö

Juhendaja: Erki Eessaar
Doktor

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Valeriia Shpychka

22.05.2017

Abstract

The present work has two goals: compare the performance of data manipulation operations of two selected database management systems (DBMSs) with each other and compare the complexity of building a Java web application over those DBMSs. The DBMSs selected for the experiment are PostgreSQL 9.6 and Couchbase community edition 4.5.1. The first of these is a SQL DBMS and the second is a document-based NoSQL DBMS.

I selected the domain of e-health as the domain of the example databases and applications. In order to do that, I analysed several e-healthcare standards. After the comparison, I selected HL7 FHIR [49] standard. Based on the structure of resources in HL7 FHIR I designed the databases. In case of PostgreSQL, the used database schema is a mix of "traditional" relational design and the use of document-based representation of data. The latter means that some data is represented as values of JSONB type. In case of Couchbase, the only option is to use document-based representation of data.

I will compare the performance of DBMSs based on the execution times of data manipulation operations. Each query will be tested with different data sizes: one million objects (in the main table/document type *Document*), 500 thousand objects, and 250 thousand objects. In order to understand the relationship between the amount of data and the execution time, I will calculate Pearson correlation coefficient and create an illustration based on the results. This comparison is needed to understand, which DBMS (PostgreSQL or Couchbase) has higher efficiency with large data.

I will compare the complexity of building a Java web application based on the time that I will spend on implementing the functionality described in the use cases. It means that the evaluation of the complexity is subjective in the present work. In the scope of testing the complexity of building a web application over the selected DBMSs, I will perform an experiment with data schema change. I stress that the created test application is not meant to be used as a real-world e-healthcare system.

An important result of this work is discovering the relationship between the amount of data and the time consumed by query execution in terms of the performance comparison experiment. Another result consists of two web applications over PostgreSQL and Couchbase DBMSs, respectively. Their creation helps me to understand as to how much work is required to integrate Java web application with those DBMSs. Yet another result comes from comparing the DBMSs and applications that are built on top of these in terms of how easy it is to change the database schema.

This thesis is written in English and is 65 pages long, including 11 chapters, 47 figures and 8 tables.

Annotatsioon

Jõudluse analüüs ning veebirakenduse loomise keerukus Couchbase ja JSONB tüübiga PostgreSQL põhjal

Käesoleval tööl on kaks eesmärki: võrrelda omavahel kahe valitud andmebaasihalduri andmekäitluskeele lausete täitmise jõudlust ning ühtlasi võrrelda, kui keerukas on nende andmebaasihaldurite abil ülesehitada Java veebirakendust. Andmebaasihaldurid, mida antud uurimistöö raames võrreldakse, olid PostgreSQL 9.6 ja Couchbase Community Edition 4.5.1. Esimene nendest on SQL-andmebaasihaldur ja teine dokumendipõhine NoSQL andmebaasihaldur.

Valisin näiteandmebaaside ja rakenduste valdkonnaks e-tervise. Eesmärgi saavutamiseks analüüsisin mitmeid e-tervisehoiu standardeid. Analüüsi tulemuste põhjal otsustasin HL7 FHIR [49] standardi kasuks. Disainisin andmebaasi HL7 FHIR ressursside struktuuri põhjal. PostgreSQL korral on realiseeritava andmebaasi skeem segu „traditsioonilisest“ tabelite disainist ja dokumendipõhisest andmete esitusest. Viimane tähendab, et osa andmeid esitatakse JSONB tüüpi väärtustena.. Couchbase’i korral on ainus võimalus kasutada dokumendipõhist andmete esitust.

Võrdlen andmebaasihaldurite suutlikkust andmekäitluskeele lausete täitmisele kulunud aja alusel. Igat lauset testitakse erinevate andmemahtudega: miljon objekti (rida põhitabelis/dokumendi tüübile vastavat dokumenti), pool miljonit objekti ja veerand miljonit objekti. Mõistmaks andmemahu ja päringule kulunud aja omavahelist seost, arvutan Pearsoni korrelatsioonikordaja väärtuseid. Võrdlus on vajalik mõistmaks, milline andmebaasihaldur (PostgreSQL või Couchbase) on suurte andmemahtudega töötamisel efektiivsem.

Võrdlen Java veebirakenduse loomise keerukust aja alusel, mis kulub kasutusjuhtudes kirjeldatud funktsionaalsuste realiseerimiseks. See tähendab, et hinnang veebirakenduse loomise keerukusele on antud töös subjektiivne. Viin läbi andmete skeemimuutuse eksperimendi, et testida veebirakenduse edasiarendamise keerukust valitud

andmebaasihaldurite korral. Rõhutan, et uurimistöö raames loodud rakendus ei ole loodud e-tervisehoiu süsteemina kasutamiseks.

Oluline töö tulemus on leida seos andmemahu ja päringule kulunud aja vahel andmebaasihaldurite suutlikkuse katse raames. Veel üks oluline tulemus on kaks veebirakendust, mis kasutavad vastavalt PostgreSQL ja Couchbase andmebaasihaldurit. Nende loomine aitab mõista, kui palju aega kulub Java veebirakenduste sidumiseks nende kahe andmebaasihalduriga. Samuti on oluliseks tulemuseks andmebaasisüsteemide ja nende peale ehitatud veebirakenduste võrdlus selles osas, kui lihtne on andmebaasis teha skeemimuudatusi.

Uuringu tulemusena kujunes mul arvamus, et eksperimendi ülesande (tsentraliseeritud e-tervise süsteem loomine) täitmiseks on PostgreSQL 9.6 parem kui Couchbase 4.5.1.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 65 leheküljel, 11 peatükki, 47 joonist, 8 tabelit.

List of abbreviations and terms

ACID	Atomicity, Consistency, Isolation, Durability
AHP	Analytic Hierarchy Process, also known as Saaty method
Big Data	Large volume of both structured and unstructured data that comes in with increasingly fast flow
DBMS	Database Management System
EHF	The Estonian E-Health Foundation
EHR	Electronic Healthcare Record
ENHIS	Estonian National Health Information System
EU	European Union
FHIR	Fast Healthcare Interoperability Resources
HI	Healthcare Informatics
Java	An object-oriented programming language
JSON	JavaScript Object Notation
JSONB	Binary JSON
MVC	Model-View-Controller
NoSQL	Not Only SQL
ORM	Object-Relational Mapping
SQL	Structured Query Language
SQL DBMS	A DBMS where one can use SQL language
TUT	Tallinn University of Technology
UML	Unified Modeling Language

Table of contents

List of figures

List of tables

1 Introduction

The world of information technology is facing new challenges every day. The number of users of various systems is constantly increasing, which leads to the necessity to store and process large amount of data. In case of badly designed databases and applications, the time of the user requests processing grows exponentially with the growing amount of data in the system. Previously, architects preferred SQL database management systems (DBMSs), since those systems are highly standardized. Unfortunately, not every application based on SQL DBMS could be efficient with the large amount of data. At this point, NoSQL DBMSs might be a solution. There are plenty of papers available on Web where one can find the following summaries “NoSQL databases can be summarized as high scalability and reliability, very simple data model, very simple (primitive) query language” [33] .

Although SQL standard is not prescriptive, SQL DBMSs still have to follow a core of the standard. In case of creating a SQL database, one has to explicitly declare data schema. There is a voluminous and complicated standard of SQL[57] , although widely used DBMSs usually do not follow it completely.

NoSQL DBMSs advertise themselves by high scalability and flexibility. The declaration of data schema is usually possible, but not required. Those DBMSs can be a part of a solution for big data processing since it is possible to distribute data over the clusters. It increases scalability and data availability. To achieve scalability, the systems weaken their guarantees to transactions.

The first goal of the present work is to compare the performance of data manipulation operations of two DBMSs. One of these – Couchbase – is a NoSQL DBMS that provides document-based data model. Another – PostgreSQL – is a SQL DBMS that allows developers to use columns of JSONB type in its schema and thus integrates SQL and document-based data model. The second goal is to compare the complexity of building a Java web application based on the DBMSs.

I have selected PostgreSQL as a representative of SQL DBMSs, since it allows us to have a JSON and JSONB type columns, which makes it more flexible in terms of database schema change. It is also a good solution for storing semi-structured data in the columns of JSON or JSONB type. At the same time, it still allows us to use the benefits of the mature SQL system. Moreover, it is popular, open-source, free, and follows the SQL standard quite well. I will use JSONB type in the PostgreSQL database schema, since the internal representation of its values is more compressed compared to JSON values. It is beneficial when there is a need to store large amount of data.

In order to choose a document-based NoSQL DBMS, I will use analytic hierarchy process (AHP) and will build a decision model. Right away I decided not to use MongoDB because it has been analyzed in plenty of other works [41] [42] . Nevertheless, I will add MongoDB as one of the alternatives to the AHP in order to compare it with other possible alternatives – Amazon Dynamo DB and Couchbase – which are also at the high place of the DBMS popularity index [58] as of spring 2017. Although there are papers that compare the performance of PostgreSQL and Couchbase [43], I decided to select this DBMS because it was the second-best option according to the selection process.

In the experimental part I will compare the DBMSs only based on performance and complexity of building web applications, because the performance of the application influences the experience of the end user. Performance of executing data manipulation statements by a DBMS influences a lot the overall performance of the system. Developers must build efficiently working, fast systems in a short time. Therefore, in the present work I will examine how much time and effort it will take to build a Java web application over the two selected DBMSs, and will compare the performance of those DBMSs.

The work will be useful for researchers who want to see a comparison process of DBMSs. It might be also interesting for system engineers who can see about the strong and weak points of each DBMS based on the cases considered by this study.

2 Initial Description of the Experiment

Like each tool, each DBMS has usage scenarios where its usage would be productive and usage scenarios where its usage would be counter-productive. There are hundreds of DBMSs [25]. Architects and designers have to select between these quite often. Of course, DBMSs evolve over time and conclusions that were made by comparing older versions might be wrong in case of the newer versions. Therefore, it is very important to describe the comparison process so that it was applicable to the future versions of the system as well (see Chapter 4). In order to compare DBMSs, I have to investigate these based on literature (see Chapter 3).

Experiments will be based on PostgreSQL (9.6) and Couchbase (Community edition 4.5.1).

Firstly, I will design a PostgreSQL database (see Chapter 7) and Couchbase database (see Chapter 8). I do it based on the same conceptual data model and general assumptions about the system (see Chapter 6). The domain of the databases and their respective applications is e-health. I will give background information about the domain in the Chapter 5.

In order to compare DBMSs performance, I will test it in terms of data growth based on the same set of data manipulation operations (Read, Update, and Create). The size of datasets that I will use in the experiment are: one million objects (in the main table/document type *Document*), 500 thousand objects, and 250 thousand objects. The comparison will be done based on the execution times of data manipulation operations. Both DBMSs make it possible to use document-based (hierarchical) representation of data. The data in both databases will be identical by content, but will have a different structure. The conceptual data model, which is a base for both implementations, can be found in the section 6.2. The difference in data schema should bring out the strong sides of each DBMS. The exact operations based on the data, results, and the analysis is in the Chapter 10.

As it was written above, having higher scalability and accessibility, NoSQL systems weaken other important aspects like data consistency. The scalability and accessibility is

achieved by distributing data over clusters. Although the possibility to distribute data over clusters is one of the most important NoSQL specialties, in the test application only one node is used. This is a weakness of my work. Since we are trying to get the created SQL-based system as close to NoSQL-based systems as possible, JSONB type for documents in PostgreSQL is used.

Since I try to use real-life scenarios in the present work, both DBMSs will be tested in case of data structure change. Almost every real-world system faces the data structure changes. It leads to the necessity to change the data schema. In the present work, I will test both DBMSs in terms of the data schema change and try to estimate, what were the consequences, how much time did it take to process all the changes and how many problems did I face. More detailed description of the experiment can be found in chapter 10.3.

In order to compare the performance of database operations as well as the ease of developing an application on top of a database, I will build a web application – prototype of an e-health information system. The evaluation of complexity of building the web applications will subjective, because I will consider only the time and effort that development took. Since the amount of data in e-health systems is growing rapidly, and the nature of data is quite sensitive, the real-world e-health systems have to be highly secured and scalable. The goal of the present work was not to build an e-health system itself, but to test two certain DBMS' in terms of real world Java application. The domain of e-health was chosen because in e-health system documents have different types and therefore different structures. The amount of data is constantly growing so the time spent on requests processing should be minimized. Although the application will not be implemented in accordance with all the requirements of a real e-health system, it allows me to see the advantages and disadvantages of both DBMSs.

Application will allow me to do the following.

1. Compare the complexity of building the Java web application based on two different DBMSs: Couchbase – a NoSQL DBMS and PostgreSQL with JSONB types, which is a SQL DMBS.
2. Compare the flexibility of chosen DBMSs in case of data schema change.
3. Analyze the performance of data manipulation operations.

I will use Gradle (3.1) project management tool and Spring (1.5.2 RELEASE) framework for building the application. In the application built over PostgreSQL DBMS, I will use Hibernate 6.0.0 Alpha2

In addition to overlooking securing matters, the application will not have tests as well. Although having tests is crucial for any real-world application, this is out of the scope of the present work. Therefore, application will not have automated tests.

In order to have more realistic data structure, the database design will be implemented according to HL7 FHIR standard (Release 3 STU), which is a standard for structuring Electronic Healthcare Records. Additionally, the challenges of HI will be analyzed in the present work.

3 Theoretical Background

The present work analyzes the performance of PostgreSQL and Couchbase DBMSs and analyzes the complexity of building an application on the top of the DBMSs. Since SQL and NoSQL DBMSs are quite different, in this chapter both families of DBMSs will be briefly introduced. The topic itself is quite voluminous. Thus, some of the key differences of SQL compared to NoSQL systems (ACID, BASE, scalability, etc.) will not be tested in the thesis and the information present in this chapter is just to give an overview about the differences of those DBMSs.

3.1 NoSQL

NoSQL (which is an acronym for Not Only SQL) systems are an alternative to SQL systems. Those are non-relational DBMSs where the database does not consist of tables and SQL is not used for data manipulation. NoSQL is an umbrella term that describes many different systems that feature different representation of data. This kind of DBMSs might be quite useful when there is a need to store big amount of unstructured data. NoSQL systems are quite handy if there is no strong need in ACID (which is an acronym to Atomicity, Consistency, Isolation and Durability) transactional guarantees since NoSQL systems mostly follow a weaker BASE (BASE is an acronym for Basically Available, Soft-state, Eventually consistent) approach. The main use-cases of NoSQL systems are [26]:

1. large-scale data processing (when there is the need to process the data in parallel over distributed systems);
2. storing of a voluminous data that has varying structure.

There are several types of NoSQL DBMSs. The Table 1 lists the types with the examples [61] . These types correspond to different ways how to represent data. A commonality is that one does not have to explicitly define a database schema when creating the database in such systems. Thus, the word “schemaless” is used to characterize the databases and the DBMSs. It is incorrect because the data has to have schema in order to be usable. However, the schema is not explicitly defined in the database. Instead, it is implicitly specified in the application code. The following table does not contain object-oriented DBMSs. Although they are also non-SQL systems they are often not counted as a part of

the NoSQL movement because the systems were at the market long before the current NoSQL hype.

Table 1Types of NoSQL DBMSs

Data representation type	Description	Examples
Key-Value Stores	Works by matching keys with values. Such systems are highly scalable and quite efficient with storing and retrieving semi-structured (actually better to say "differently-structured") and unstructured data (unstructured for the DBMS that does not understand the structure of the values).	1. Redis 2. Memcached 3. Riak KV 4. Hazelcast 5. Encache
Wide column stores	Are based on the key-value model. Data is represented as a two-dimensional array. The data is stored in records which have the possibility to hold large number of dynamic columns.	1. Cassandra 2. Hbase 3. Accumulo
Document stores	Compared to the key-value stores the values are not unstructured blobs for the DBMS but documents. The DBMSs can see within the document values and use their content to search data or modify data. Documents might be stored in JSON, BSON, XML, etc. format. Documents (records) in such DBMSs do not require a unified structure. Thus, documents may have different and possibly nested structure. In some implementations, indexing is possible.	1. MongoDB 2. Amazon DynamoDB 3. Couchbase 4. CouchDB 5. MarkLogic
Graph DBMS	Graph-oriented DBMS represent data in graph structure – nodes and edges. The latter represent the relationships between the nodes.	1. Neo4j 2. OrientDB 3. Titan

Data representation type	Description	Examples
		4. ArangoDB 5. Virtuoso
RDF DBMS	The Resource Description Framework is a subclass of graph-oriented DBMS since it represents the data in triples – subject-predicate-object where predicate is a connection between subject and object. This type of DBMS was originally developed for describing metadata of IT resources. Nowadays is widely used in the semantic web.[62]	1. MarkLogic 2. Virtuoso 3. Jena 4. Algebraix 5. AllegroGraph
Native XML DBMS	Data is represented in terms of XML documents. It is possible to store hierarchical data in such DBMSs. It also allows embedded declarations in XML documents and supports XML-specific query languages (XQuery, XSLT, XPath)	1. MarkLogic 2. Virtuoso 3. Sedna
Content stores	Type of DBMS, which specializes in storing digital content. Apart from storing and querying, they provide the possibility of full-text search and hierarchical storing of data.	1. Jackrabbit 2. ModeShape
Search engines	As the name says, it is the type of DBMS dedicated to searching from data content. Allows different types of searches, including full-text search, geospatial search, and distributed search for higher scalability.	1. Elastic search 2. Solr 3. Splunk 4. MarkLogic 5. Sphinx

3.2 SQL

SQL DBMS (Database Management System) is the DBMS where SQL (Structured Query Language) is used to manipulate the data. Tables are used to represent the data in SQL databases. Each table consists of n-number of columns and m-number of rows. (where n>=1 and m>=0) Each column must have a type (String, Boolean, etc.). Rows in different tables as well in the same table are connected by using data values (no hidden references) – foreign key values. SQL DBMSs are believed to be less tolerant to data-structure changes since their databases have explicitly defined schema. Thus, changes in the data structure lead to the necessity to change the database schema. SQL DBMSs are also known for being weak in horizontal scalability but quite vertically scalable. It is proven by well-known services [26] that scalability depends much on the implementation. SQL DMBSs traditionally use strong guarantees for transactions and try to follow ACID properties (Atomicity, Consistency, Isolation, and Durability):

- **Atomicity** – an atomic transaction means that this transaction must be or fully completed or rolled back, meaning no changes of the data are made as the result [2] .

- **Consistency** – no transaction cannot violate the consistency of the database. It means that the operations, which break the integrity rules, cannot be finished.[2]

- **Isolation –** each transaction is isolated from all the other transactions, meaning that transactions should not see incomplete results of other transactions in order to avoid mistakes in data processing. SQL specifies different isolation levels.[2]

- **Durability –** if the system successfully completes a transaction, then the results (data modifications) of the transaction cannot be lost by the system.[2]

3.2.1 Building a Document Store by using PostgreSQL

PostgreSQL is one of the SQL DBMSs. The popularity of PostgreSQL is growing for multiple reasons. It has voluminous number of useful plugins, ACID compliance, full-text search, indexing, etc. [3] PostgreSQL provides the datatype called Hstore. Each value that belongs to the type is a set of key-value pairs. However, it is not enough to store the documents and provide the possibility to process the documents. PostgreSQL supports the JSON type, allows us to create columns with such type, and allows us to

create indexes on such columns. Starting from the 9.4 release it provides a possibility of using JSONB type, which will be used in the example database [4] . In case of JSONB, the JSON value is internally represented in a decomposed binary form. It is a compressed representation, which suits better the need to store larger amount of data. This is the difference from the JSON type, in case of which the exact copy of the input text is stored.

The JSON types bring the flexibility of the NoSQL databases to an SQL database, although the consistency of the database might be suffering from that – in this case the possibility of having duplicated data is much higher.

4 Comparing Some Document-Based DBMSs

Table 2 contains comparison of the document-based NoSQL systems that I considered as a possible representative of NoSQL systems. Table 2 also describes criteria that I will use to choose a NoSQL system that I will use in the experiment.

Table 2Comparison of some document-based NoSQL DBMSs

	MongoDB	Couchbase	Amazon DynamoDB
Version as of May 2017	3.4	4.6.0	API Version 2012-08-10
Year of the initial release	2009	2011	2012
DBMS popularity ranking [58] as of May 2017 (document stores/overall)	1/5	3/23	2/22
Data-schema	Flexible (not required)	Not required	Requires key-schema
Open source	✓	✓	
Object references	✓ (Manual and DBRefs)	✓	
Primary key	✓	✓	✓
Indexing	✓	✓	✓
Joins	✓	✓	
Java programming language support	✓	✓	✓

One of the most famous and widely used NoSQL DBMS is MongoDB. It is a popular DBMS so that it takes the first position in the rank of document stores and fifth position in general ranking [25] as of the March 14, 2017. The constant development and popularity growth makes MongoDB a de-facto standard for document stores.

The Amazon DynamoDB DBMS also belongs to the NoSQL family. First release was in the year 2012 and by 14th of March 2017, it has taken the 2nd position in the rank of document stores and 22nd position in the overall ranking of DBMSs.

Couchbase is a NoSQL document store, which was released in the year 2011. By the current moment (May 2017), it takes the third position in the document stores ranking and 23rd position in the DBMS ranking.

In this paragraph, those three DBMSs will be compared based on the criteria mentioned in the Table 2. The versions of DBMSs are the following: MongoDB – 3.4, Couchbase 4.6.0, and Amazon DynamoDB of the version release on 12 November 2015.

1. Open-source.
    - *MongoDB* is an open-source DBMS, which provides also commercial services. [22]
    - *Couchbase* is an open-source DBMS, which provides commercial services. [23]
    - *Amazon DynamoDB* is not open-source DBMS and is provided on the commercial base. [25]
2. Data-schema.
    - *MongoDB* has flexible data-schema, which means that data-schema is not required, but there are mechanisms, which allow validating documents by the set of rules. Those rules are defined on a per-collection basis with “validator” option. Validation might be performed during the “update” and “insert” query executions, which means that “old” documents will not be validated. There is also an option called “validationLevel”, which provides the possibility to control the handling of the existing documents. Default option is “strict”. In this case, only the documents which go through the update/insert operations are validated, the rest of objects remains the same. However, it is possible to set the value to “moderate”, which allows validating updates and inserts to the documents that match the validation criteria. [24]

- *Couchbase* does not require a data-schema. Moreover, it delegates the validation of the document and the definition of the structure of the document to be stored to the application itself. [23]

- *Amazon DynamoDB* claims to be schema-free, although some set of definitions is still required. In order to create a table, one has to declare the "AttributeDefinitions" that represent the list of column names and types. "AttributeDefinitions" describe the indexes, table key schema, and the "KeySchema" which defines the primary key for the table. [25]

3. Referencing of objects

   - *MongoDB* provides two methods to reference documents: manual referencing and DBRefs. The manual referencing works in the following way: the _id field of the "foreign" document is stored in the "target" document as a reference. It allows application to run a query to retrieve a "foreign" document in case of the need. In order to retrieve the "foreign" document with DBRef, one has to reference the "foreign" document from the target document by the _id, collection name, and database name, if needed. In this case, application must run a query to retrieve the "foreign" document as well. [24]

   - *Couchbase* allows to reference "foreign" documents by using the "item-key", which is the unique identifier of the "foreign" document. In Couchbase documents have unique identifiers (meta().id of the document), which might be treated as a primary key. This key should be stored in the corresponding field of the "target" document. [23]

   - *Amazon DynamoDB* does not support referencing. [25]

All DBMSs, which are described in this paragraph, support embedding, which can cover some foreign document use cases.

4. Primary key

   - *MongoDB* requires each document to have _id field with the unique value. If new document does not have an _id field, then DBMS automatically generates the field and assigns a unique BSON ObjectId. The system generates it based

on the current timestamp, process-local incremental counter, and process- and machine- id. [24]

- *Couchbase* requires all the documents to have a unique identifier – id. Assigning an id is a responsibility of the application. It may have any form until the length is less than 250 bytes and it has UTF-8 encoding. Although application performs the generation of ids, Couchbase provides the counter. It eases the process of id generation by incrementing the counter on every insert. [23]

- *Amazon DynamoDB* requires each document to have the primary key. There are two types of primary keys supported by Amazon DynamoDB: partition key and composite primary key. The last one forms of partition and sort keys. The partition key is formed by the help of internal hash function. Based on the output of the function, DynamoDB chooses the internal physical storage where it will store the item. The "composite" key consists of the described above partition key and sort key. All the items with the same partition key are stored together and sorted by the sort-key values. Amazon DynamoDB allows several objects to have the same partition key. Those objects must have different sort-keys. Each primary key must have only one value (be scalar). [25]

5. Indexing

   - *MongoDB* supports indexing. Indexes are created by the special command: db.collection.createIndex(). It can be created only in case where there is no already existing index with the same specification. By default, index is created on the _id field. There are also other types of indexes available: single field index, compound index, multikey index, geospatial index, text index, etc. [24]

   - *Couchbase* supports two types of indexes: global and local indexes. Local indexes are used for complex index logic and global ones are used for low latency queries. [23]

   - *Amazon DynamoDB* supports secondary indexes, which are a set of attributes and keys to support querying. Every secondary index is associated with one

and only table. It is the “base table” for the index. DynamoDB supports two types of secondary indexes – global and local secondary index. The difference in those types of indexes is that global index might have partition and sort keys different from the base table, when local index must identical partition key and a different sort key with the base table. [25]

6. Joins

    - *MongoDB* supports left outer join provided by the $lookup aggregation stage, which is applied to unsharded collection within one database. [24]

    - *Couchbase* uses N1QL – “a query language that extends SQL for JSON” [60] . N1QL provides two types of joins: index and lookup joins. Lookup join is only performed from left-to-right. By default, inner join is performed. If “LEFT” of “LEFT OUTER” are specified, then DBMS will perform left outer join. [23] . Index join helps us to join parent table with a child table. It works in the following way: scans using index on a selected key using meta().id. If it finds something, then it fetches join, groups, and aggregates. Index join can be combined with other types of joins.

    - *Amazon DynamoDB* does not support joins. [25]

4.1 Selecting the Document Store

AHP (Analytic Hierarchy process) is a method that helps us to make decisions (choose between alternatives) based on a set of the criteria. It uses the pairwise comparison of criteria and later alternatives in terms of each criterion. As a result it helps us to calculate the relative importance of each criterion and later each alternative. Based on the author of the method it is often called Saaty method. In order to use this method, the problem should be determined and structured as a hierarchy. Criteria should form the tree and the matrix for every non-leaf should be created. Based on the matrices, one has to create the priority vectors. The priorities should be aggregated by levels in the following way: “the priorities from one node are used to weight the priorities in the node below and then are added to obtain the global priority” [21]. For performing the analysis in the current work, I chose PriEst tool. In the current work all the pairwise comparisons were done only by me – no other experts were used.

The criteria were the following.

1. Primary key support – primary key is an important unique identifier for the object in the system. It is the easiest way to refer to the object.

2. Foreign key support – having foreign keys is useful for data consistency. When objects are stored in embedded way, the change of a value in a certain field in embedded object leads to the need to change the value over all objects in the system. Having foreign key reference makes it easier – one has to change the document only in one place.

3. Indexing – it is important when it comes to retrieving the data. Indexes make data retrieving faster and more efficient.

4. Join operations support – it makes querying easier, when one has to select values from more than one document.

5. Schema-free (not required) – is one of key features of NoSQL databases.

6. Java support – it is important because I will implement a Java based application.

7. Open source – it is one of the criteria, since usually open source products do not require a subscription fee. In my case this criteria is not crucial since payed services usually provide a trial period, which could be enough to perform the experiment.

I evaluated the criteria based on the needs of the system to develop.

Color legend:

1. **Primary key support;**
2. **Foreign key support;**
3. **Indexing;**
4. **Join operation support;**
5. **Schema-free;**
6. **Java support;**
7. **Open-source;**

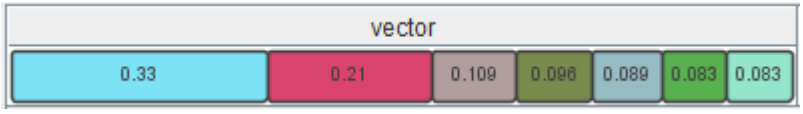

Figure 1AHP - Criteria Importance

On the Figure 1, the importance of the criteria listed above is shown correspondingly.

The DBMSs that were compared in terms of the previously mentioned criteria are presented on Figure 2.

Figure 2 AHP - Options to Compare

The results of the comparison are shown on the Figure 3.

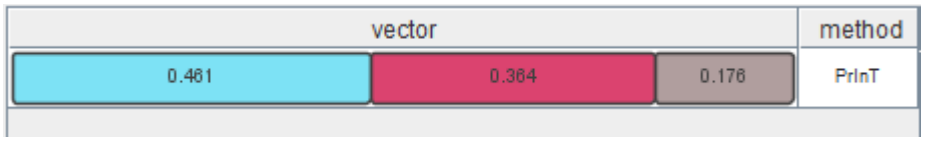

Figure 3 AHP - Results of the Comparison

According to the results, the best option would be MongoDB. However, as it was said before, MongoDB is a widely-used DBMS which has already been described in other works. Therefore, I selected Couchbase because it is the second-best option. The comparison matrices and sensitivity analysis can be found in Appendix 6 – AHP Matrices. Expectations

After making a theoretical research about strong and weak sides of each used DBMS, I have a set of expectations.

1. Couchbase will have much better performance with big data sizes compared to PostgreSQL. The reason of the expectation is that NoSQL DBMSs claim to guarantee fast data-access and be efficient with big data sizes.

2. Couchbase will be more tolerant to data-structure change (the data-schema change experiment). This assumption is based on the fact that there is no data-schema declared in Couchbase. Therefore, the experiment with the Couchbase version of the application does not include DBMS-related work. It should be just a set of application-level changes. I expect that PostgreSQL will be less tolerant to this experiment, since, apart from application-level changes, the experiment will also include creating a migration script to process data-schema change.

3. I expect that PostgreSQL-based Java application will be easier to build because PostgreSQL is a mature DBMS, which has integration with plenty of Java libraries. It also has enough of meaningful documentation available on WEB. I

expect that using Couchbase will not be smooth because the DBMS is quite new (came to market in 2011). It means that there will obviously be less information available on WEB compared to PostgreSQL.

5 Digital Challenges in Healthcare

In the scope of the current work e-health applications will be developed in order to compare DBMSs. I will create applications for this particular domain because in real life such systems have to work with big amount of data. Because health-related processes produce a lot of documents, the document-based representation of data seems natural. Moreover, this data has very variable structure and thus the “schemaless” nature of the document-based data representation might be an advantage.

Nevertheless, I want the application to be somewhat realistic. The application must not be treated as a real e-health system since real systems has much more strict requirements especially when it comes to storing sensitive patient’s data. In order to understand how real systems are build and what are the real requirements, I have checked which challenges those systems are facing. Based on the literature Iwill describe it in the current chapter.

Healthcare records are increasingly becoming digitalized which sets new requirements for e-healthcare systems. Modern systems should solve the following problems [8] .

1. Growing flood of data – “increasing data generation and the need for its secure storage and management” [8] .

2. Privacy requirements – all the sensitive data should be secured and restorable in case of need.

3. Data storage capacity – healthcare data storages are growing rapidly due to the need to keep patient’s data indefinitely.

4. Lack of communication among different healthcare systems – systems have limited capability to exchange data, which restricts the ability to automate processes.

5.1 E-Health Systems: Overview of the Requirements and Current State in Different EU Countries

There has been a lot of development of e-health systems during the last decades. It has lead the world to the definition of certain requirements to such systems. Those requirements refer to the data that must be stored, to the format of that data, to security measures, etc. In this paragraph, the requirements to the real-world e-health system will be described based on the EU standards to EHR. EHR – electronic health record – "systematized collection of patient and population electronically-stored health information in a digital format" [11] .

1. Health data to be included to EHR.

Different countries have different laws at this point. According to the overview of national laws on electronic health records in EU [12] , some EU member countries (as of 2014) require that, apart from general administrative data, only health data is included to EHR. It means that developers who will implement the system should be acquainted with the local law.

Although there is no legal definition of EHR in Estonia, regulation details the list of the documents that must be uploaded to the ENHIS [13] :

1. "ambulatory epicrisis" [13] ;
2. "stationary epicrisis" [13] ;
3. "doctor's letter entitling the patient for a medical procedure or appointment with another doctor" [13] ;
4. "reply to doctor's query and letter entitling the patient for a medical procedure or appointment with another doctor" [13] ;
5. "notice of opening an ambulatory medical case" [13];
6. "notice of opening a stationary medical case" [13];
7. "notice of closing an ambulatory medical case" [13];
8. "notice of closing a stationary medical case" [13];

9. “notice on assessment of development” [13];

10. “notice of immunization” [13];

11. “notice of side effects of immunization” [13];

12. “notice of physical examination” [13];

13. “notice of counseling” [13];

14. “notice of growing” [13].

According to ENHIS (Estonian National Health Information System) not only medical documents are included to EHR. It also includes the information about “patient's employer and profession, description of work conditions, educational institution, the family situation, health habits, psychosocial background and development, mental background and development”. [13]

2. Common terminology and clinical coding systems.

EU does not require any specific terminology or coding system, allowing countries to define it on the governmental level. Fourteen EU member countries have set a legal requirement to use common health terminology or a specific coding system. By the year 2014, different members of EU have approved the terminology system on the governmental level, the most frequently used are: the International Classification of Diseases and Related Health Problems, NOMESCO, NCSP+, SNOMED Clinical Terms, etc. [12]

In case of Estonia, “EHF (Estonian Health Foundation) has developed and published the classifications, standards, and nomenclatures based on Estonian and medical terminology (Ancient Greek and Latin)” [13] , which are legalized for EHR in Estonia. [13]

3. Requirements on institutions hosting and managing EHRs

The EU law requires Member States to provide that the data controller must “implement appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access, in particular where the processing involves the transmission of data over a

network, and against all other unlawful forms of processing." [12] All the EU member countries have data protection rules, although only 15 countries have set specific rules. [12]

"Estonian law contains specific regulatory requirements on the security level of the ENHIS. The content of the required security measures is determined by detailed guidelines issued by the Ministry of Economic Affairs and Communication" [13]. The security classes of ENHIS are the following.

- "confidentiality – S2 i.e. confidential information: the use of data is only allowed to certain user groups, access is allowed in the case of justified interest." [13]

- "integrity – T3 i.e. source of information, modifying, and destroying data must always be recorded; constant control of whether the data is correct, complete and up to date" [13].

- "availability – K2 i.e. reliability 99% (around 2 hours of down time per week allowed), allowed increase in reaction time at peak capacity – minutes (1÷10)" [13].

The overall security level of data managed under ENHIS has "H" (high) classification due to the sensitivity of the data. It leads to the requirement to audit the security measures in every two years.

4. Legal requirements for encrypted data

Data encryption is one of the common ways to ensure data security. The encryption works in the following way: data is translated into the secret code and then in order to read the data a special key is needed to decrypt it. Almost all the countries of EU have the health data encrypted in some form. [12]

There is no obligation to encrypt the data by Estonian Law although the law requires the level of security to be high. Nevertheless, in practice ENHIS data is encrypted since healthcare providers forward EHRs to other providers in an encrypted form. [13]

5. Specific rules on patient's consent

The concept of having explicit patient's consent on storing one's personal data comes from the need to assure that the right to privacy is respected in case of health data. "Consent is, under Article 8(2)(a) of Directive 95/46/EC, one of the exceptions to the general rule of prohibition of the processing of special categories of data, including data concerning health; in accordance with the definition of the same Directive, the consent must be freely given, specific and informed" [12]. Only half of EU members have specific legal rules for patient's consent in relation to EHRs [12]. Although it does not mean that the explicit consent is required for EHR establishment.

In Estonia the patient's consent is not required for the creating or processing of EHR although patient can prohibit sharing EHRs in the ENHIS by submitting an application to one's healthcare provider or to the Ministry of Social Affairs (in case of the need to prohibit access to all personal data in the ENHIS) [13].

6. Specific authorization

The information, which is stored in EHR, is sensitive so the set of people, who are able to access it, is limited. According to the Data Protection Working Document: "the essential principle concerning access to an EHR must be that – apart from the patient himself – only those healthcare professionals/authorized personnel of healthcare institutions who presently are involved in the patient's treatment may have access. " [15]

Most of the EU members do not go beyond the provisions of the Directive, but some of those countries have set up specific authorization requirements for hosting and processing EHRs. [12]

In Estonia, hospitals strictly regulate, which employees and under which conditions can access EHRs and ENHIS. [13]

Although not all the requirements are listed in this paragraph, it gives an overview of the challenges that the real systems face and how many requirements they have to meet. The more detailed requirements list one can find in Directives and Laws of EU [16].

5.2 EHR Software

Although the requirements set for EHR software are high, there are nowadays some software available. In the current paragraph two open-source software systems which use a SQL DBMS will be briefly introduced.

1. GNUmed is a software for medical practice based on PostgreSQL, Python, and wxWindows. The area of the use of the application is comprehensive care departments. Although it can be used for some hospital departments it is NOT intended to be used in hospitals. The application has client-server architecture and database services are distributed. The developers of the application claim that the database is well designed – it has: "tables' normalization, data integrity, authentication and secure communication, audit trailing". [27]

2. Care2x is a hospital information system based on MySQL, Apache web server, and PHP scripting engine. This application is meant to be used in hospitals, clinics, private medical centres, etc. It has client-server architecture and uses a single database and single data format which "solves the data redundancy issues". [45]

5.3 Health Informatics Standards

"As defined by the U.S. National Library of Medicine, health informatics is the interdisciplinary study of the design, development, adoption, and application of IT-based innovations in healthcare services delivery, management, and planning." [5] The main purpose of the studies is to standardise, simplify, and unify health care processes and relevant data storage. In order to achieve the consistency, standards were developed. Standards are aimed at reusable structures which makes the systems planning and implementing easier for different organizations [6]. The most well-known EHR standards are openEHR, ISO 13606, and HL7. [34]

OpenEHR is a HI standard that describes storage, retrieval, and management of EHRs. The foundations of openEHR are clinical models and templates. Clinical models consist of archetypes. Every archetype represents a discrete specification in terms of a reference model. The reference model guarantees that the key attributes in EHR are processed and must not be addressed in each archetype. Every term in the archetype might be bound to

the terminology. Templates consist of one or more archetypes and add constraints for archetypes usage in particular settings. [39]

ISO 13606 is developed for standardizing the architecture for managing EHR data. In the standard specification, reference models and archetype models are defined. Reference model is used for representing the properties of EHR. It specifies the way to aggregate the data into complex structures following the ethical and legal requirements. Typically, it contains a set of primitive types, a set of classes which define building blocks of EHR, a set of classes to describe the context information, and a set of classes to describe demographic data. [38] Archetype model is a structured combination of reference models (entities) that is used to represent a particular clinical concept. The structure has to be defined by a domain expert since it is not prescribed. Archetype model consists of header (Meta data about the archetype), definition (the description of the clinical concept, represented by archetype, in terms of reference model entities), and ontology (binding of entities to terminology). [37]

Health level 7 refers to a set of international standards for clinical and administrative data transfer and is used by various healthcare providers. [7] The level 7 refers to the 7^{th} level of communication model of Open Systems Interconnections created by International Organization for Standardization (ISO) – application level. FHIR – (Fast Healthcare Interoperability Resources) is a standard for exchanging healthcare information electronically. The standard is developed to build a base set of resources that would satisfy the majority of use cases. FHIR modelling uses a composition approach – resources are combined and tailored to meet the specific requirements of use case's. Resources have a wide range of uses, from pure clinical content such as "care plans" and "diagnostic reports" through to pure infrastructure such as "Message Header" and "conformance statements". [9]

In the present work, it was decided to use HL7 FHIR since the reference models are designed in a manner that covers multiple use cases (use cases of the test application in particular). Another reason was the availability of sample data and the variety of resource representations (in JSON, XML, UML, etc. format) on the official web site.

6 Analysis

In this chapter I provide the analysis of the requirements to the database and applications that I will build to evaluate the two DBMSs.

6.1 The Assumptions of the System

In the present work, I will build the prototype of an e-health system. The application will not be implemented according to all the requirements of a real e-health system. Therefore, some of important aspects of a real-world e-health system (like security) were not considered during the development. The system itself is a centralized repository, where different medical institutions of Estonia upload data about the patients.

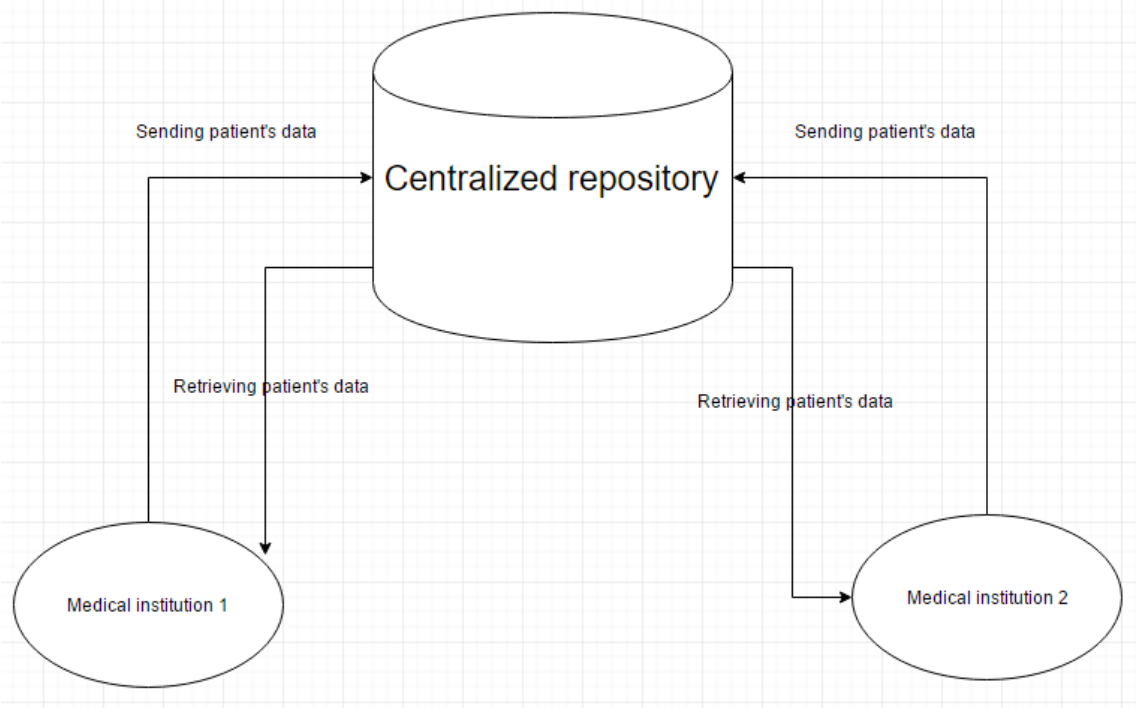

Figure 4 Centralized Repository

Before building the application, certain very strong assumptions were made.

1. I do not consider the security aspects of the system since the goal of the work is not to implement a real e-health system but to examine DBMSs, and not from the security aspects.

2. I assume that there is no login system, which means that in order to associate document creator with a certain doctor, one has to pick the doctor from the list.

3. All the doctors in Estonia are uploading the patient's info and related documents to the centralized repository.

4. The system is meant for use in Estonia which means that the unique Identifier of the person (id-code) should match the Estonian standards [10]:

    - "First digit. Gender and century identifier" [10].
    - "Second and third digits. Last two digits of the year of birth" [10].
    - "4th and 5th digit. Month of birth" [10].
    - "6th and 7th digit. Date of birth" [10].
    - "Digits 8-10. A serial number" [10].
    - "11th digit. Control number calculated using modulo 11 algorithms" [10].

5. All the doctors in Estonia use only this system. It means that the system works as the repository that must get all the documents, related to registered patients. Moreover, none of the medical institutions has their own system or keeps some data from being uploaded to the centralized repository.

6. Only the workplace of doctors is implemented in the scope of the work. It means that the only user in the system is doctor.

7. The system does not get data from the Estonian Population Register [63] .It means that doctors create the records of new patients manually.

In order to make the system more realistic I met with Mr Gunnar Piho [40], who recommended designing the database according to the international healthcare standards. After a research the HL7 standard was chosen.

6.2 Conceptual Data Model

The Figure 5 demonstrates the conceptual data model. Optional fields are marked with "N".

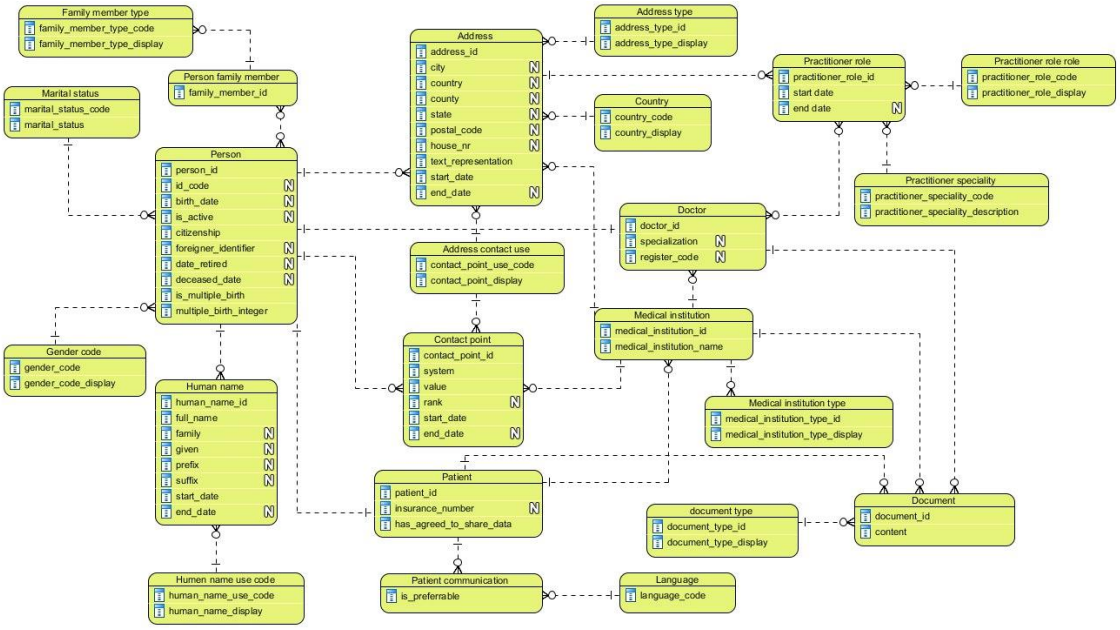

Figure 5 Conceptual data model

The conceptual data model was created based on the resources structures of HL7 FHIR standard. The schema is not following all the recommendations and requirements of the standard, since during the development process it came out that standard has too loose requirements. Therefore, I analysed the requirements of the standard and implemented a customized schema based on it.

The analysis of the standard revealed some possible problems that can be found in Appendix 7 – Possible Problems of the Standard.

6.3 Goals

The goal of the system I will implement is to provide a centralized repository that would store all the data about patients in Estonia in one place. The only actor in the current implementation of the system is a doctor. For the initial version of the system, the amount of use-cases is quite small due to the lack of time to implement wider scenarios. Currently doctor (user from now on) is able to search for documents belonging to a particular patient, search for documents with a particular type, create and update documents, and search detailed patient's information. The current database has some objects which are not updated/created/read in any use case (family member). This is done in order to widen the amount of use-cases of the system in the future work.

6.4 Use Case Model

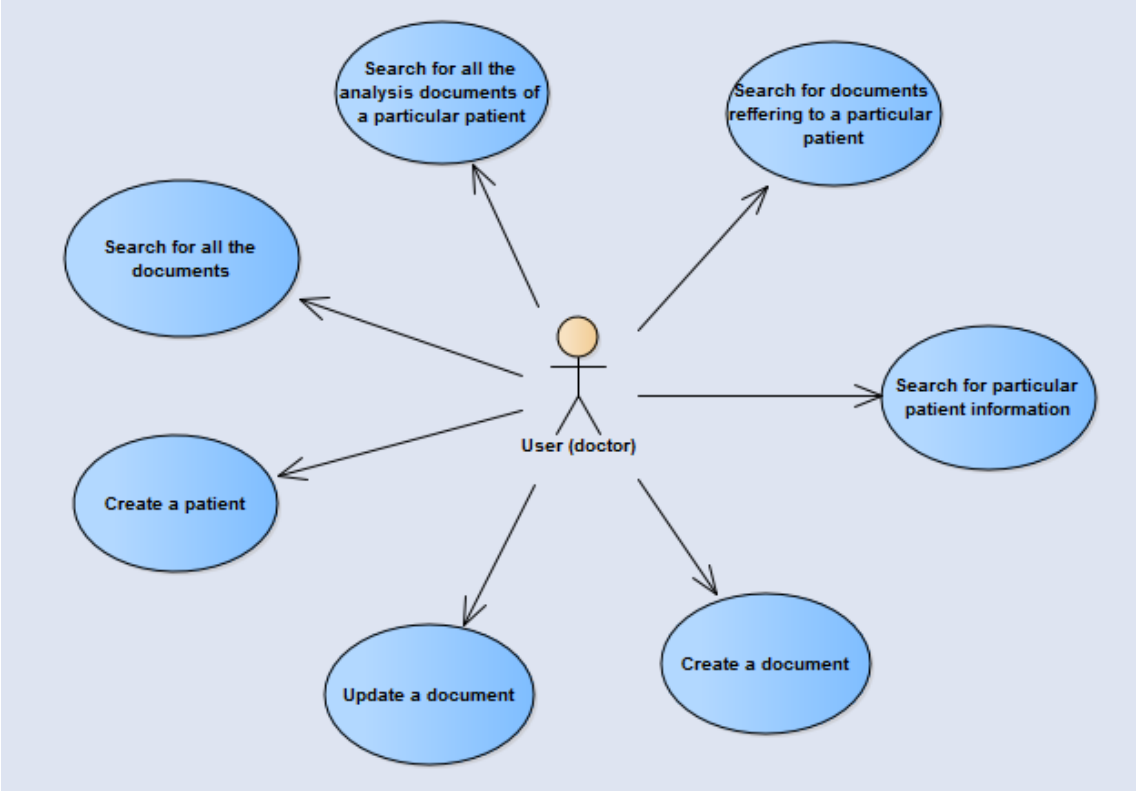

Figure 6 Use Case Model

Use case: Create a patient.

Actor: Doctor (user).

Description: The doctor picks the operation (create a new patient), being on patients list page ("localhost://patient/list"), enters the data of the patient, and submits the form. The system registers the new patient. If the patient is duplicated based on the identifier, then the system notifies the doctor about the duplication and provides the data of the already registered patient who has the same identifier. In this case new patient is not created. If the patient's data is incorrect, then the system notifies the doctor about the fields that did not pass the validation. New patient is not created.

Use case: Create a document.

Actor: Doctor (user).

Description: The doctor picks the operation (create new document), being on patients details page ("localhost://patient/read/{id}"), enters the data about the document and patient's identifier, and submits the form. The system creates the new document. The system sets the medical institution where the doctor is currently working as the owner of

the document. If the patient does not exist, then the system notifies the doctor that the patient does not exist. In this case, the new document is not created. If the document's data is incorrect, then the system notifies the doctor about the fields that did not pass the validation and the new document is not created.

Use case: Search for all the documents reporting analysis results of the particular patient.

Actors: Doctor (user).

Description: The doctor picks the operation (search for all the analysis documents), being on the patient details page ("localhost://patient/read/{id}") and submits the form. The system performs the search of documents referring to the target patient that have type "analysis report" and shows the search result as a list of documents.

Use case: Search for all the documents of the particular patient.

Actors: Doctor (user).

Description: The doctor picks the operation (search for all the documents of patient), being on the patient details page ("localhost://patient/read/{id}"). The system performs the search of documents referring to the target patient and shows the search result as a list of documents.

Use case: Search for the detailed information of a particular patient.

Actors: Doctor (user).

Description: The doctor picks the operation (search for patient detailed info) being on patients list page ("localhost://patient/list") and selects the patient. The system shows the corresponding detailed information.

Use case: Update a document.

Actor: Doctor (user).

Description: The doctor goes to document details view ("localhost://document/read/{id}") and picks the operation "edit". The system provides the modal to edit the document. Doctor enters the new data. In case the data is valid, the document is updated, otherwise an error message is shown.

Use case: Search for all the documents in the system.

Actors: Doctor (user).

Description: The doctor goes to document list view ("localhost://document/list")). The system performs the search of documents and shows the search result as a list of documents.

7 Database Physical Design based on PostgreSQL with JSONB types

Figure 7-Figure 13 present the design of PostgreSQL tables. The diagrams were created by using IntellijIdea [50] .

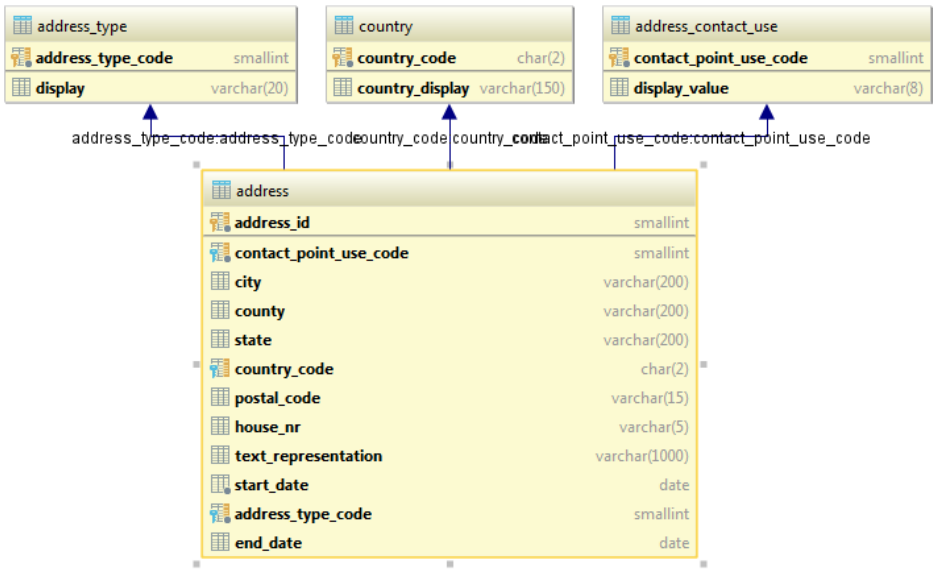

Figure 7 Address and related tables in the PostgreSQL database

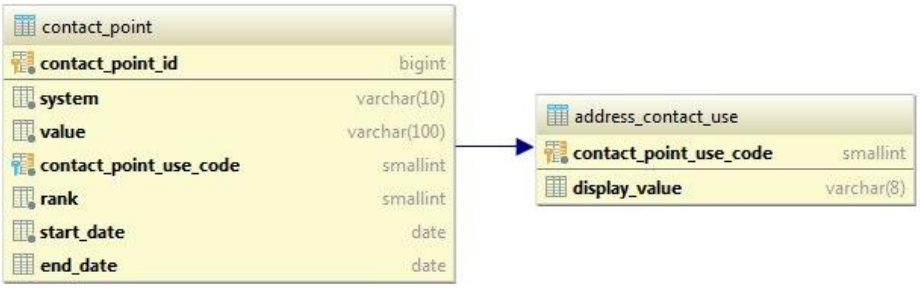

Figure 8 Contact_point and related tables in the PostgreSQL database

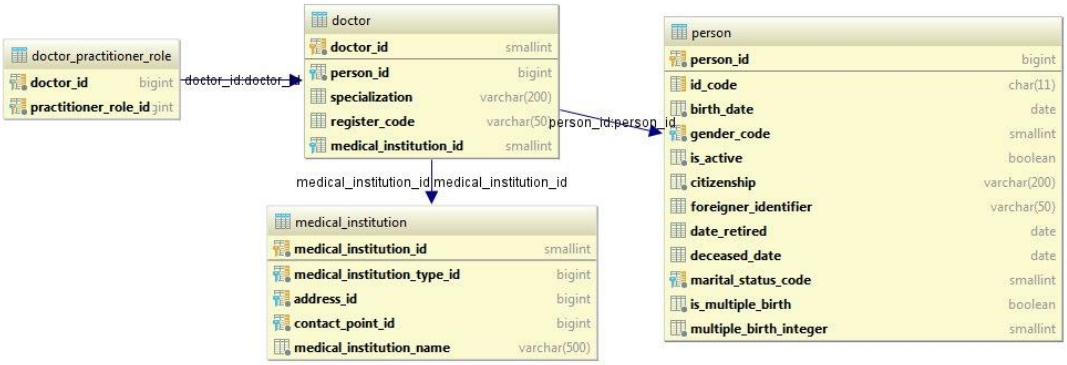

Figure 9 Doctor and related tables in the PostgreSQL database

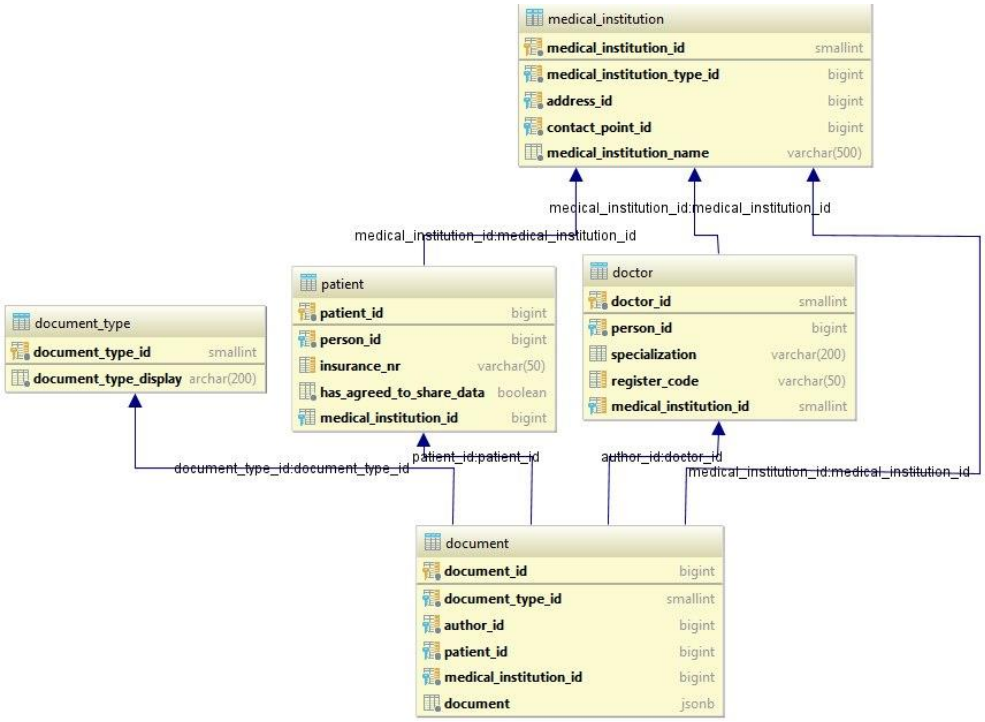

Figure 10 Document and related tables in the PostgreSQL database

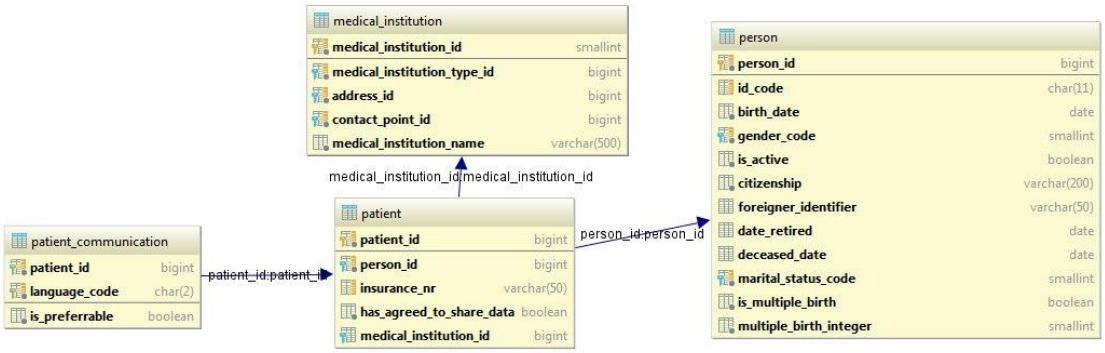

Figure 11 Patient and related tables in the PostgreSQL database

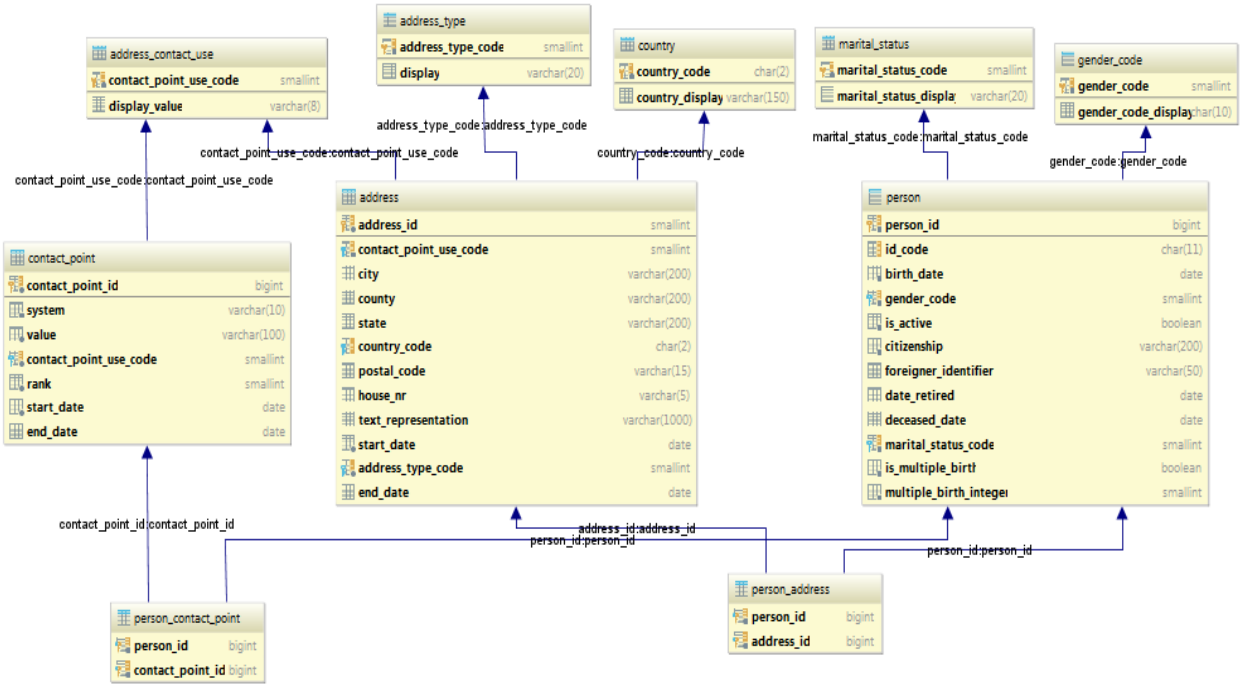

Figure 12 Person and related tables in the PostgreSQL database

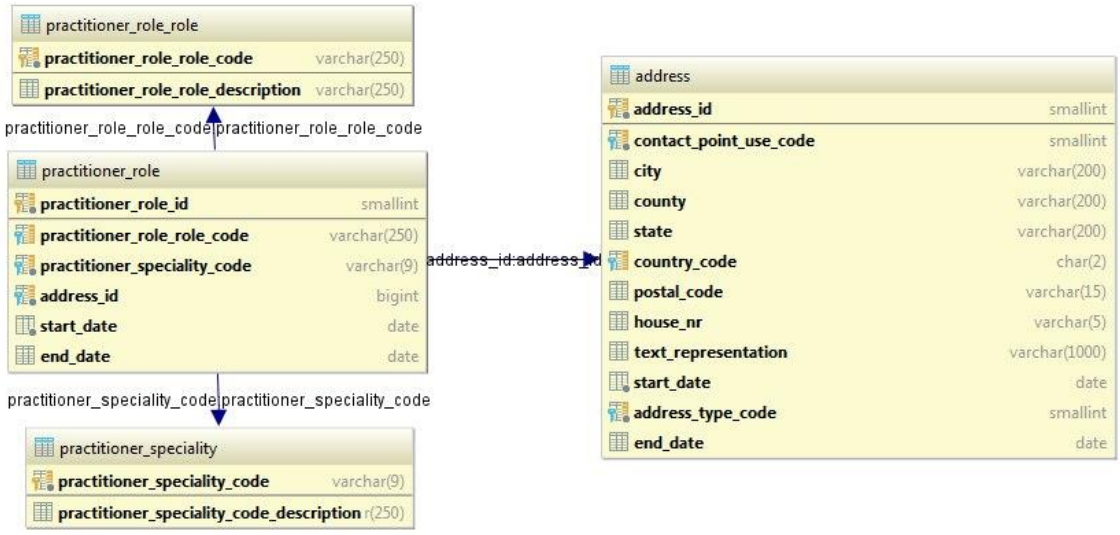

Figure 13 Practitioner and related tables in the PostgreSQL database

8 Database Physical Design based on Couchbase

Compared to PostgreSQL, Couchbase does not use the relational data model as its underlying data model. In Couchbase, the main building block of a database is document type. The documents will be stored in JSON format. Each document type has zero or more corresponding documents of this type in the database and each document represents an object.

A possibility to register relationships between documents is to register references to other documents and access the referenced documents in case of a need. This is similar to the additional normalization of tables in case of SQL databases. However, it is recommended only in certain cases. It might be useful to use the referencing approach if the "target" and the "foreign" document are not frequently accessed together. Referencing is also good when it comes to data consistency because the system has to do less work in order to modify data. Since documents reference each other and not embed each other, it is enough to update the data in one place. It is also quite beneficial for the memory usage optimization, since the amount of duplicated data is much lower that reduces the memory need. The second option to register data about relationships is to have embedded documents within a document. It is similar to the denormalization of tables in SQL databases. If the related documents are often accessed together, it makes sense to store those documents in the embedded way. [23]

In the Couchbase version of the system the embedding approach was chosen. Considering the use cases of the system, it might be beneficial to store some documents in the embedded way. For example, compared to PostgreSQL version, where "Person" is a separate table that is connected through foreign keys to the table like "HumanName", in Couchbase version additional data about persons (like human name) will be embedded within the documents about persons. The reason is that the documents are rarely modified, but often accessed in order to get complete information about a patient.

Couchbase does not provide the mechanism to set constraints at the database level. All the constraints will be enforced in the application itself.

In the test application, mandatory fields correspond to the requirements of conceptual data model Figure 5. The constraints, which are not present in the conceptual data model, are listed in the Appendix 4 – Couchbase Constraints.

On the Figure 24 Couchbase Patient document structure, the structure of the "Patient" document is demonstrated. In this document, "HumanName", "ContactPoint", "Address", "FamilyMember", and "Communication" are embedded in the "Patient" document since for the current set of use cases, none of the embedded document is accessed or modified separately.

On the Figure 14-Figure 24, the detailed JSON illustrations of embedded documents are shown. The illustrations were generated using the JSONMate web application [32] .

Colours represent data types:

- **Dark blue** – array;
- **Green** – string;
- **Orange** – object;
- **Blue** – number;
- **Light green** – Boolean;

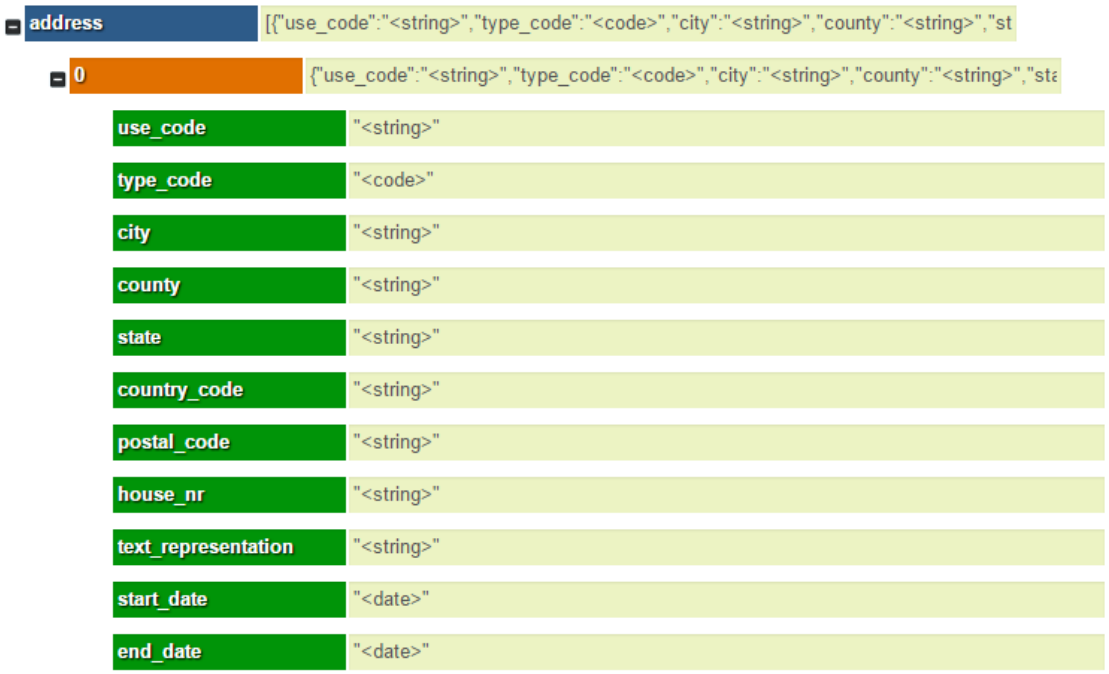

Figure 14 Couchbase Address document structure

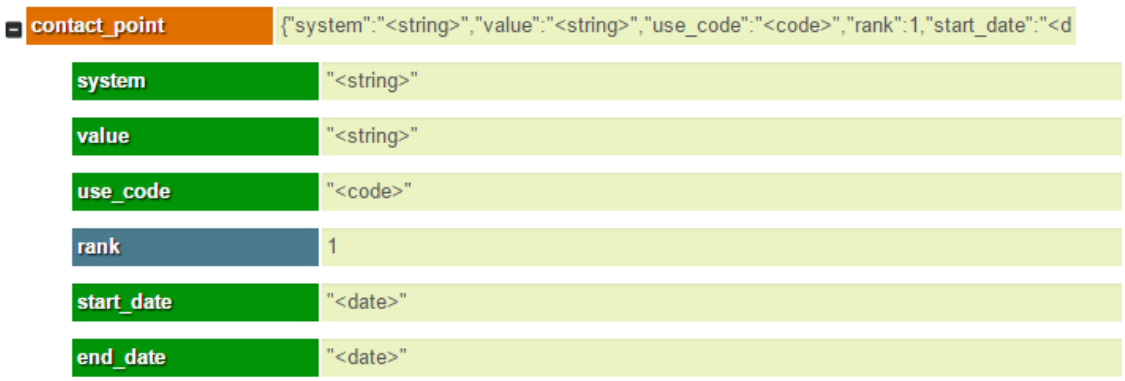

Figure 15 Couchbase ContactPoint document structure

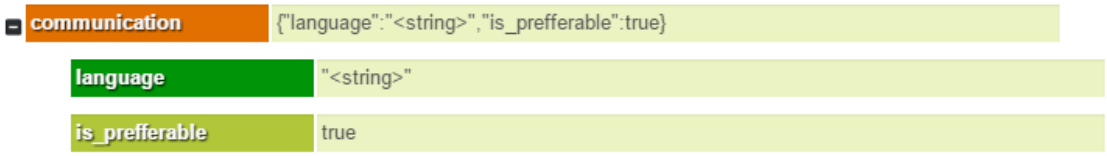

Figure 16 Couchbase Communication document structure

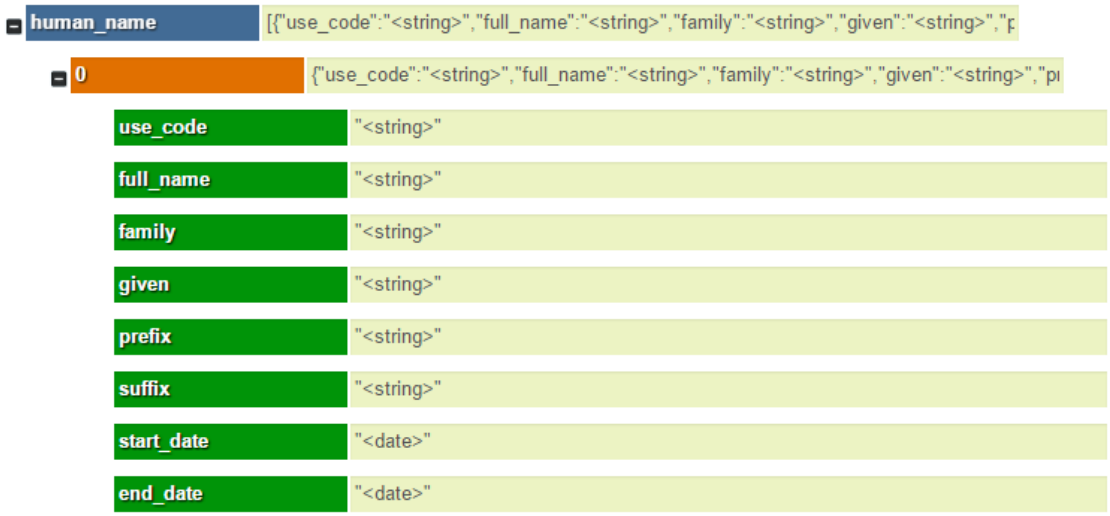

Figure 17 Couchbase HumanName document structure

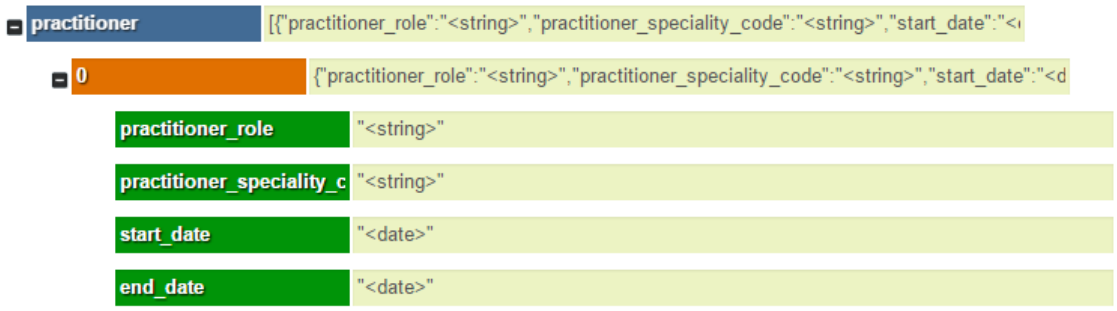

Figure 18 Couchbase Practitioner document structure

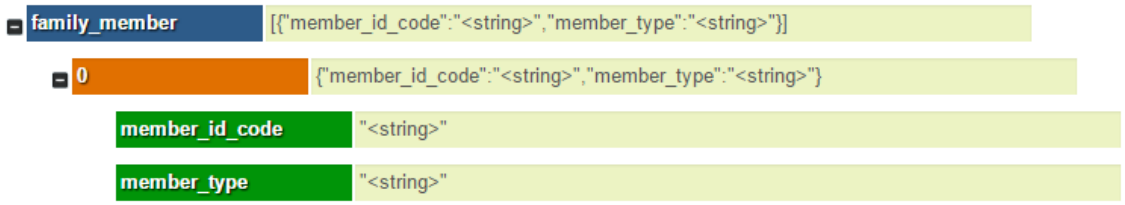

Figure 19 Couchbase FamilyMember document structure

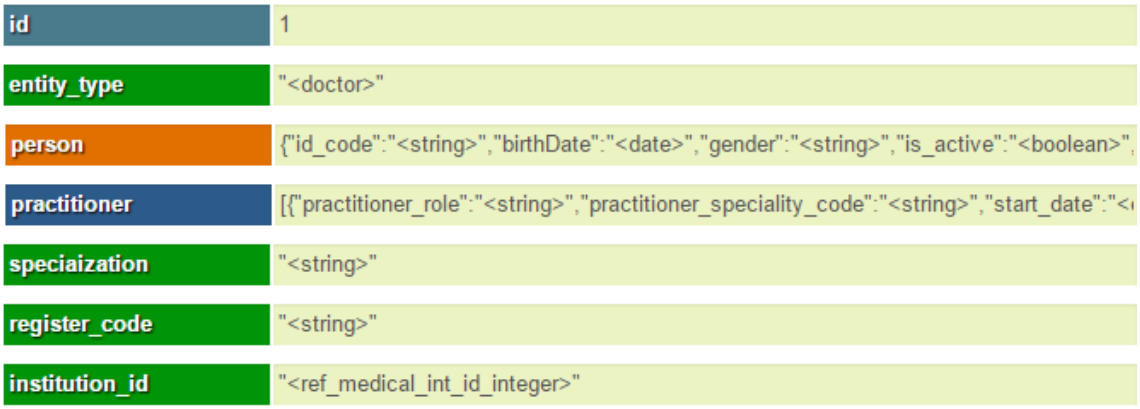

Figure 20 Couchbase Doctor document structure

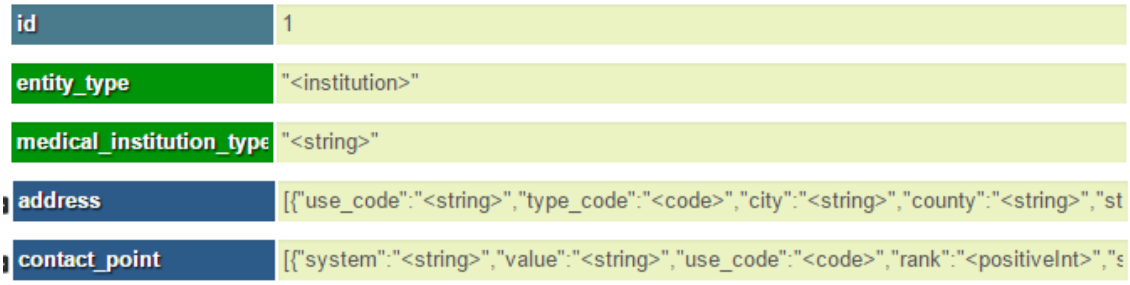

Figure 21 Couchbase MedicalInstitution document structure

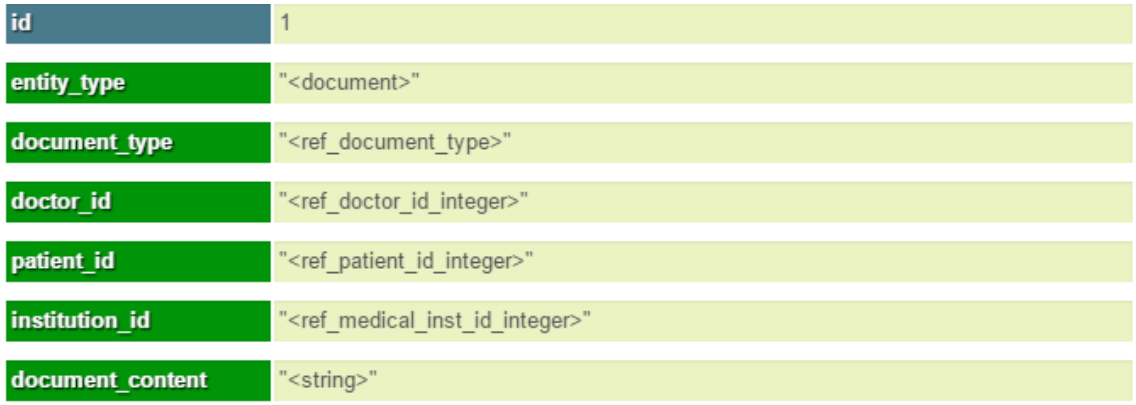

Figure 22 Couchbase Document document structure

Compared to other document types, documents with the type “Document” contain references instead of embedded documents, because it is crucial for the business logic to

have up-to-date and consistent data about the creator, owner, and target patient of the document.

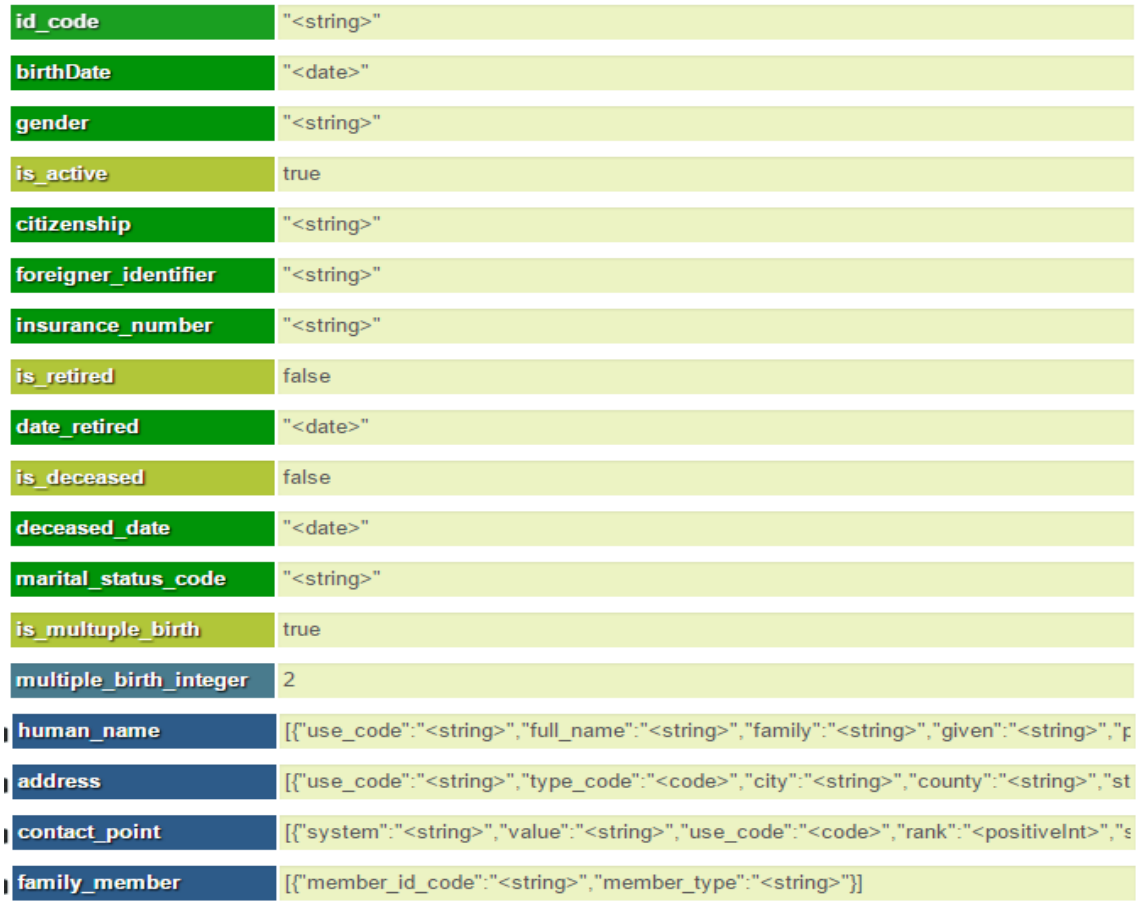

Figure 23 Couchbase Person document structure

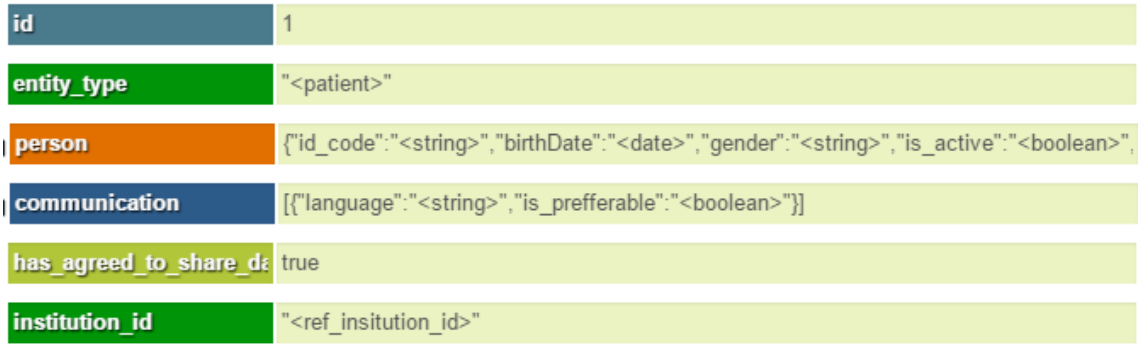

Figure 24 Couchbase Patient document structure

As it was said before, “Address”, “HumanName” and “ContactPoint” entities are embedded over the system, this decision was made because all of those entities are not reusable and have strict date range of being active.

9 Application Design

The “toy” application that is a result of the thesis is a web application implemented in Java. The architecture of the application is developed according to the best practices of n-tier architecture. In n-tier [44] architecture layers are separated logically and physically to provide better maintainability mechanisms, flexibility of the system, and improve the performance. The application is split into three layers: presentation, business, and data layer.

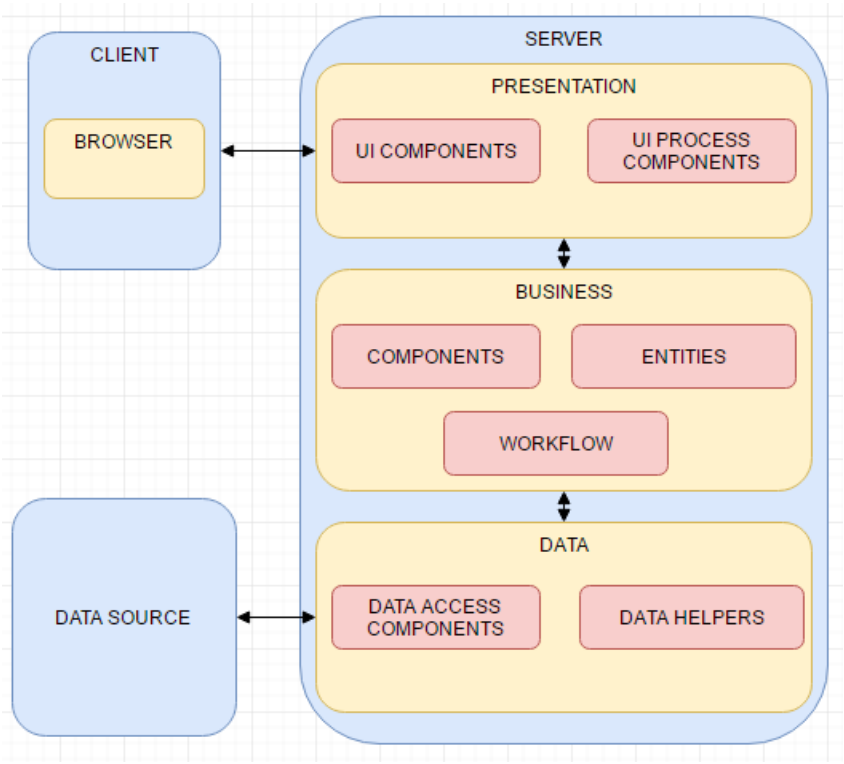

Figure 25 Application Architectural Design

On the Figure 25, the architectural design of the application is illustrated, Figure 26 explains the components.

The “presentation” layer of the application is a layer with which user interacts. It communicates with the “business” layer where all the user requests are processed

according to the business logic. The “business layer” communicates with the “data” layer, which is responsible for database operations connectivity.

Colour definition of the components in Figure 26 and Table 3:

- **component** – is a web browser which is required to operate the application;
- component – is a JavaScript component;
- component – is a component of the application written in Java;
- component – is a representation of the User Interface which includes HTML and CSS;
- component – is an external DBMS.

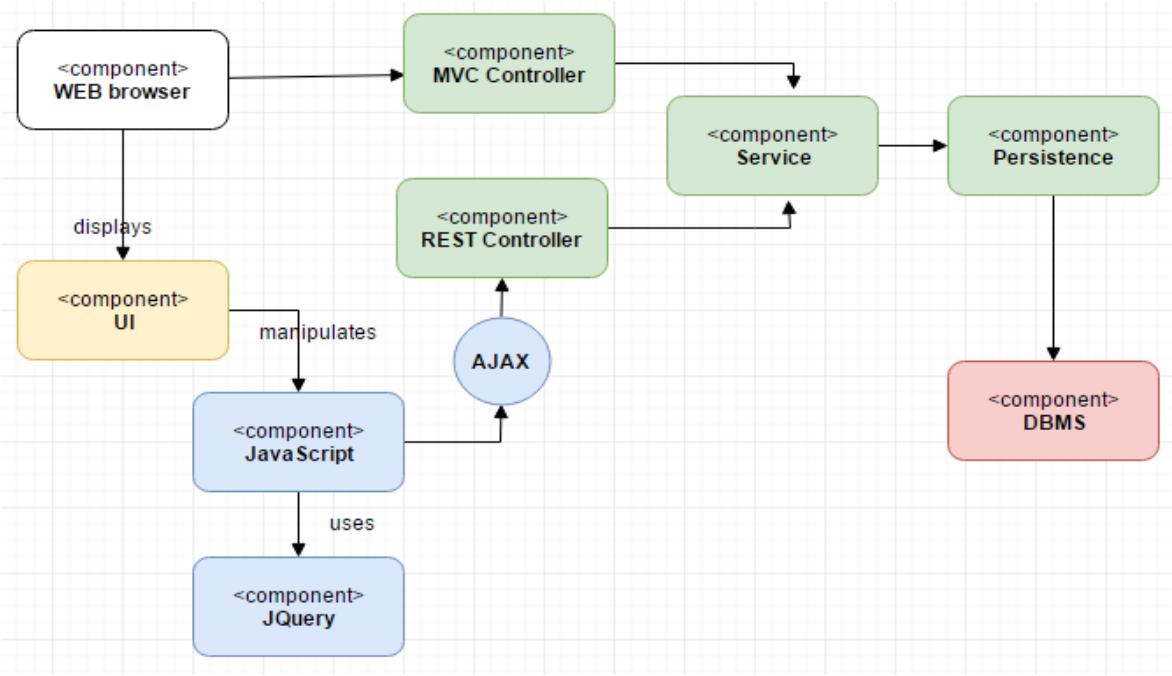

Figure 26 Application Components and their interactions.

Table 3Description of Application Components

Component	Description
Web Browser	The Web Browser is not a component of the application, but it provides user with the ability to operate the application.
JavaScript	The JavaScript component belongs to the “presentation” module. It manipulates HTML elements and sends request with AJAX calls using JQuery.
JQuery	The JQuery component belongs to the “presentation” module. It is used for Ajax interactions, HTML elements manipulations, and handing of events.
UI	The User Interface component belongs to the “presentation” module. It represents HTML pages to user, providing it the requested information and interaction controls.
MVC Controller	The MVC Controller component belongs to the “presentation” module. It handles requests, calls the “business” components to process the input data, and returns a requested view.
REST Controller	The REST Controller component belongs to the “presentation” module. It handles requests, calls the “business” components to process the input, and returns the Result object.
Service	The Service component belongs to the “business” module. It processes the input according to the business logic and communicates with the “data” module that sends and retrieves objects from the database.
Persistence	The Persistence component belongs to the “data” module. It processes the queries to the external DBMS.
DBMS	The DBMS component is used to store the queried data.

9.1 Application Development Process

The artifacts were created in the following order.

1. Based on the *Patient*, *Person*, *Practitioner*, and *Document* entities taken from HL7 FHIR resources, data structure was designed and implemented. Enterprise Architect CASE tool was used as the data-modeling tool.

2. Based on the created data structure, use cases were modeled by using Enterprise Architect. Initially the use cases were picked in order to implement different data manipulation operations, which would show the difference in the performance between the two selected DBMSs.

3. The application was created using Intellij [50] built-in application generator. The created application had group-id of "ee.ttu.thesis.DBMS(Couchbase or PostgreSQL)" and artefact-id of "medical-institution". Gradle was chosen as a builder so "build.gradle" file was generated automatically with the default configuration.

4. The "data" module was created. This module has inherited the group-id from the root project and had artefact-id "data". It is responsible for the persistence configuration: connection to the database, jpa configuration and configuration of the flyway (for PostgreSQL based application). In this module entities and repositories are stored.

5. The "business" module was created. It stores all the business logic and stores services.

6. The last created module in the application was "presentation" module. It stores all the UI components, controllers, configurations of the web application, and the Application.java class itself.

7. Every module got an automatically generated "build.gradle" file with the dependencies. Dependencies specific for each module were added to the corresponding "build.gradle" files.

8. In order to connect modules the requirement to compile "upper" module first was added to "build.gradle" files (the "business" module compiles after the "data"

module and the “presentation” module compiles after the “business” module). All modules were included to the root project in the “settings.gradle” file.

9. Corresponding components were added to modules (“repositories” and “entities” to “data”, “services” and “service implementations” to “business”, “controllers” and “views” to “presentation”) based on the architectural demand and tables (documents) of DBMS.

10. HTML files were created based on the preselected use cases.

I estimate that it took for me approximately 50 hours to build the application for the PostgreSQL database and 70 hours to build the application for the Couchbase database.

9.2 Physical Design of the Application

In the current paragraph the common aspects of the physical design of both applications will be described.

- Gragle 3.1 builder. Initially it was planned to use Maven but I decided to try the newer option – Gradle since it is gaining popularity over the last few years.

- Spring-boot 1.5.2 RELEASE framework. Spring is a popular Java-based framework used to build web applications. Spring provides to its users several ways of configuring beans (XML, JavaConfig, Annotations), which gives the freedom to choose the approach which is easier for the particular developer, as well as simplifies the integration with other frameworks

- Thymeleaf 1.5.2 RELEASE. It is a server-side Java template engine, which has more powerful and readable syntax and is easier to be integrated with Spring system.

9.2.1 PostgreSQL Application

- Hibernate 5.0.12. Hibernate is an implementation of Java Persistence API, which allows developers to create classes following Object-oriented practices. It is also known for being scalable and provides a possibility to improve the performance.

- FlyWay 3.0 is an open-source tool for migrating the database scripts, which helps to track the script versions.

- PostgreSQL server 9.6.

9.2.2 Couchbase Application

- Couchbase server 4.5.1 community.

9.2.3 Patterns used in the Application

1. Builder pattern. In order to simplify objects creation in the PostgreSQL application the Builder pattern is used.

2. MVC pattern – a pattern used for separating application's concerns. Model is a Java POJO object, View is a visualization of the object and Controller updates the view, controls the data flow into the model object, and keeps model and view separately.

3. Data Transfer Object pattern is used to optimize the load on the system by reducing the number of entities to be selected (all the information needed to be shown on the page is stored in one object).

4. AJAX pattern. Front-end pattern used to minimize the traffic between front- and back- end. It improves user experience since provides the possibility to load additional data "on demand". It reduces the time spent on page load.

10 Experiments, Results, and their Analysis

In this chapter, I present the actual data manipulation operations that I used for the performance experiments, present the results of the experiments as well analyze the results. I also evaluate the complexity of creating an application that uses the implemented databases. I also present the results of a small data-schema change experiment.

10.1 Performance

Both DBMSs were tested on the same machine. The machine has the following characteristics:

Processor: Intel Core 4th Gen i7-4700MQ (6M Cache, 3.4 GHz),

SSD: 256GB SSD,

Operation system: Windows® 7 Pro (64bit) ENG.

In order to perform the experiment, I generated the test data for PostgreSQL DBMS with the following sizes.

- The data size for the first experiment included 1 000 000 rows in *document* table, 5000 rows in *patient* table, 1000 rows in *doctor* table, and 100 rows in *medical_institution* table.
- The data size for the second experiment included 500 000 rows in *document* table, the data in other tables was not changed.
- The size for the third experiment included 250 000 rows in *document* table, the data in other tables was not changed.

The initial test data set was generated by using online tool Mockaroo [48] . Initial data set included 1000 addresses, 6000 contact points, 10 000 human names, 100 medical institutions, 5000 patients, 4000 persons, 3000 connections between persons and addresses, 4000 connections between persons and contact points, 1000 doctors, 10 000 documents, 5 000 patients, and some small amount of other data. After that, the number of documents was increased to 1 000 000 with the script.

In order to have identical data in both databases, I created a data migration application. This application is connected to both databases and works by the following principle: it selects data about all the objects from the PostgrSQL database, serializes every object into JSON, and saves the result to Couchbase database. This application is not meant to be a part of the current work, it is just a utility used for data transfer. Therefore, the code is not optimized there.

In order to measure performance I will execute each statement five times and calculate the geometric mean of the results. The summary tables Table 4-Table 7 present the means in case of different data sizes. I will use the geometrical mean because previous executions make the system warm and thus they have an impact on each other in terms of performance. [64] Another possibility is to use median. Arithmetic average is not good approach because it is influenced too much by extreme cases.

The DBMSs will be tested on different data sizes in order to understand as to whether the time spent on data manipulation operation execution influences the performance in the linear manner or not. I will calculate Pearson correlation coefficient value in order to find out the relationship between data size of *document* table/document type and time of query execution. Since I want to save time on the calculations, I will use an online service for calculating Pearson correlation coefficient [47] .

In case of PostgreSQL and Couchbase, I will use SQL statements and N1QL statements respectively, to perform the data manipulation operations.

In order to measure PostgreSQL performance, I will use the command *explain analyze*. *Explain* is used to "show the execution plan of a statement" [51] , *analyze* is used to "carry out the command and show the actual run times"[51] .

In case of Couchbase, since I could not find any tool that would analyze the time of a single query execution, I will use network statistics. When N1QL query is executed, the server sends response that contains the statistics information. I used "metrics.executionTime", which reports the query execution time, which does not include the time that request spends in the queue before processing . When the query result was too large and it was not possible to get the precise execution time, I used graphical user interface, where one can find a rounded time of query execution.

10.1.1 Experiment 1 – Select all Documents

Description: In this experiment, I retrieved the data needed for the documents list view (*search from all documents* use case). I requested all the documents that are available in the database. Since I do not need all the detailed information about every document on the list view, I requested only certain values: id, document type, medical institution name, author name, and patient name. In both DBMSs, document has references to foreign objects. It means that in order to retrieve the information needed for the view, I had to add join statements to the query.

Results: see Table 4.

Table 4Results of the Experiment 1

	PostgreSQL	Couchbase
One million *Documents*	7046.889(ms) (7.04 s)	-
One million *Documents* (a simplified query)	497.240 (ms) (0.5 s)	44.692(s)
500 000 *Documents* (a simplified query)	295.186 (ms) (0.3 s)	16.057 (s)
250 000 *Documents* (a simplified query)	137.238 (ms) (0.14 s)	8.570 (s)

Comments:

1. PostgreSQL:

   The query for conducting this experiment in case of one million *Documents* can be found in Appendix 2 - Experiment 1 – PostgreSQL query 1.1. This query includes invocation of the aggregation function – *array_agg()*. It produces a single result (value) from a set of values. I use this function because person might have multiple active names. Since the generated data is random and there are no requirements about priority of human name types, I select the first element of the

set. Aggregate functions require grouping non-aggregated fields, therefore all the other select fields of the query appear in *group by* clause.

2. Couchbase:

   The experiment was not possible to conduct in case of one million *Documents* because it took too long to wait until the query was executed. Thus, I had to cancel the query execution. The query that I tried to execute can be found in Appendix 3 - Experiment 1 – Couchbase query 1.1. I could not find the exact problem, but the assumption is that indexes do not apply correctly in case of complicated query with multiple joins. In order to improve the performance, one has to cover query with index – "index should include all the fields that are specified in the query" [52] . During the investigation, I discovered that it is a known problem that was reported for Couchbase Community 4.5.1 in March 2017 on Couchbase Forum [53] .

   In order to proceed with the experiment, I simplified the query. The simplified query can be found in Appendix 3 – Experiment 1 – Couchbase query 1.2. The execution time of the simplified query is:

   Since one of the goals was to compare the performance and the requirement was to test both DBMSs with the same tasks, I decided to test the performance of PostgreSQL with the simplified query. The complicated version of query was not used in case of other data sizes. The simplified query can be found in Appendix 2 - Experiment 1 – PostgreSQL query 1.2.

10.1.2 Experiment 2 – Select Documents with Search Parameters

Description: In this experiment, I retrieved all the documents that are related to a certain patient and have a certain type. I retrieved document id, name of its author, name of the medical institution, and document content. The query has two restrictions: *patient_id* and *document_type_id* are restricted to the search values and jsonb content of the document should contain key '*resourceType'* with value '*Observation'*.

Results: see Table 5.

Table 5Results of the Experiment 2

	PostgreSQL	Couchbase
One million *Documents*	4.965(ms) (0.004965 s)	
One million *Documents* (a simplified query)	0.472 (ms) (0.000472 s)	12.255 (s)
500 000 *Documents* (a simplified query)	0.449 (ms) (0.000449 s)	6.152 (s)
250 000 *Documents* (a simplified query)	0.354 (ms) (0.000354 s)	2.909 (s)

Comments:

1. PostgreSQL:

   The query for conducting this experiment in case of one million *Documents* can be found in Appendix 2 - Experiment 2 – PostgreSQL query 2.1.

2. Couchbase:

   The attempt to run the query for this experiment (Appendix 3 - Experiment 2 – Couchbase query 2.1) showed the same result as in the Experiment 1. It was not possible to execute the query. Therefore, I tested the simplified query. The simplified query can be found in Appendix 3 - Experiment 2 – Couchbase query 2.2. The simplified query for PostgreSQL is in (Appendix 2 - Experiment 2 – PostgreSQL query 1.2).

10.1.3 Experiment 3 – Update a Document

Description: In this experiment, I updated a certain document. In the scope of the experiment, the document content and document type will be changed. The "update" date remains the same for every query in scope of the present experiment.

Results: see Table 6.

Table 6Results of the Experiment 3

	PostgreSQL	Couchbase
One million *Documents*	0.408 (ms)	1.516 (ms)
500 000 *Documents*	0.108 (ms)	1.516 (ms)
250 000 *Documents*	0.166 (ms)	1.319 (ms)

Comments:

1. PostgreSQL:

   The statement for conducting this experiment can be found in Appendix 2 - Experiment 3 – PostgreSQL query 3.1.

2. Couchbase:

   The statement can be found in Appendix 3 - Experiment 3 – Couchbase query 3.1.

10.1.4 Experiment 4 – Create a document

Description: In this experiment, I created a new document. I set the values of *patient_id*, *author_id*, *document_type_id*, *medical_institution_id*, and content of the document. As a content of a document, I took a JSON example of the document from the HL7 FHIR resources. The rest of the values I took from the existing objects of the database. Since I had to run the statement several times, in order to get the mean time of execution, I changed *document_id* every time I run new query.

Results: see Table 7.

Table 7Results of the Experiment 4

	PostgreSQL	Couchbase
One million *Documents*	1.801 (ms)	1.776 (ms)
500 000 *Documents*	0.393 (ms)	1.319 (ms)
250 000 *Documents*	0.290 (ms)	1.516 (ms)

Comments:

1. PostgreSQL:

   The query for conducting this experiment can be found in Appendix 2 - Experiment 4 – PostgreSQL query 4.1.

2. Couchbase:

   The query can be found in Appendix 3 - Experiment 4 – Couchbase query 4.1.

10.1.5 Dependency of the Data Size

In order to understand the dependency between data size and query execution time (two variables), I calculated Pearson correlation coefficient by using the online calculator [54] . “It has a value between +1 and −1, where 1 is total positive linear correlation, 0 is no linear correlation, and −1 is total negative linear correlation.” [65] For each DBMS and experiment I calculated the dependency based on three sets of variable values: three data sizes and three average geometrical means on execution times. I acknowledge that for the more precise results the sets should be bigger. The results are in Table 8. Because all the values are quite near to 1 it means that there is a strong positive linear correlation between the data size and decrease of performance of data manipulation operations.

Table 8Pearson coefficient values

	PostgreSQL	Couchbase
Experiment 1	0.9929	0.9908
Experiment 2	0.8634	0.9999
Experiment 3	0.8694	0.7559
Experiment 4	0.9631	0.7126

10.1.6 Analysis of the Results

Although the initial expectation was that Couchbase will show significantly better performance compared to PostgreSQL in case of data growth, the experiment showed that PostgreSQL works more efficiently with big data than Couchbase. The performance of the Couchbase was constantly lower than PostgreSQL in case of all the tested operations.

The assumption about the root problem is that indexes do not behave as expected. Developers claim that the queries can be slow when the corresponding data is not covered by indexes [55] . In my experiments, I tried to cover the queried data with indexes. This can be checked with the “explain” command, which provides the execution plan of the query. As an example, the output of the “explain” command run with the

Experiment 2 – Couchbase query 2.2 is shown on the Figure 27. The list of all the indexes that I set for my experiment can be found in Appendix 5 – Couchbase Indexes.

Another possible problem is the data structure and, consequently, the number of joins in the query (see Experiment 1 – Couchbase query 1.1). That assumption comes from the fact that it was still possible to receive the response from the database having a simple structured query (see

Experiment 1 – Couchbase query 1.2).

Figure 27 Couchbase Explain Query Output

10.2 Integration

One of the goals of the current work was to compare the complexity of building Java web application over the selected DBMSs. The current paragraph will describe the problems that appeared during the integration with both DBMSs. In order to minimise the number of issues, I tried to follow the official tutorials and documentation.

10.2.1 Integration with PostgreSQL

During the development of PostgreSQL based application, I did not face any significant problems. To be fair, I have to say that I have been developing applications based on PostgreSQL DBMS before. Thus, I was familiar with the majority of the issues I faced and the ways to fix those. Otherwise, PostgreSQL developers provide detailed documentation, which is enough in case of need. In order to minimise manual query writing and objects mapping, I used Hibernate ORM framework. The problem I faced at that step is that some of the aggregate functions supported by PostgreSQL (for example

array_agg) are not supported by Hibernate, but those issues can be solved by rewriting the query in HQL or by creating a native SQL query. I also used Flyway for scripts migration, which helped me to have versioned scripts and track the changes not changing the initial script. All in all, the building of the Java web application based on PostgreSQL DBMS went smoothly.

10.2.2 Integration with Couchbase

The development of the Couchbase based application was even more challenging than I assumed. Before proceeding with the development, I had to install Couchbase Server on my personal computer. The first version I tried to install was 4.6.1 Enterprise edition but after the installation, Couchbase Server was not up. After continuous attempts to put the server up, I had to turn to the technical support who recommended me to downgrade the version to Couchbase 4.5.1 Community edition. Following the recommendation, I uninstalled 4.6.1 Enterprise edition version and installed 4.5.1.Community edition which, unfortunately, did not fix the problem. After researching, it came out that Couchbase Server has compatibility problems with Windows 10 that is running on my computer. Thus, I had to switch to Windows 7 in order to make it work.

The second issue I faced was during the configuration of the application. In the WebApplicationConfig.java class I used Spring annotation @EnableWebMvc, which imports Spring MVC configuration from WebMvcConfigurationSupport.java. I got the exception that was reporting about conflicting beans. It took some time to go through forums and find out the problem.

After adding the components with corresponding annotations, I still could not build the application since repository beans were not detected. This problem was easily fixed after researching on @EnableCouchbaseRepository annotation in the official documentation [59] .

One of the positive things I discovered is that it is possible to use N1QL for querying, since Couchbase supports this query language. The syntax of N1QL is similar to SQL. Therefore, it was quite easy to create queries.

To be objective, I have to say that previously I did not have any experience with Couchbase. The number of challenges I faced was higher compared to PostgreSQL due to my lack of experience.

10.3 Schema Modification

In this section I explain the schema modification experiment.

10.3.1 Change 1

Idea: I add an optional field "medical_institution_id" to the "patient" referencing the "medical_institution". In real life, most of the patients have a family doctor, who belongs to a certain medical institution.

Changes in the database: PostgreSQL application required a migration script to create a new column and add a referential constraint that points to the foreign table. The script can be found in Appendix 1 – PostgreSQL Schema Change Script. Lines 4-7 correspond to the changes needed for the current experiment.

Changes in the application: Both Couchbase and PostgreSQL application required same changes on the level of the application.

- In Couchbase version, I added field of type String called "medicalInstitutionName" to both Patient.java and PatientDTO.java.
- In PostgreSQL version, I added the field of MedicalInstitution.java to Patient.java and mapped it with Hibernate. Since PatientDTO.java stores only values for the fields that are needed on certain views, I added there "medicalInstitutionName" field of String type.

Both applications needed html and javascript changes, as well as rewriting queries to select data that has the new structure.

I estimate that it took for me 3 hours to modify database and application in case of PostgreSQL and 3 hours to modify database and application in case of Couchbase.

10.3.2 Change 2

Idea: I removed the fields “is_retired” and “is_deceased” from person. Since there are fields “date_retired” and “date_deceased” that are not mandatory, “is_retired” and “is_deceased” fields might be dropped not influencing the business logic. Such modification improves data structure because there is now less duplication.

Changes in the database: PostgreSQL application required migration script to drop the columns. The script can be found in Appendix 1 – PostgreSQL Schema Change Script, lines 10 and 11. The consequences of dropping the columns are that once we drop it, we cannot take the corresponding data from the current version of the database. It means that the data is lost.

Changes in the application: The application level changes for both systems were the same – I rewrote the queries to select/update data and removed corresponding fields from Person.java classes and htmls.

I estimate that it took for me 3 hours to modify database and application in case of PostgreSQL and 3 hours to modify database and application in case of Couchbase.

10.3.3 Results

The initial expectation was that the PostgreSQL-based system will be less tolerant to data-schema change than the Couchbase-system, since the latter does not require any explicit schema declaration. It was also expected that the volume of work the developer should make to process the changes will be higher in PostgreSQL-based application.

The experiment showed that the main difference in processing data schema change is that PostgreSQL DBMS needs the migration script to update the schema and Couchbase does not. The changes done on the application level were similar. Although it is worth to mention that if the first experiment required medical institution to be a mandatory field, then it would require some default value. It means that I would have to create some non-existing medical institution and reference it from the patients.

11 Summary

The present work had two goals: compare the performance of SQL and NoSQL DBMS in case of data manipulation operations and complexity of building Java web application over those DBMSs. In order to achieve the goals, a Java web application had to be created and DBMSs had to be tested with a certain set of data manipulation statements with three different data sizes.

In order to fulfill the goals, I designed and implemented two databases – one for PostgreSQL and one for Couchbase. For each database, I built a web application that uses it. The domain of the databases and applications is e-health. However, the resulting systems are not real e-health systems due to very strong and in real life unrealistic assumptions. Nevertheless, I am interested in the domain, investigated it in the present work, and tried to make my implemented systems more realistic. I even found some possible problems of HL7 FHIR standard and suggested improvemehnts.

Experiments were based on PostgreSQL (9.6) and Couchbase (Community edition 4.5.1).

The goals of the present work were fulfilled, although some of the actual results did not meet the initial expectations.

An initial expectation was that Couchbase will show better performance with large data, although the experiment showed that I was mistaken. Couchbase performance is much lower compared to PostgreSQL performance on any data size and any operation. During the experiment, I discovered problems related to the indexing in Couchbase. Although the present work does not present a solution due to lack of time, it might be a topic for future studies.

My second expectation was that PostgreSQL and application that is built on top of it would be less tolerant to data schema change. It was not proven by the experiment. There was almost no extra work in case of PostgreSQL compared to Couchbase apart from the

need to write a migration script. I acknowledge that the experiment is small and it might have influenced the results.

I expected that building a Java web application over Couchbase might be more difficult compared to PostgreSQL. In this case expectations were met and I faced some obstacles during the development, which were successfully solved.

In order to improve the performance of Couchbase, the future work might include extending the experiment by changing from one-node to multiple-nodes architecture. This could provide better availability and faster data-access.

Based on the research I got an opinion that for the task of the experiment (creating a centralized e-health system) PostgreSQL 9.6 is better than Couchbase 4.5.1.

Another experiment that might be conducted in the future work, is implementing PostgreSQL data-schema with columns of JSONB (or JSON) types only. This will allow PostgreSQL to have almost the same data structure as the current Couchbase database. It would be interesting to compare performance in this case. Another interesting practical work would be to have much more extensive data structure changes.

References

[1] Luke P. Issaac, “SQL vs NoSQL Database Differences explained with few Example DB” 14 January 2014 [Online] Available http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/ [Accessed 28.10.2016]

[2] “ACID properties of Sqlserver 2005” , 4 March 2009. [Online] Available https://hassanszone.wordpress.com/2009/03/04/acid-properties-of-sqlserver-2005/ [Accessed 28 October 2016]

[3] Liam Mclennan”Postgresql as NoSQL Document Store” [Online]Available http://withouttheloop.com/articles/2014-09-30-postgresql-nosql/ [Accessed 28 October 2016]

[4] Rob Conery “Document Storage Gymnastics with Postgres”, 1 March 2015 [Online] Available http://rob.conery.io/2015/03/01/document-storage-gymnastics-in-postgres/ [Accessed 28 October 2016]

[5] “Health informatics defined”, 07 January, 2014 [Online] Available http://www.himss.org/health-informatics-defined [Accessed 10 January 2017]

[6] “Health Informatics Standards - Health Information Systems and Processes”, [Online] Available https://wiki.ecdc.europa.eu/fem/w/wiki/health-informatics-standards-health-information-systems-and-processes [Accessed 30 January 2017]

[7] “Health level 7”, 02 November 2016 [Online] Available https://en.wikipedia.org/wiki/Health_Level_7 [Accessed 10 January 2017]

[8] Steven Saslow “5 Information Technology Challenges Faced by Healthcare Organizations”, 5 April 2016 [Online] Available https://www.itgct.com/5-information-technology-challenges-faced-by-healthcare-organizations/ [Accessed 10 January 2017]

[9] “FHIR Overview”, [Online] Available http://www.hl7.org/fhir/overview.html [Accessed 12 January 2017]

[10] Andres Kütt “On Estonian id-code” , 11 December 2014 [Online] Available https://www.ria.ee/riigiarhitektuur/blog/2014/12/11/on-estonian-id-code/ [Accessed 24 February 2017]

[11] “Electronic Health Record”, [Online] Available https://en.wikipedia.org/wiki/Electronic_health_record [Accessed 27 February 2017]

[12] “Overview of the national laws on electronic health records in the EU Member States and their interaction with the provision of cross-border eHealth services. Final report and recommendations”, 23 July 2014 [Online] Available http://ec.europa.eu/health//sites/health/files/ehealth/docs/laws_report_recommendations_en.pdf [Accessed 27 February 2017]

[13] “Overview of the national laws on electronic health records in the EU Member States National Report for the Republic of Estonia”, 13 May 2014 [Online] Available https://ec.europa.eu/health/sites/health/files/ehealth/docs/laws_estonia_en.pdf [Accessed 27 February 2017]

[14] “EU Directive 95/46/EC – The Data Protection Directive, Chapter I – General Provisions” [Online] Available https://www.dataprotection.ie/docs/EU-Directive-95-46-EC-Chapter-1/92.htm [Accessed 27 February 2017]

[15] “Working Document on the processing of personal data relating to health in electronic health records (EHR)” , Adopted on 15 February 2007 [Online] Available http://www.dataprotection.ro/servlet/ViewDocument?id=228 [Accessed] 28 February 2017

[16] “Regulations, Directives and other acts” [Online] Available http://europa.eu/european-union/eu-law/legal-acts_en [Accessed 01 March 2017]

[17] “Resource – Patient” [Online] Available https://www.hl7.org/fhir/patient.html#Patient [Accessed 30 October 2016]

[18] “Data type – address” [Online] Available https://www.hl7.org/fhir/datatypes.html#Address [Accessed 30 October 2016]

[19] “Data type – Human name” [Online] Available https://www.hl7.org/fhir/datatypes.html#HumanName [Accessed 30 October 2016]

[20] “Data type – ContactPoint” [Online] Available https://www.hl7.org/fhir/datatypes.html#ContactPoint [Accessed 30 October 2016]

[21] Renzo Cristian Bertuzzi Leonelli School of Computer Science, A dissertation submitted to the University of Manchester for the degree of Master of Science in the Faculty of Engineering and Physical Sciences “ENHANCING A DECISION SUPPORT TOOL WITH SENSITIVITY ANALYSIS ” [Online] Available https://studentnet.cs.manchester.ac.uk/resources/library/thesis_abstracts/MSc12/FullText/BertuzziLeonelli-RenzoCristian-fulltext.pdf [Accessed 18 March 2017]

[22] “Software and Documentation” [Online] Available https://www.mongodb.com/community/licensing [Accessed 20 March 2017]

[23] Documentation to Couchbase Server 4.5 [Online] Available https://developer.couchbase.com/documentation/server/current/introduction/intro.html [Accessed 23 March 2017]

[24] “The MongoDB 3.4 Manual” [Online] Available https://docs.mongodb.com/manual/ [Accessed 23 March 2017]

[25] “Developer Guide (API Version 2012-08-10)” [Online] Available http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html [Accessed 23 March 2017]

[26] Yoshinori Matsunobu “MyRocks: A space- and write-optimized MySQL database” [Online] Available https://code.facebook.com/posts/190251048047090/myrocks-a-space-and-write-optimized-mysql-database/ [Accessed 04 April 2017]

[27] [Online] Available http://www.gnumed.de/promotion/GNUmed-introducion.pdf

[28] “Resource – Person” [Online] Available https://www.hl7.org/fhir/person.html [Accessed 30 October 2016]

[29] “Resource – Practitioner” [Online] Available https://www.hl7.org/fhir/practitioner.html [Accessed 30 October 2016]

[30] “Resource – Patient” [Online] Available https://www.hl7.org/fhir/patient.html#Patient [Accessed 30 October 2016]

[31] “Data type – Codeable Concept” [Online] Available https://www.hl7.org/fhir/datatypes.html#CodeableConcept [Accessed 30 October 2016]

[32] JSONMate tool [Online] Available http://jsonmate.com [Accessed 02 March 2017]

[33] Clarence J M Tauro, Aravindh S, Shreeharsha A.B, “Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases”, June 2012 [Online] Available http://s3.amazonaws.com/academia.edu.documents/30869512/pxc3880336.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1493746641&Signature=oEJterKbmU9AxGyvjXPcvY7a8T4%3D&response-content-disposition=inline%3B%20filename%3DComparative_Study_of_the_New_Generation.pdf [Accessed 27 April 2017]

[34] G. Kopanitsa1,3; C. Hildebrand1; J. Stausberg2; K. H. Englmeier1 “Visualization of Medical Data Based on EHR Standards” 2013, [Online] Accessed [02 May 2017]

[35] “OpenEHR” [Online] Available http://www.openehr.org/what_is_openehr [Accessed 02 May 2017]

[36] “ISO 13606-1:2008” [Online] Available https://www.iso.org/standard/40784.html [Accessed 02 May 2017]

[37] “CEN/ISO EN 13606 Archetype Model” [Online] Available http://www.en13606.org/the-ceniso-en13606-standard/archetype-model [Accessed 02 May 2017]

[38] “CEN/ISO EN 13606 Reference Model” [Online] Available http://www.en13606.org/the-ceniso-en13606-standard/reference-model[Accessed 02 May 2017]

[39] “Clinical Models Program” [Online] Available http://www.openehr.org/programs/clinicalmodels/ [Accessed 02 May 2017]

[40] Gunnar Piho, Associate Professor in Information Systems, Head of Business Information Technology study program

[41] Dmitri Maksimov, master’s thesis, “Performance Comparison of MongoDB and PostgreSQL with JSON types”, 2015

[42] Fotache, Marin; Cogean, Dragos. Informatica Economica, “NoSQL and SQL Databases for Mobile Applications. Case Study: MongoDB versus PostgreSQL”, 2013 [Online] Available http://search.proquest.com/openview/b7c8815b88139f1f02b157a7436cac9e/1?pq-origsite=gscholar&cbl=55108 [Accessed 02 May 2017]

[43] Stephan Schmid, Eszter Galicz, Wolfgang Reinhardt, “Performance investigation of selected SQL and NoSQL databases”, June 2015 [Accessed 02 May 2017]

[44] Ben Säid M, Simonet A, Guillon D, Jacquelinet C, Gaspoz F, Dufour E, Mugnier C, Simonet M, Landais P “A dynamic Web application within an n-tier architecture: a Multi-Source Information System for end-stage renal disease.” 01 January 2003, [Online] Available http://europepmc.org/abstract/med/14663969 [Accessed 02 May 2017]

[45] “Care2x the Open Source Hospital Information System” [Online] Available http://www.care2x.org/solution [Accessed 03 May 2017]

[46] EMS PostgreSQL Data Generator 2.0 [Online] Available https://www.sqlmanager.net/products/postgresql/datagenerator/download [Accessed 05 May 2017]

[47] “Pearson Correlation Coeficient Calculator” [Online] Available http://www.socscistatistics.com/tests/pearson/Default2.aspx [Accessed 14 May 2017]

[48] “Mockaroo – realistic data generator” [Online] Available https://mockaroo.com/ [Accessed 09 May 2017]

[49] “Resource index” [Online] Available https://www.hl7.org/fhir/resourcelist.html [Accessed 5 November 2016]

[50] “IntelliJ IDEA” – IDE for JVM

[51] “PostgreSQL 9.6.3 Documentation” [Online] Available https://www.postgresql.org/docs/9.6/static/sql-explain.html [Accessed 18 May 2017]

[52] “Covering Indexes” [Online] https://developer.couchbase.com/documentation/server/current/indexes/covering-indexes.html [Accessed 19 May 2017]

[53] “Optimize Query with JOIN” [Online] Available https://forums.couchbase.com/t/optimize-query-with-join/11977 [Accessed 18 May 2017]

[54] “Коэффициент корреляции Пирсона” [Online] Available http://planetcalc.ru/527/ [Accessed 19 May 2017]

[55] Nick raboy, “Using Covering Indexes on a Multiple Document Type Bucket” 17 May 2016 [Online] Available https://blog.couchbase.com/using-covering-indexes-on-a-multiple-document-type-bucket/ [Accessed 18 May 2017]

[56] Pawel Krawczyk “European personal data regex pattern”, 20 April 2012 [Online] Available https://ipsec.pl/data-protection/2012/european-personal-data-regexp-patterns.html [Accessed 25 April 2017]

[57] SQL standards – zip file , Available http://www.wiscorp.com/sql20nn.zip [Accessed 19 May 2017]

[58] “DB-Engines Ranking of Document Stores” [Online] Available https://db-engines.com/en/ranking/document+store [Accessed 10 Oxtober 2016]

[59] “Annotation Type EnableCpuchbaseRepositories” [Online] Available http://docs.spring.io/spring-data/couchbase/docs/current/api/org/springframework/data/couchbase/repository/config/EnableCouchbaseRepositories.html [Accessed 26 April 2017]

[60] “SQL for JSON” [Online] Available https://www.couchbase.com/n1ql [Accessed 15 April 2017]

[61] “Encyclopedia Index” [Online] Available https://db-engines.com/en/articles [Accessed 19 May 2017]

[62] “RDF stored” [Online] Available https://db-engines.com/en/article/RDF+Stores?ref=Tripel+Stores [Accessed 19 May 2017]

[63] “Population Register” [Online] Available https://e-estonia.com/component/population-register/ [Accessed 22 May 2017]

[64] Chris Gallant, “What is the difference between arithmetic and geometric averages?” 03 May 2017, [Online] Available http://www.investopedia.com/ask/answers/06/geometricmean.asp [Accessed 22 May 2017]

[65] “Pearson correlation coefficient” [Online] Available https://en.wikipedia.org/wiki/Pearson_correlation_coefficient [Accessed 22 May 2017]

[66] “ISO/IEC 5218” [Online] Available https://en.wikipedia.org/wiki/ISO/IEC_5218 [Accessed 22 May 2017]

[67] “PriEst tool” [Online] Available https://sourceforge.net/projects/priority/ [Accessed 22 May 2017]

[68] “BPMSG AHP Online System” [Online] Available http://bpmsg.com/academic/ahp.php [Accessed 22 May 2017]

Appendix 1 – PostgreSQL Schema Change Script

```
4   ALTER TABLE patient ADD COLUMN medical_institution_id BIGINT;
5   ALTER TABLE patient ADD CONSTRAINT FK_doctor_medical_institution
6   FOREIGN KEY (medical_institution_id) REFERENCES medical_institution (medical_institution_id)
7   ON DELETE No Action ON UPDATE No Action;
8
9
10  ALTER TABLE person drop column if exists is_deceased ;
11  ALTER TABLE person drop column if exists is_retired ;
```

Figure 28 PostgreSQL Migration Script

Appendix 2 – PostgreSQL DBMS Queries

1. Experiment 1 – PostgreSQL query 1.1

```
explain analyze select d.document_id,
dt.document_type_display,
i.medical_institution_name,
(array_agg(hn_pat.full_name))[1] as pat_name,
(array_agg(hn_doc.full_name))[1] as doc_name
from document as d
inner join document_type as dt on dt.document_type_id = d.document_type_id
inner join medical_institution as i on i.medical_institution_id = d.medical_institution_id
inner join patient as pat on pat.patient_id = d.patient_id
inner join person as pat_per on pat_per.person_id = pat.person_id
left outer join human_name_person as hnp_pat on hnp_pat.person_id = pat_per.person_id
left outer join human_name hn_pat on hn_pat.human_name_id = hnp_pat.human_name_id

inner join doctor as doc on doc.doctor_id = d.author_id
inner join person as doc_per on doc_per.person_id = doc.person_id
left outer join human_name_person as hnp_doc on hnp_doc.person_id = doc_per.person_id
left outer join human_name hn_doc on hn_doc.human_name_id = hnp_doc.human_name_id


where hn_pat.end_date is null and hn_doc.end_date is null
group by d.document_id, dt.document_type_display,i.medical_institution_name
```

Figure 29 PostgreSQL Query with Joins for Experiment 1

2. Experiment 1 – PostgreSQL query 1.2

```
explain analyze select
document_id,
patient_id,
medical_institution_id,
author_id,
document_type_id
from document
```

Figure 30 PostgreSQL Query for Experiment 1

3. Experiment 2 – PostgreSQL query 2.1

```
explain analyze select d.document_id,
dt.document_type_display,
i.medical_institution_name,
(array_agg(hn_pat.full_name))[1] as pat_name,
(array_agg(hn_doc.full_name))[1] as doc_name
from document as d
inner join document_type as dt on dt.document_type_id = d.document_type_id
inner join medical_institution as i on i.medical_institution_id = d.medical_institution_id
inner join patient as pat on pat.patient_id = d.patient_id
inner join person as pat_per on pat_per.person_id = pat.person_id
left outer join human_name_person as hnp_pat on hnp_pat.person_id = pat_per.person_id
left outer join human_name hn_pat on hn_pat.human_name_id = hnp_pat.human_name_id

inner join doctor as doc on doc.doctor_id = d.author_id
inner join person as doc_per on doc_per.person_id = doc.person_id
left outer join human_name_person as hnp_doc on hnp_doc.person_id = doc_per.person_id
left outer join human_name hn_doc on hn_doc.human_name_id = hnp_doc.human_name_id


where hn_pat.end_date is null and hn_doc.end_date is null
and d.patient_id = 100 and d.document_type_id = 15
and (document -» 'resourceType' ) LIKE('Observation')
group by d.document_id, dt.document_type_display,i.medical_institution_name
```

Figure 31 PostgreSQL Query with Joins for Experiment 2

4. Experiment 2 – PostgreSQL query 2.2

```
explain analyze select
document_id,
patient_id,
medical_institution_id,
author_id,
document_type_id
from document
where patient_id = 100
and document_type_id = 15
and (document -» 'resourceType' ) LIKE('Observation')
```

Figure 32 PostgreSQL Query for Experiment 2

5. Experiment 3 – PostgreSQL query 3.1

```
explain analyze update document set document_type_id = 13, document = '{
          "resourceType": "DiagnosticReport",
          "id": "example-pgx",
          "effectiveDateTime": "2016-10-15T12:34:56+11:00",
          "issued": "2016-10-20T14:00:05+11:00",
          "performer": [
            {
              "actor": {
                "reference": "Organization/4829"
              }
            }
          ],
          "result": [
            {
              "reference": "Observation/example-phenotype"
            }
          ],
          "presentedForm": [
            {
              "contentType": "PDF",
              "language": "en",
              "data": "cGRmSW5CYXNlNjRCaW5hcnk=",
              "hash": "571ef9c5655840f324e679072ed62b1b95eef8a0",
              "title": "Pharmacogenetics Report",
              "creation": "2016-10-20T20:00:00+11:00"
            }
          ]
        }'
where document_id = 100100
```

Figure 33 PostgreSQL for Experiment 3

6. Experiment 4 – PostgreSQL query 4.1

explain analyze insert into document (document_id, document_type_id, author_id, patient_id, medical_institution_id, document) values (1000104, 16, 707, 238, 1,

```
'{
        "resourceType": "DetectedIssue",
        "id": "ddi",
        "status": "final",
        "category": {
                "coding": [

                    {
                        "system": "http://hl7.org/fhir/v3/ActCode",
                        "code": "DRG",
                        "display": "Drug Interaction Alert"
                      }
                  ]
                },
        "severity": "high",
        "date": "2014-01-05",
        "author": {
                "reference": "Device/software"
        },
        "implicated": [
                {
                  "reference": "MedicationStatement/example001",
                  "display": "500 mg Acetaminophen tablet 1/day, PRN since
2010"
                },
                {
                "reference": "MedicationRequest/medrx0331",
                "display": "Warfarin 1 MG TAB prescribed Jan. 15, 2015"
                }
        ],
```

"detail": "Risk of internal bleeding. Those who take acetaminophen along with the widely used blood-thinning drug warfarin may face the risk of serious internal bleeding. People on warfarin who take acetaminophen for at least seven days in a row should be closely watched for bleeding.",

"mitigation": [

{

"action": {

"coding": [

{

"system": "http://hl7.org/fhir/v3/ActCode",

"code": "13",

"display": "Stopped Concurrent Therapy"

}

],

"text": "Asked patient to discontinue regular use of Tylenol and to consult with clinician if they need to resume to allow appropriate INR monitoring"

},

"date": "2014-01-05",

"author": {

"reference": "Practitioner/example",

"display": "Dr. Adam Careful"

}

}

]

}');

Figure 34 PostgreSQL for Experiment 4

Appendix 3 – Couchbase DBMS Queries

1. Experiment 1 – Couchbase query 1.1

```
select
document.id,
document_type.display,
institution.medical_institution_name,
ARRAY_AGG(DISTINCT patient_current_names.full_name) as patient_name,
ARRAY_AGG(DISTINCT doctor_current_names.full_name) as doctor_name
from `medical-institution` document
INNER JOIN `medical-institution` document_type
ON KEYS document.type_id
INNER JOIN `medical-institution` institution
ON KEYS document.institution_id
INNER JOIN `medical-institution` patient
ON KEYS document.patient_id
LEFT OUTER UNNEST patient.person.human_names patient_current_names
INNER JOIN `medical-institution` doctor
ON KEYS document.patient_id
LEFT OUTER UNNEST doctor.person.human_names AS doctor_current_names
WHERE document.entity_type = 'document'
AND institution.entity_type = 'institution'
AND patient.entity_type = 'patient'
AND doctor.entity_type = 'doctor'
AND document_type.entity_type = 'document_type'
AND (patient_current_names.end_date IS NULL OR patient_current_names.end_date IS MISSING)
AND (doctor_current_names.end_date IS NULL OR doctor_current_names.end_date IS MISSING)
GROUP BY document_type.display, document.id, institution.medical_institution_name
```

Figure 35 Couchbase Query with Joins for Experiment 1

2. Experiment 1 – Couchbase query 1.2

```
select
document.id,
document.type_id,
document.patient_id,
document.institution_id,
document.doctor_id
from `medical-institution` document
USE INDEX (def_document_list_ix USING GSI)
where document.entity_type = 'document'
AND document.id IS NOT MISSING
AND document.type_id IS NOT MISSING
AND document.patient_id IS NOT MISSING
AND document.institution_id IS NOT MISSING
AND document.doctor_id IS NOT MISSING
```

Figure 36 Couchbase Query for Experiment 1

3. Experiment 2 – Couchbase query 2.1

```
select
document.id,
document_type.display,
institution.medical_institution_name,
ARRAY_AGG(DISTINCT patient_current_names.full_name) as patient_name,
ARRAY_AGG(DISTINCT doctor_current_names.full_name) as doctor_name
from `medical-institution` document
INNER JOIN `medical-institution` document_type
ON KEYS document.type_id
INNER JOIN `medical-institution` institution
ON KEYS document.institution_id
INNER JOIN `medical-institution` patient
ON KEYS document.patient_id
LEFT OUTER UNNEST patient.person.human_names patient_current_names
INNER JOIN `medical-institution` doctor
ON KEYS document.patient_id
LEFT OUTER UNNEST doctor.person.human_names AS doctor_current_names
WHERE document.entity_type = 'document'
AND institution.entity_type = 'institution'
AND patient.entity_type = 'patient'
AND doctor.entity_type = 'doctor'
AND document_type.entity_type = 'document_type'
AND (patient_current_names.end_date IS NULL OR patient_current_names.end_date IS MISSING)
AND (doctor_current_names.end_date IS NULL OR doctor_current_names.end_date IS MISSING)
AND document.content.resourceType = 'Observation'
GROUP BY document_type.display, document.id, institution.medical_institution_name
```

Figure 37 Couchbase Query with Joins for Experiment 2

4. Experiment 2 – Couchbase query 2.2

```
select
document.id,
document.institution_id,
document.doctor_id
from `medical-institution` document
where document.entity_type = 'document'
and document.patient_id = 'patient_100'
and document.type_id = 'document_type_15'
and document.doctor_id is not missing
and document.institution_id is not missing
and document.id is not missing
and document.patient_id is not missing
and document.type_id is not missing
and document.content.resourceType = 'Observation'
```

Figure 38 Couchbase Query for Experiment 2

5. Experiment 3 – Couchbase query 3.1

```
UPDATE `medical-institution`
USE KEYS 'document_100100'
SET document_type_id = 13, content = '{
            "resourceType": "DiagnosticReport",
            "id": "example-pgx",
            "effectiveDateTime": "2016-10-15T12:34:56+11:00",
            "issued": "2016-10-20T14:00:05+11:00",
            "performer": [
              {
                "actor": {
                  "reference": "Organization/4829"
                }
              }
            ],
            "result": [
              {
                "reference": "Observation/example-phenotype"
              }
            ],
            "presentedForm": [
              {
                "contentType": "PDF",
                "language": "en",
                "data": "cGRmSW5CYXNlNjRCaW5hcnk=",
                "hash": "571ef9c5655840f324e679072ed62b1b95eef8a0",
                "title": "Pharmacogenetics Report",
                "creation": "2016-10-20T20:00:00+11:00"
              }
            ]
          }'
returning meta().id
```

Figure 39 Couchbase Query for Experiment 3

6. Experiment 4 – Couchbase query 4.1

```
INSERT INTO `medical-institution` ( KEY, VALUE )
  VALUES
  (
    "document_1000104",
    {
  "doctor_id": "doctor_1",
  "entity_type": "document",
  "patient_id": "patient_681",
  "type_id": "document_type_16",
  "id": 1000104,
  "content": '{
                    "resourceType": "DiagnosticReport",
                    "id": "example-pgx",
                    "effectiveDateTime": "2016-10-15T12:34:56+11:00",
                    "issued": "2016-10-20T14:00:05+11:00",
                    "performer": [
                        {
                        "actor": {
                            "reference": "Organization/4829"
                            }
                        }
                    ],
                    "result": [
                        {
                            "reference": "Observation/example-phenotype"
                        }
                    ],
                    "presentedForm": [
                        {
                        "contentType": "PDF",
                        "language": "en",
                        "data": "cGRmSW5CYXNlNjRCaW5hcnk=",
                        "hash": "571ef9c5655840f324e679072ed62b1b95eef8a0",
                        "title": "Pharmacogenetics Report",
                        "creation": "2016-10-20T20:00:00+11:00"
                    }
                    ]
                }',
  "institution_id": "institution_38"
})
RETURNING META().id as docid, *;
```

Figure 40 Couchbase Query for Experiment 4

Appendix 4 – Couchbase Constraints

1. human_name.start_date < human_name.end_date;
2. address.start_date < address.end_date;
3. contact_point.start_date < contact_point.end_date;
4. either person.id_code or person.foreigner_iderntifier is not null;
5. person.id_code must match regex "([1-6][0-9]{2}[1,2][0-9][0-9]{2}[0-9]{4})|^$" [56] ;
6. person.id_code must be unique if not empty;

Appendix 5 – Couchbase Indexes

1. CREATE PRIMARY INDEX `def_primary_ix` ON `medical-institution`;
2. CREATE INDEX `def_id_ix` ON `medical-institution`(`id`);
3. CREATE INDEX `id_meta_ix` ON `medical-institution`((meta().`id`));
4. CREATE INDEX `def_documents_2_no_join_ix` ON `medical-institution`(`id`,`type_id`,`patient_id`,`institution_id`,`doctor_id`) WHERE ((`document`.`entity_type`) = "document");
5. CREATE INDEX `def_document_list_ix` ON `medical-institution`(`id`,`type_id`,`patient_id`,`institution_id`,`doctor_id`) WHERE (`entity_type` = "document").
6. CREATE INDEX `def_document_institution_join_with_pars_ix_2` ON `medical-institution`(`id`,`type_id`,`patient_id`,`doctor_id`,`institution_id`,`medical_institution_name`,(`content`.`resourceType`)) WHERE ((((`document`.`entity_type`) = "document") and ((`document`.`patient_id`)IS NOT MISSING)) and ((`document`.`type_id`) IS NOT MISSING))

Appendix 6 – AHP Matrices

In order to perform AHP analysis, I used PriEst tool [67] . Unfortunately, the GUI of the tool did not allow me to make screenshot of the criteria. Therefore, I used another online tool – BPMSG AHP Online System[68] , which allowed me to have criteria and evaluation in readable format.

Priorities

These are the resulting weights for the criteria based on your pairwise comparisons

	Category	Priority	Rank
1	Primary key support	36.3%	1
2	Foreign key support	10.0%	4
3	Indexing	17.2%	2
4	Joins	9.6%	5
5	Schema-free (not required)	6.7%	6
6	Java support	14.6%	3
7	Open source	5.5%	7

Number of comparisons = 21
Consistency Ratio CR = 5.6%

Decision Matrix

The resulting weights are based on the principal eigenvector of the decision matrix

	1	2	3	4	5	6	7
1	1	5.00	2.00	8.00	3.00	1.00	7.00
2	0.20	1	0.50	1.00	2.00	1.00	2.00
3	0.50	2.00	1	2.00	2.00	2.00	2.00
4	0.12	1.00	0.50	1	2.00	1.00	2.00
5	0.33	0.50	0.50	0.50	1	0.50	1.00
6	1.00	1.00	0.50	1.00	2.00	1	3.00
7	0.14	0.50	0.50	0.50	1.00	0.33	1

Principal eigen value = 7.450
Eigenvector solution: 5 iterations, delta = 6.6E-8

Figure 41 Comparison Matrices BPMSG AHP Online System

Figure 41 includes also calculated Consistency Ratio and Eigen value. Consistency Ratio shows the percentage of inconsistency in matrix. It is allowed to be up to 10%.

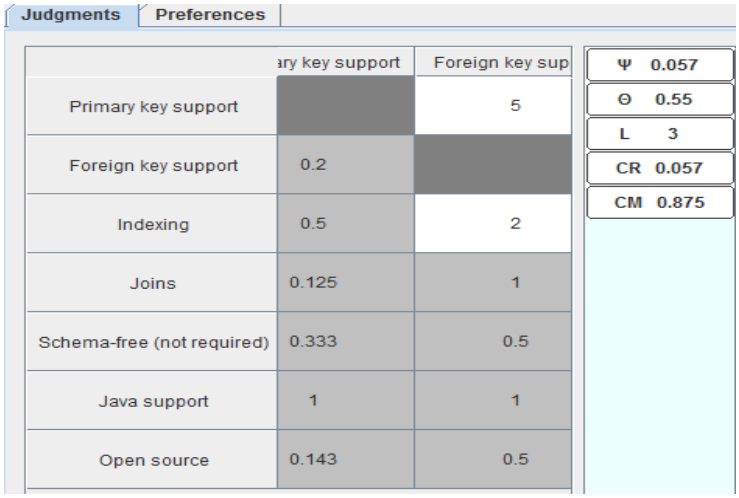

Figure 42 Comparison Matrices PriEst

Figure 42 shows the same Consistency Ratio with different precision (screenshot is taken from PriEst).

Appendix 7 – Possible Problems of the Standard

HL7 FHIR specification describes a set of resources to exchange over different systems. Although it is officially stated that the rules in the specification are quite loose to cover bigger amount of possible use cases, it leads to problems.

In this work I analyzed some of the resources. The analysis was performed only to the resources and fields, which are present in the implementation of the test application.

In the description under the figures, the problematic fields are described and a possible suggestion is provided.

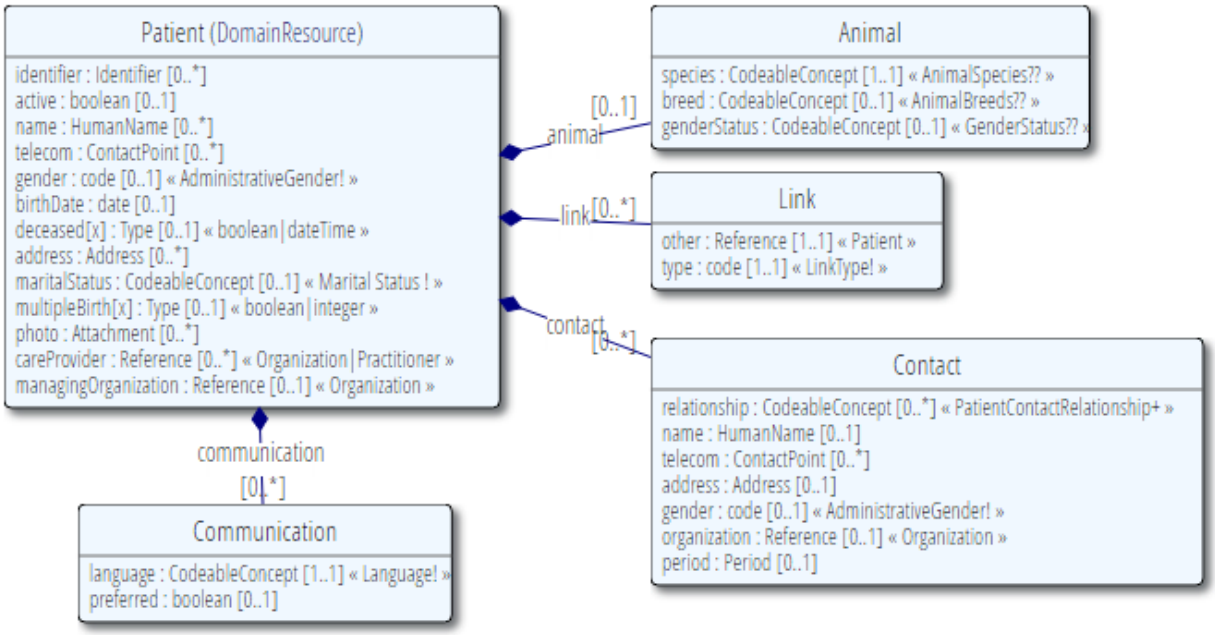

Figure 43 Patient Resource [30]

- Identifier <Identifier>:

Definition: "An identifier for this patient. This is a business identifier, not a resource identifier. "[17]	Requirements: "Patients are almost always assigned specific numeric identifiers." [17]
Implementation: In the application, patient will have unique identifier, which is required. Since the	Suggestion: In real-life application, patients must have an identifier. In most of the cases, id-code is used as a unique identifier. Since the

assumption of the system is that it is used in Estonia only and all the residents of Estonia have unique id-code, unique identifier of the patient is an Estonian id code.	standard is not used only in one country only, the suggestion is to form the identifier from the country and region code + id-code of the person and make this field required. This would allow keeping the data about every single patient consistent, since patient will be referenced via unique identifier.

- Active <Boolean>:

Definition: "Whether this patient record is in active use." [17]	Requirements: "Need to be able to mark a patient record as not to be used because it was created in error." [17]
Implementation: In the application, the "active" field has the type Boolean and is required.	Suggestion: According to the standard, the field is not required, although in real systems patient's profile should be either active or not, no middle state should be allowed. Suggestion is to make the field required with the default value set to "true".

- Name <HumanName>:

Definition: "A name associated with the individual." [17]	Requirements: "Need to be able to track the patient by multiple names. Examples are your official name and a partner name." [17]
Implementation: In the application, the "name" field is required.	Suggestion: According to the standard, the field is not required, although patient must

	have at least some name. Suggestion is to make this field required.

- Gender <Code>:

Definition: "Administrative Gender - the gender that the patient is considered to have for administration and record keeping purposes." [17]	Requirements: "Needed for identification of the individual, in combination with (at least) name and birth date. Gender of individual drives many clinical processes." [17]
Implementation: In the application, the "gender" field is required and ISO/IEC 5218 standard will be used.	Suggestion: According to the standard, the field is not required. The suggestion is to make this field required since every person has a gender, which would correspond to the ISO code. In addition, the gender code has the type "code" which values are not specified in the standard. The suggestion is to restrict the coding system on the standard level, which would decrease the data inconsistency. The standard could use the ISO coding of human sexes [66]

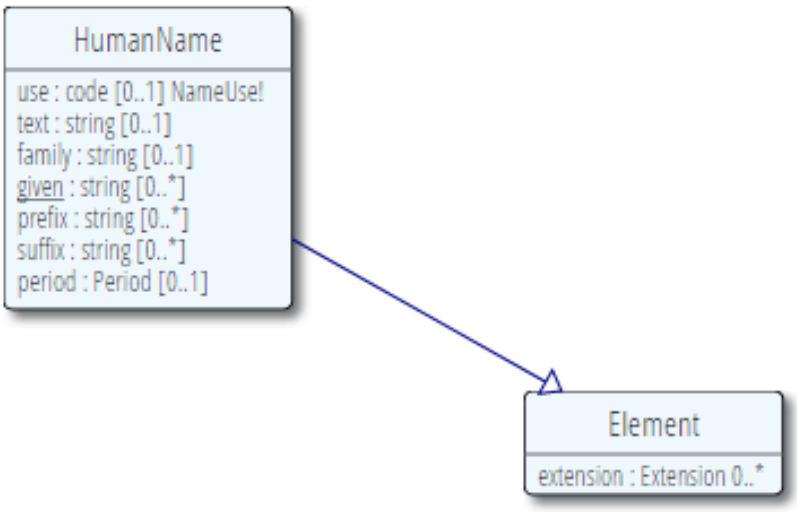

Figure 44 Human Name Resource [19]

- Use <Code>:

Definition: "Identifies the purpose for this name." [19]	Requirements: "Allows the appropriate name for a particular context of use to be selected from among a set of names." [19]
Implementation: In the application, the "use" field is required.	Suggestion: According to the standard, the field is not required. Since the field is a classifier for the "HumanName", it is suggested to make it required, because every name has some use.

- Given <String>:

Definition: "Given name." [19]	
Implementation: In the application, the "given" can be empty or have just one value.	Suggestion: According to the standard, the field might have multiple values. Suggestion is to make the connection [0...1], like it is with family name, since the type of data is <String>. The field may have multiple values space- or comma separated

- Prefix <String>:

Definition: "Part of the name that is acquired as a title due to academic, legal, employment or nobility status, etc. and that appears at the start of the name." [19]	
Implementation: In the application, the "prefix" can be empty or have just one value.	Suggestion: According to the standard, the field might have multiple values. Suggestion is to make the connection [0...1], like it is with family name, since the type of data is <String>.

	The field may have multiple values space- or comma separated.

- Suffix <String>:

Definition: "Part of the name that is acquired as a title due to academic, legal, employment or nobility status, etc. and that appears at the end of the name." [19]	
Implementation: In the application, the "suffix" can be empty or have just one value.	Suggestion: According to the standard, the field might have multiple values. Suggestion is to make the connection [0...1], like it is with family name, since the type of data is <String>. The field may have multiple values space- or comma separated.

- Period <Period>:

Definition: "Indicates the period of time when this name was valid for the named person." [19]	
Implementation: In the application, the field "period" is not present, instead of that there are two fields – "start_date" and "end_date". Only "start_date" is required.	Suggestion: According to the standard, the field is not required. Since patient should have at least one active name and the entity does not have "active" field, the suggestion is to make the field required.

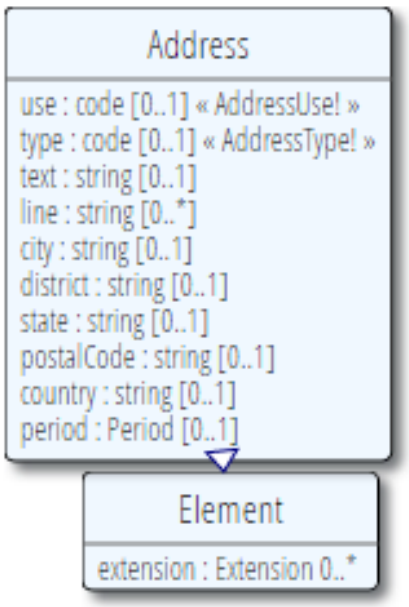

Figure 45 Address Resource [18]

- Use <Code>:

Definition: "The purpose of this address." [18]	Requirements: "Allows an appropriate address to be chosen from a list of many." [18]
Implementation: In the application, the "use" field is required.	Suggestion: In the standard, the field is not required. The suggestion is to make the field required in order to have the use for every address in the system.

- Type <Code>:

Definition: "Distinguishes between physical addresses (those you can visit) and mailing addresses (e.g. PO Boxes and care-of addresses). Most addresses are both. " [18]	
Implementation: In the application, the "type" field is required.	Suggestion: In the standard, the field is not required. The suggestion is to make the field

	required in order to have the type for every address in the system.

- Text <String>:

Definition: "A full text representation of the address." [18]	Requirements: "A renderable, unencoded form." [18]
Implementation: In the application, the "text" field is required.	Suggestion: In the standard, the field is not required. The suggestion is to make the field required because if there is any address entry, that is possible to split between other fields, it should be possible to put the value to this field as well.

- Country <String>:

Definition: "Country - a nation as commonly understood or generally accepted." [18]	Requirements: "A renderable, unencoded form." [18]
Implementation: In the application, the "country" field is required and ISO 3166 codes are used.	Suggestion: In the standard, the field is not required. Since every address should belong to some country, the suggestion is to make this field required. The recommendation from the standard itself is: "ISO 3166 3 letter codes can be used in place of a full country name." [18].

- Period <Period>:

Definition: "Time period when address was/is in use" [19]	Requirements: "Allows addresses to be placed in historical context." [19]

Implementation: In the application, the field “period” is not present, instead of that there are two fields – “start_date” and “end_date”. Only “start_date” is required.	Suggestion: According to the standard, the field is not required. Since patient should have at least one active address and the entity does not have “active” field, the suggestion is to make the field required.

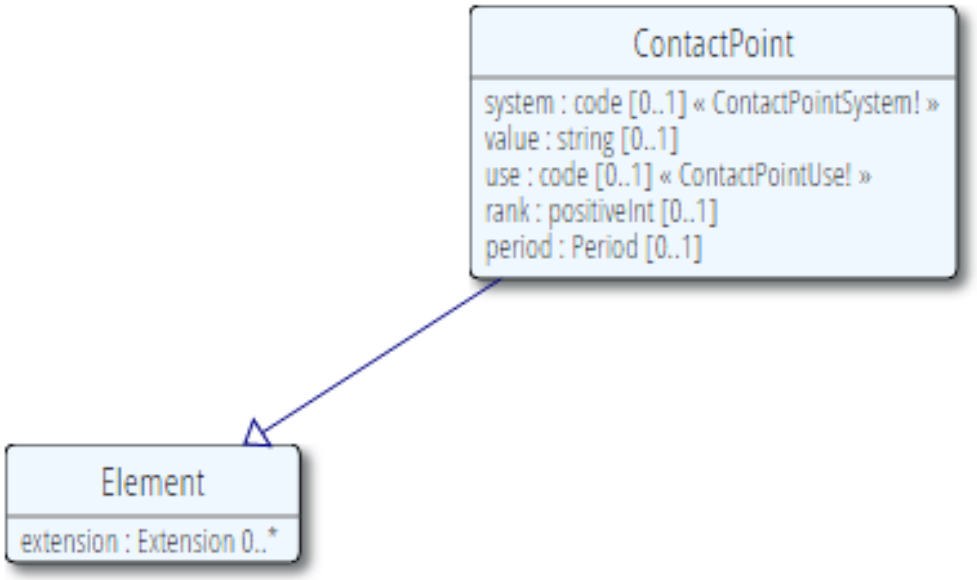

Figure 46 ContactPoint Resource [20]

- System <Code>:

Definition: “Telecommunications form for contact point - what communications system is required to make use of the contact.” [20]	
Implementation: In the application, the field “system” is required.	Suggestion: According to the standard, the field is not required. In order to avoid storing misleading data (contact information without the system to make use of it), it is recommended to make the field required.

- Value <String>

Definition: "The actual contact point details, in a form that is meaningful to the designated communication system (i.e. phone number or email address). [20]	Requirements: "Need to support legacy numbers that are not in a tightly controlled format." [20]
Implementation: In the application, the field "value" is required.	Suggestion: According to the standard, the field is not required although it does not make sense to store the other attributes of the object since they are not usable. Therefore, it is recommended to make the field required.

- Period <Period>:

Definition: "Time period when contact point was/is in use." [20]	
Implementation: In the application, the field "period" is not present, instead of that there are two fields – "start_date" and "end_date". Only "start_date" is required.	Suggestion: According to the standard, the field is not required. Since patient should have at least one active address and the entity does not have "active" field, the suggestion is to make the field required.

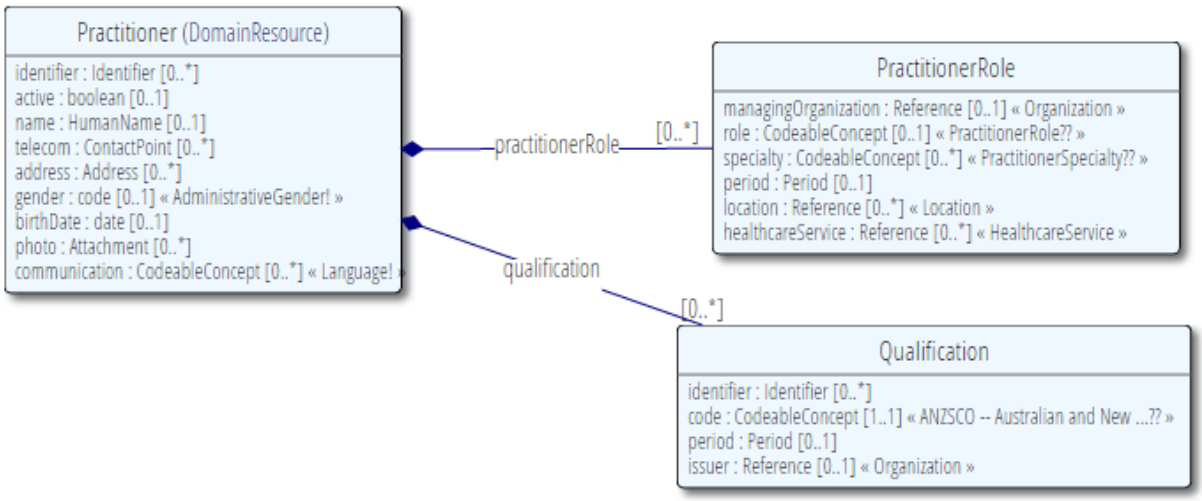

Figure 47 Practitioner Resource [29]