Using CQRS Pattern for Improving Performances in Medical Information Systems

Petar Rajković
University of Niš,
Faculty of Electronic Engineering
Aleksandra Medvedeva 14
+381 18 588448
petar.rajkovic@elfak.ni.ac.rs

Dragan Janković
University of Niš,
Faculty of Electronic Engineering
Aleksandra Medvedeva 14
+381 18 529101
dragan.jankovic@elfak.ni.ac.rs

Aleksandar Milenković
University of Niš,
Faculty of Electronic Engineering
Aleksandra Medvedeva 14
+381 18 588448
amilenkovic@elfak.ni.ac.rs

ABSTRACT

During exploitation of a medical information system dedicated to primary care facilities, we realized that significant amount of often used, but rarely changed data is scattered in large number of data tables requiring many join operations when need to be retrieved. Additionally, in many cases, many data fields are retrieved by select queries and later neither displayed on user interface, nor used in any other way. This situation could be a cause for some performance issues as well as unnecessary increase of data traffic. For this reason we decided to improve our system by applying command-query responsibility segregation (CQRS) pattern with de-normalized read database in order to reduce time and data amount needed for some often executed queries. In read database design process we applied model driven approach and used our existing data modeling, mapping and code generating tools. Also, we developed a synchronization component responsible for migrating data from main to read database based on the existing data replicator. In this paper we present the results on applying this approach mainly on demographic, administrative and partly on medical data. In the near future we plan to extend this approach in our medical information system as much as possible.

Categories and Subject Descriptors

H.4.0 [**Information Systems Applications**] Information system applications - General.

General Terms

Performance, Design, Human Factors

Keywords

Medical Information Systems, command-query responsibility segregation, model driven development.

1. INTRODUCTION AND MOTIVATION

During last four years we worked on the development of medical information system (MIS) called Medis.NET which is dedicated for the use in primary and ambulatory care medical facilities in the Republic of Serbia [1]. Since it became fully operative, the MIS developed by our research group is deployed in more than 20 different medical institutions, and after more than two years of full-scale use we identified the system weak spots and start working on possible architectural improvements. In this paper we will describe the problem related by slower system response as a result of data traffic rate higher than really needed.

We realize that we have several queries that are executed very often and which load much larger amount of data, scattered in several tables, than displayed on the user interface. We examined two different approaches – defining reduced data classes in the data object model and to generate separate database that will be used only for reading the data.

We choose the second approach, since generated read database could be used later for personal health Web site and reduce the load on our main database. For implementing the concept with read database we decided to use command-query responsibility segregation CQRS architectural pattern [2] and experiment initially with a limited set of data stored in read database. The results we get encourage us to continue with this approach and to support more data access processes within read database.

In this paper we present our results on applying separate read database for some often used queries. In section 2 the brief description of CQRS architectural pattern is given. Section 3 contains description of CQRS supporting component, while the structure of our initial read database along with experimental results are given in the section 4. Section 5 contains brief overview on related work. In the last section we present guidelines for our future work and a conclusion.

2. ABOUT CQRS PATTERN

The main idea with CQRS pattern is to divide data store and have one that serves only for performing insert, update and delete, by one word commands and another that serves only for select queries. The connection between two data stores is done through synchronization component that is responsible for ensuring data consistency. Usually those two data stores are called main database and read database.

Unlike the standard multitier architecture that request strict separation between layers, CQRS introduces separation within components of the same layer [2]. One possible approach to CQRS is given on the Figure 1. Presentation layer will remain unique, while business logic layer is separated on two parts where one (called command logic on Figure 1) supports insert/update/delete commands and another handles read requests.

Data storage layer consists of two databases – main and read. Main is connected to command logic, while read sends data to query logic. Link between main and read data stores is synchronization/de-normalization component responsible for ensuring data consistency. When CQRS pattern is fully implemented, main database accepts no select queries from any part of the business logic layer while read database serves only for read operations.

CQRS approach is more suitable for the systems where number of select queries is far bigger than number of commands, such is a case with online shopping services, but it is applicable on standard information systems too.

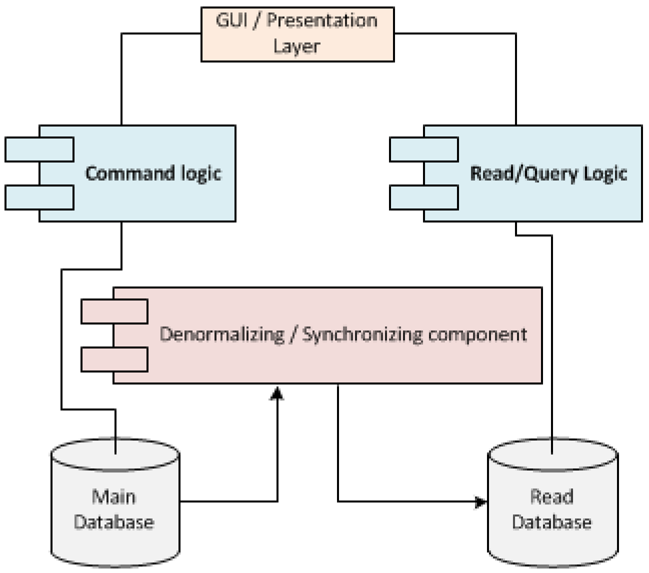


Figure 1 Relation of extension service components and major components of base system

3. INTRODUCING CQRS CONCEPTS

As we have running medical information system, we decided to introduce CQRS approach incrementally in order to initially test performances on an insulated set of functionality. Eventually, all read requests will end up in read/query logic component (Figure 1). Until that point, main database will not serve only commands, but some read requests too.

During design phase we decided that we should left possibility that our system could work both with (CQRS working mode) and without read database (standard working mode). This requests a design update that will be explained later. Also, we decided that initial set of data that will be moved to read database would be chosen by the rule "often used – rarely changed".

As it has been mentioned before, we ensure that our system could operate equally in a standard mode (with only main data store installed) and in a mode where dedicated read data base is present. Since all read operations will pass through same part of the business logic, we support hot-swap on reading operations target database that can be used if read database went offline. Thanks to that feature, we can disconnect read database, update its structure, synchronize data and bring it back online.

Visual components, as a part of a presentation layer of Medis.NET information system, communicate with a standard business logic layer and data access process is a transparent process looking from their perspective. They always send equally formatted request and expect the same response from the business logic layer.

The business logic layer passes request to a data model layer that encapsulates data access objects and operations. For read operations that are envisioned to work with both databases, the set of interfaces is defined. In further text we will refer them as read operation interfaces. Read operation interfaces are by default implemented in an object model that supports main database and they represent an extension point of the system. Classes responsible for reading data from main database already implements these interfaces. The result of this approach is that system will use main database both for read and write if read database is missing.

The classes within read/query logic component must implement same read interfaces that can be used. If some interface is not yet implemented in read/query logic, it will be used from existing command logic. With this approach, we will make our system running and not much dependent on the progress in development process.

Looking from the implementation process perspective, to make read database integrated in a system, assemblies containing its logic and data object model must be generated and loaded by the business logic layer. Since our information system is developed in the .NET technology, assembly reloading in a run-time is a secure and not complicated to implement operation.

Read logic, which interacts with read database object model, have to include read operation interfaces and implement those that have to be supported. The read component can support either all read interfaces or just a sub-set. Using this approach we tried to identify the best possible subsets of read interfaces that should be supported by read logic, and in the same time check what is the optimal solution for a structure in the read database.

3.1 Defining Read Database

Both for defining read database and corresponding data object model we used a slightly modified MEDIS.NET modeling and mapping tool [3]. Initially, modeling tool was intended for users having domain specific knowledge in order to design parts of the system. Based on that model, our generation tool is able to automatically generate several different software components from data tables to GUI components. The mapping tool was used then for defining field-level relations between classes or data tables.

For this application we used functionality of the mapping tool that loads the structure of main database along with all defined read interfaces and offer to user a possibility to define tables for read database and to establish a mapping between them. Tables for read database cannot be defined absolutely deliberately but only in conjunction to selected read interfaces.

Before start defining the mapping, user has to choose one of the read interfaces. By selecting single interface, the list of the fields from main database tables used in default read operation will appear. Then, user can start with defining tables for read database and to create a mapping. This mapping is used then for synchronization/de-normalization component, SDC in further text, as well as for the automatic code generation for the classes that will implement selected read interface. On this way, user can move query processes incrementally to read database ensuring overall data consistency.

At the end, generator tool will create data model component for read database, classes containing necessary read logic and triggers requested for the SDC.

3.2 Synchronization/De-normalization Component

The SDC has two main tasks – data synchronization and update-upon-action. The mappings created in read database definition process are used by SDC as a data copying rules, both during synchronization and triggering (update-upon-action) processes.

The data synchronization is a process when the SDC checks the differences between main and read database and perform the update on read. The data synchronization process is based on our existing platform independent database replication solution [4] and can work in invalidate-insert, delete-insert or clean-up mode.

Invalidate-insert mode works on the way that identifies edited and removed data rows from main database, marks corresponding data in read database as invalid, and copy to read database only newly created rows. This mode is used in case when database used for reading operations was offline and when need to be re-synchronized, but for some reason, old data has not to be removed.

Unlike invalidate-insert, the delete-insert mode removes from read database copies of updated and deleted rows and copy only newly added. Clean-up mode is used then to remove all invalidated rows from read database. This mode is also used mainly for data synchronization, and it usually is configured to run once daily in order to clean up read database form non actual data.

Update-upon-action is the main working mode of the SDC. In this mode, SDC is constantly active, listening data changes in the main database, captures them, perform reformatting and de-normalization and stores data in read database. Listening data changes in the main database is done through different trigger processes that react on changes in main and copy affected data to read database. The triggers are implemented within SDC and will react on any of insert, update and delete operation within its scope and initiate changes in the read database.

No matter in which mode SDC works, it is the only component that can insert, update or delete data from the read database. All the other components are used only to query data.

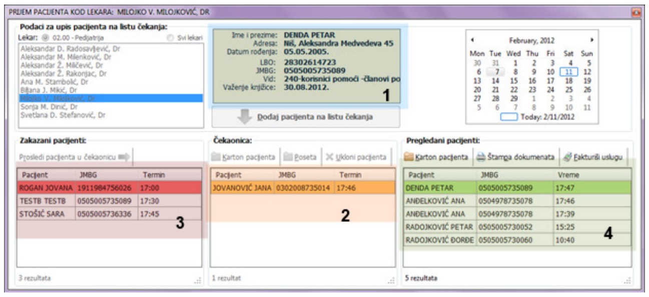
4. Example of Defined Read Database

The main database of the Medis.NET contains 348 different data tables and the instance we used as an example collected for one year of usage 5.5 GB of data, which is enough to perform different analysis. The database instance we used for analysis is installed in Primary Medical Care Center in Nis, Republic of Serbia. Mentioned medical institution serves as central primary care facility for a population of 430000 people. During one year, it has been 2.25 million of patient visits registered, along with total 3.4 million different medical examinations, laboratory analysis and therapeutic treatments.

It is interesting to note that the highest number of visits per a single patient during last year was fantastic 748. In the same time, around 230000 people have not visited doctor during last year. During every visit, the user of the medical information system passes through relatively same set of forms where the data from many tables are collected, joined and displayed. Doctors initially access the admission form. From the admission form they can either immediately start new visit, or, which is much more frequent, open patient's medical record overview containing wide variety of different, but rarely changed data. Our intention was to move these data, as much as possible, in read database and store in tables created by de-normalizing structure from the main database. Looking at the current database structure and activities we decided to start with demographic and administrative data since its change rate is very low (Table 1).

Table 1. Statistics for data table most used in select queries

Table	Row count	Added/Updated/Deleted row count (percent)
Patient	431567	17335 (4%)
MedicalRecord	360234	10170 (2.8%)
ScheduledVisit	1885030	78434 (4.1%)
Insurance	460333	55635 (12%)


PRIJEM PACIJENTA KOD LEKARA: MILOJKO V. MILOJKOVIĆ, DR
Podaci za upis pacijenta na listu čekanja:
Lekar: 02.00 - Pedjatrija
Svi lekari
Dodaj pacijenta na listu čekanja
Zakazani pacijenti:
Prosledi pacijenta u čekaonicu
Čekaonica:
Karton pacijenta
Poseta
Ukloni pacijenta
Pregledani pacijenti:
Štampa dokumenata
Fakturiši uslugu
February, 2012
Today: 2/11/2012
1
2
3
4

Aleksandar D. Radosavljević, Dr
Aleksandar M. Milenković, Dr
Aleksandar Ž. Milčević, Dr
Aleksandar Ž. Rakonjac, Dr
Ana M. Stambolić, Dr
Biljana J. Mikić, Dr
Milojko V. Milojković, Dr
Sonja M. Dinić, Dr
Svetlana D. Stefanović, Dr

Ime i prezime: DENDA PETAR
Adresa: Niš, Aleksandra Medvedeva 45
Datum rođenja: 05.05.2005.
LBO: 28302614723
JMBG: 0505005735089
Vid: 240-korisnici pomoći -članovi po
Važenje knjižice: 30.08.2012.

Pacijent	JMBG	Termin
ROGAN JOVANA	1911984756026	17:00
TESTB TESTB	0505005735089	17:30
STOŠIĆ SARA	0505005736336	17:45

3 rezultata

Pacijent	JMBG	Termin
JOVANOVIĆ JANA	0302008735014	17:46

1 rezultat

Pacijent	JMBG	Vreme
DENDA PETAR	0505005735089	17:47
ANĐELKOVIĆ ANA	0504978735078	17:46
ANĐELKOVIĆ ANA	0504978735078	17:39
RADOJKOVIĆ PETAR	0505005730052	15:25
RADOJKOVIĆ ĐORĐE	0505005730060	10:40

5 rezultata

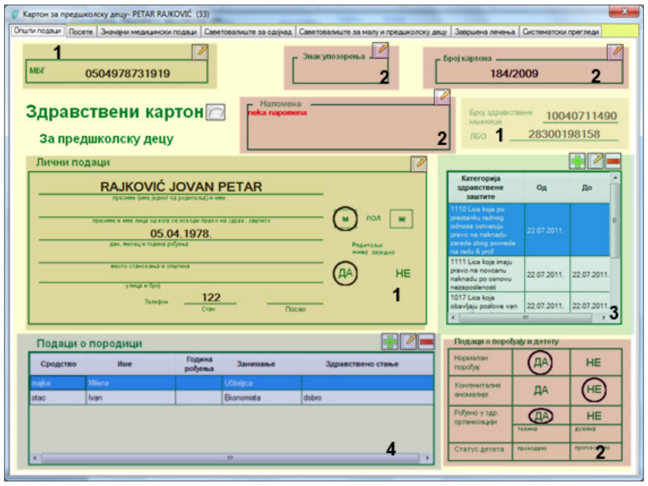
Figure 2 The overview of the admission form – 1: selected patient's personal data, 2: list of patient that are currently waiting, 3: list of scheduled visits, 4: list of already examined patient

Figure 2 shows the admission form. After patient has been selected, the admission form is displayed. In zone 1 of Figure 2 basic patient related data are displayed – name, address, date of birth, unique identifying number, type of insurance and insurance expiration date. These pieces of information are scattered in five different tables, and very rarely changed. This fact qualified them to move them in first table in our read database, table named ReadPatientData containing 13 different columns. These columns are copied from initial 7 different tables, where original tables were connected by 6 relations and contained total 72 columns. The underlying business process connected with visual component marked with the number 1 (PatientBasicDataDisplay) on the Figure 2 is designed to use one of the mentioned read interfaces. To enable using the data from ReadPatientData table, the new class implementing mentioned interface is created. Thanks to that, PatientBasicDataDisplay will each time connect to read database and get the data. Since PatientBasicDataDisplay appears in 26 different forms and controls, it will affect general data traffic reduction. This paper will not present these results but only those related to admission and medical record form.

Next candidate for moving to read database was table containing information on scheduled exams terms. It contains only 16 columns, but 8 constraints. In the admission form, data related to a scheduled term appears in zones 2, 3 and 4 of Figure 2. For display purpose only time and status are needed, as well as references to a doctor and patient. In total, the number of fields is reduced from 16 to 5. New table was named ReadScheduledTerm and appropriate object model and business logic classes are generated.

The admission form is of top importance since it is used not only for selecting the patient for the exam, but also as a display in waiting rooms, where it refreshes it content periodically.

As it has been mentioned, the users usually open patient record overview first. Figure 3 displays start page of full electronic patient record. This page contains data from ReadPatientData (zones assigned with the number 1), overall medical warnings and notices (2), insurance data (3) and data about patient's employment or, if the patient is a child, their parents employment and health risk factors (4). The most of doctors take a look initially on the data displayed here before start visit. From the admission form, doctors can immediately start new medical examination (only 7% of interviewed do this), open list with active medical treatments (29%) or open the form with full medical record (64%).


Картон за предшколску децу- PETAR RAJKOVIĆ (33)

Општи подаци | Посете | Значајни медицински подаци | Саветовалиште за одојчад | Саветовалиште за малу и предшколску децу | Завршена лечења | Систематски прегледи

1 МБГ 0504978731919

Знак упозорења 2

Број картона 184/2009 2

Здравствени картон

За предшколску децу

Напомена: neka napomena 2

Број здравствене књижице 10040711490

ЛБО 1 28300198158

Лични подаци

RAJKOVIĆ JOVAN PETAR

презиме (име једног од родитеља) и име

презиме и име лица од кога се изводи право на здрав. заштиту

05.04.1978.

дан, месец и година рођења

место становања и општина

улица и број

Телефон: Стан 122; Посао

[✓ М] пол Ж

Родитељи живе заједно [✓ ДА] НЕ

1

Категорија здравствене заштите	Од	До
1110 Lica koja po prestanku radnog odnosa ostvaruju pravo na naknadu zarade zbog povrede na radu ili prof	22.07.2011.	
1111 Lica koja imaju pravo na novčanu naknadu po osnovu nezaposlenosti	22.07.2011.	22.07.2011.
1017 Lica koja obavljaju poslove van	22.07.2011.	22.07.2011.

3

Подаци о породици

Сродство	Име	Година рођења	Занимање	Здравствено стање
majka	Milena		Učiteljica	
otac	Ivan		Ekonomista	dobro

4

Подаци о порођају и детету

Нормалан порођај	[✓ ДА]	НЕ
Конгениталне аномалије	ДА	[✓ НЕ]
Рођено у здр. организацији	[✓ ДА]	НЕ
	тежина	дужина
Статус детета	проходало	проговорило

2

Figure 3 Patient's medical record - overview on demographic data, insurance, and medical warnings – 1: data taken ReadPatientData, 2: overall medical warnings, 3: insurance related data, 4: family members' employment data

Medical warnings are taken from base medical record table and this table containing 19 columns is entirely copied into read database and named as ReadElectronicPatientRecord having total 17 columns. Some constraint columns are removed, but one additional is introduced used when our class models some specialist medical record. This field contains a digest of the data contained in the specialist record and is defined as a long text. Our system generally supports several types of specialist medical records, such are dental, gynecological, ophthalmological etc., but general medical warnings and notices are stored in mentioned base medical record that acts as a parent entity for all specialist records.

Insurance related data displayed as items in the table in zone 3 are read from 3 different tables containing total 28 columns. They are replaced by single de-normalized table (ReadInsurance) containing 6 columns. In similar manner employment related data (from zone 4) are reduced from 6 tables with total 43 columns to a single table (ReadEmployment) with 11 columns.

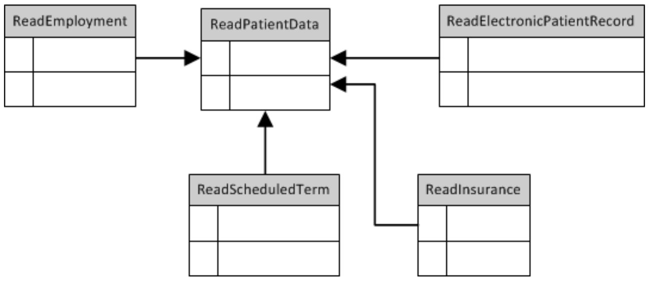


Figure 4 Initial structure of read database

On the described way, we created simple initial read database (Figure 4) containing total 5 tables having in total 54 columns and 4 constraints only instead of 17 tables with 178 columns. This is a somehow new approach since we have not moved complete query logic to new component and have not migrate all data used for read operations in new database. Also, our system support hot-swap for changing the source for read operations. Our read database currently supports just few read operation, but their share in total number of select operations is significant since they fill with data two forms that are in most frequently open forms by the users.

Table 2. Effects of using read database tables on admission form – 1: ReadPatientData, 2: ReadScheduledTerm

Read tables in use	Average response time (in seconds)	Average amount of raw data loaded from database (in KB)
None	1.243	32.296
1	0.961	21.553
1, 2	0.774	11.658

Table 3. Effects of using read database tables on electronic patient record form – 1: ReadPatientData, 2: ReadElectronicPatientRecord, 3: ReadInsurance, 4: ReadEmployment

Read tables in use	Average response time (in seconds)	Average amount of raw data loaded from database (in KB)
None	1.927	12.858
1	1.675	11.026
1, 2	1.674	11.027
1,2,3	1.509	7.182
1,2,3,4	1.416	4.971

Mentioned response time is total time needed from the point when data retrieval is requested until the visual windows form appeared. This time is used on data retrieving, processing and displaying graphic user interface. Our update is applied mainly on data retrieving level, partially on data processing, but it does not affect processes related to the user interface. Mentioned data amounts are related to raw data retrieved directly from data base without any further formatting.

Introducing read database in our system, even in such a small scale, we got improved initial system response and reduced data traffic. Looking on a global level, all processes that use visual components which corresponding business and data object model classes are replaced by those connected to read database will get some improvement.

5. RELATED WORK

The basis of the CQRS pattern is command-query segregation principle defined in Bertrand Meyers's book on object oriented software construction from 1988 [5]. With this time distance, we cannot say that this concept is new, but it is not often used as an architectural solution and there is not much of academic literature covering this topic.

From the other side, many online articles could be found arguing in favor of CQRS usage. Nevertheless, Microsoft included lately the CQRS pattern in their pattern library and published useful CQRS guide [6]. The guide [6] itself was a result of CQRS Journey [7] project conducted by Microsoft and was, as they said, "focused on building highly scalable, highly available, and maintainable applications with the Command & Query Responsibility Segregation and the Event Sourcing patterns".

The articles written by Greg Young [2], Udi Dahan [8] and Martin Fowler [9] are considered as the major sources for the CQRS pattern. The most important definitions along with recommendations and best practice are given here. We used [2][6][7][8] and [9] as a starting point in our research.

Also, we have found several interesting examples showing different area where CQRS has been successfully used. In [10] the CQRS pattern is a part of an approach for more efficient Web application development. In the paper, the authors described the solutions for eliminating fundamental challenges they have "namely object-relational impedance mismatch and the consequences of CAP theorem when scaling out" [10] by using architectural patterns, including CQRS.

Thesis [11] was important to us since there we could found detailed description of a process of "code generation within CQRS framework". The author gave also an overview on a visual CQRS workbench used for defining model that used for later code generation. Only parts of the code that should be written manually in the system described in [11] are event handlers on the query/read side. Another interesting thing in [11] was that presented case study is related to demo software that would be used in the hospital environment.

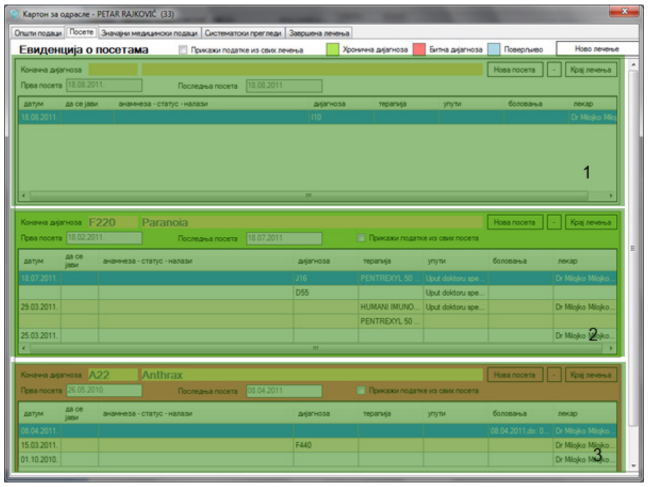
The article [12] was the closest reference to our work. The authors described the potentials of using CQRS during development of medical record based information system. The authors describe potential good sides, but also claim that "depending on the domain, segregation of command and query activities may not yield a substantial benefit".

Assuming all known pros and contras we found in the literature, we started our research with the main aim to check upon which level CQRS could be used and explore which potential benefits we can get on which cost.

6. CURRENT AND FUTURE WORK

We are working on modeling the extension for our read database that will cover the set of medical data, often used in the active treatment overview tab on electronic health record form (Figure 5). Zones marked as 1, 2 and 3 represent active medication processes. Each of these processes contains data collected during patient visits and represented in a form of grid. User can see there teaser information on identified diagnoses, prescribed therapy, and created requests for further medical examinations (Figure 6).

In total, 93% of doctors open this overview before start entering the data about new patient's visit. Another interesting fact is that already stored visits are opened very rarely. By our preliminary checks, only in 4.5% of all access to active treatment overview, users opened some of old visits to check it in details. Additionally, only 3.2% of already created visits are reopened in order to update some of entered data.



Коначна дијагноза
Прва посета 18.08.2011. Последња посета 18.08.2011
Нова посета - Крај лечења

датум	да се јави	анамнеза - статус - налази	дијагноза	терапија	упути	боловања	лекар
18.08.2011.			I10				Dr Milojko Miloj

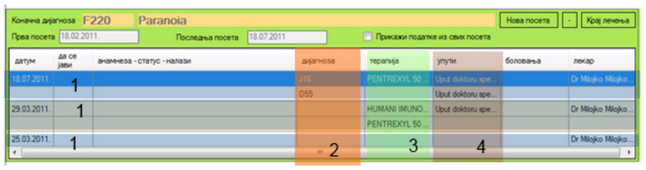
Коначна дијагноза F220 Paranoia
Прва посета 18.02.2011. Последња посета 18.07.2011
Прикажи податке из свих посета
Нова посета - Крај лечења

датум	да се јави	анамнеза - статус - налази	дијагноза	терапија	упути	боловања	лекар
18.07.2011.			J16	PENTREXYL 50 ...	Uput doktoru spe...		Dr Milojko Milojko...
			D55		Uput doktoru spe...		
29.03.2011.				HUMANI IMUNO...	Uput doktoru spe...		Dr Milojko Milojko...
				PENTREXYL 50 ...			
25.03.2011.							Dr Milojko Milojko...

Коначна дијагноза A22 Anthrax
Прва посета 26.05.2010. Последња посета 08.04.2011
Прикажи податке из свих посета
Нова посета - Крај лечења

датум	да се јави	анамнеза - статус - налази	дијагноза	терапија	упути	боловања	лекар
08.04.2011.						08.04.2011.do: 0...	Dr Milojko Milojko...
15.03.2011.			F440				Dr Milojko Milojko...
01.10.2010.							Dr Milojko Milojko...

Figure 5 Active treatments overview – zones 1, 2 and 3 represents three different active medications



Коначна дијагноза F220 Paranoia
Прва посета 18.02.2011. Последња посета 18.07.2011
Прикажи податке из свих посета
Нова посета - Крај лечења

датум	да се јави	анамнеза - статус - налази	дијагноза	терапија	упути	боловања	лекар
18.07.2011.			J16	PENTREXYL 50 ...	Uput doktoru spe...		Dr Milojko Milojko...
			D55		Uput doktoru spe...		
29.03.2011.				HUMANI IMUNO...	Uput doktoru spe...		Dr Milojko Milojko...
				PENTREXYL 50 ...			
25.03.2011.							Dr Milojko Milojko...

Figure 6 Data displayed within a single medication overview – 1: data collected during single patient visit, 2: identified diagnoses, 3: teaser info on active prescriptions, 4: teaser info on generated requests for further medical examinations

One patient visits contains information on given medical services, identified diagnoses, prescribed therapies, created requests for further medical examinations and data on created medical documents. To get data displayed in a single 8-column row in visit overview (Figure 6), data related to all mentioned entities has to be joined, loaded and processed. Behind mentioned 8 displayed rows lays the structure of 32 tables and 455 rows. The structure is filled with during different medical examinations, and all of them are available in detailed visit overview accessed in less than 5% of cases.

When we finish with this read database extension, we expect that the amount of loaded raw data from database will be reduced to less of 10% of its actual amount, while overall response time we expect to be reduced to less than one half of currently needed.

In the future we plan to improve our modeling tool that can generate components that will use data not only from the database but from the other sources, and in same time prepare data not only for our GUI, but also and for specified Web services. In the same time, we will try to develop a database watcher that will monitor activities in the database and identify the best possible structures that can be de-normalized and moved to read database.

Also, we tend to use our modeling tools not only in the scope of medical information system itself, but also as general purpose software. We perform some initial tests in this direction, and use our tools for generating parts of e-learning system dedicated to medical students and clinical staff members.

7. CONCLUSION

The overall performance problem in information systems as a result of loading large, but unnecessary amount of data is not a new problem and there is variety of very different solutions starting from definition views on the database, via intelligent object models to applied architectural patterns such is CQRS.

All of these solutions have well known advantages and drawbacks, and we wanted to focus in our research on a relatively new approach that are not widely described in scientific papers, but is generally well known at used up to some point.

In our approach we tried to create a flexible business logic layer, which parts can be overridden in the software component connected to read database. This will gave flexibility to our system which would be able to work both with and without active read database. Additionally, we did not start with a presumption that all data should be copied onto read database, but only those that frequently used and rarely changed.

By using this approach we managed to reduce overall response time in critical parts of the software by approximately 40%, while average data traffic is reduced to about one third of its original amount. By expanding the read database and increase the number of supported business processes we expect to improve overall system performance and increase customer satisfaction level.

8. ACKNOWLEDGMENTS

This paper is a result on a project dedicated to a development of collaborative e-learning tools (supported by the Ministry of Education and Science of the Republic of Serbia).

9. REFERENCES

[1] Petar Rajković, Dragan Janković, Vladimir Tošić, A Software Solution for Ambulatory Healthcare Facilities in the Republic of Serbia, 11th International Conference on e-Health Networking, Application & Services – HealthCom2009, Proceedings ISBN, 978-1-4244-5014-5, pp. 161-168, Sydney, Australia, December 2009

[2] Greg Young, CQRS Documents by Greg Young, http://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf

[3] Rajkovic, P., Jankovic, D., Stankovic, T., Tosic, V.: Software tools for rapid development and customization of medical information systems, 12th IEEE International Conference on e-Health Networking Applications and Services (Healthcom 2010), 1-3 July 2010, Lyon, France, 119-126. (2010)

[4] Stankovic T., Pesic S., Jankovic D., Rajkovic P., Platform Independent Database Replication Solution Applied to Medical Information System, ADBIS 2010, published in Springer-Verlag LNCS 6295,ISBN 978-3-642-15575-8, pp. 587-590

[5] Bertrand Meyer, Object-oriented Software Construction, Prentice Hall 1988, ISBN 0-13-629049-3.

[6] Dominic Betts, Julián Domínguez, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, Foreword by Greg Young: Exploring CQRS and Event Sourcing, http://www.microsoft.com/en-us/download/details.aspx?id=34774

[7] CQRS Journey, http://msdn.microsoft.com/en-us/library/jj554200.aspx

[8] Udi Dahan, Clarified CQRS. Dec. 2009. url: http://www.udidahan.com/2009/12/09/clarified-cqrs/

[9] Martin Fowler. CQRS, http://martinfowler.com/bliki/CQRS.html.

[10] Hendrikse, Z. W., and K. Molkenboer, A radically different approach to enterprise web application development, 2012, http://www.codeboys.nl/white-paper.pdf

[11] Fitzgerald, Seán, A pattern for state machine persistence using Event Sourcing, CQRS and a Visual Workbench, 2012.

[12] Arunava Chatterjee**,** Healthy Architectures - Using CQRS and Event Sourcing for Electronic Medical Records**,** http://www.infoq.com/articles/healthcare-emr-ehr;jsessionid=2E1EE38911CC1632B2012A572455AC2C