Using CQRS Pattern for Improving Performance in Relational Databases

Valmir Sinani
Faculty of Information and Communication Technologies
University of St. Kliment Ohridski,
1 Maj bb., 7000 Bitola, Republic of Macedonia
sinani.valmir@uklo.edu.mk

***Abstract*—Data-intensive applications, such as social media, financial and trade systems, rely heavily on data generated by users. This type of application requires high performance, and traditional architecture may not be suitable for complex applications. CQRS is a programming model pattern that changes the traditional mentality of using one programming model as CRUD, where writing and reading operation work in a single model. CQRS segregates CRUD into two programming models. CQRS pattern treats retrieving data and writing data separately with different responsibilities. Here in the introduction, some quotes describe the CQRS pattern. Also, here are some motivations for the implementation of the CQRS pattern. The Case Study section presents an implementation of the CQRS pattern in API applications with the intention of optimization. Below is presented the value of segregating the reading data model from data store.**

Keywords—CQRS pattern, CRUD, performance, use case.

I. INTRODUCTION

This paper will present one table which will be used to perform traditional functions such as Create, Update, Delete and read data using MSSQL. This table will contain rows of products and will have five columns: Id, Product Name, Producer, Price, and Description. The aim of creating this table is to test the load performance while comparing traditional CRUD with CQRS pattern. The CQRS pattern will use two different models: one model for creating, updating, and deleting, and another model for reading data only. Each model will be treated separately, as shown in Fig 1.

To test performance and show a comparison between traditional CRUD and CQRS patterns, it will present a Web API in .NET for both approaches. To evaluate the performance in data retrieval, the Apache JMeter application was used. Apache JMeter application helps in testing load performance of Web applications. More information about Apache JMeter can be read in the documentation of this site application. [8]

The CQRS pattern has various advantages and disadvantages. Some advantages include scalability, reduced single points of failure, increased security, and maintainability. However, there are also some disadvantages, such as increased complexity, weak acid pattern, and data duplication. In this paper, our focus is on improving performance.

In the next section, it will descript CQRS patter. Also, in the II section, it is presented the difference between traditional pattern and CQRS pattern.

II. CQRS PATTERN

Data-intensive applications, such as social media, financial and trade systems, rely heavily on data generated by users. This type of application requires high performance and availability to handle a large amount of data in real time. Also, this application needs to be always available to provide a seamless user experience.

The traditional architecture uses the same model for reading and writing data in relational databases. This architecture is suitable for applications that are not complex. However, if the model becomes complex and requires more focus, then the treatment of the same model for reading and writing could become challenging. Due to the complexity of reading and writing data in large applications, the extensibility of it may become difficult to manage when scaling and maintaining it.[1] Big applications could encounter load problems, as demonstrated in the example presented in this paper.

Data architects could improve performance and data availability by examining data activity distinctly in creating, reading, updating, and deleting. As a result, the data created once may be accessed multiple times by users.[2]

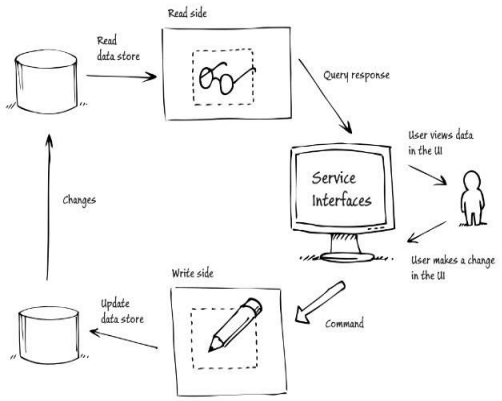


Fig 1 Command and Query models using CQRS pattern.

CQRS is a design pattern that separates the responsibilities of reading and writing data into two distinct processes. CQRS is a pattern that offers a separation between reading and writing data.

CQRS stands for Command and Query Responsibility Segregation. This pattern firstly was described by Greg Young in 2010. The base of the CQRS pattern is the segregation model for reading data from the model of retrieving data [3].

Before using CQRS pattern, it was the same object that was used for command and query. CQRS is based on segregation in two models, one model is used for reading while the other is used for query. CQRS presented here is like Meyer used in Command and Query Separation: query returns data only and doesn't change data, while command changes the state and doesn't return any data [4].

In section III, we will present a CQRS pattern, and a simple architecture prepared for this use case. Here, we have presented the infrastructure of the use case and application used for test scenarios.

III. Use Case

Below will be a product table, which will be used to analyze performance. The performance analysis will be using simple architecture or traditional CRUD and CQRS patterns. CQRS patterns will be presented in two types of data management: with a single relational database and, with two relational databases.

A. Simple Architecture and CQRS Pattern

For setting up a simple architecture was used CRUD .NET API technology. Within this project are three roots, as could see in Fig 2. The first root retrieved all data from the product table, the second root retrieved product data by id, and the last root was used to insert data in the database.

The CQRS pattern, as we can see from the acronym, separates responsibility into two models, one model for command and the other for the query. The command model operates for writing in the database, while the query model is used for reading data in database. The command model performs only one specific task; thus, its duty is to write in the database, therefore it is simple. Query models only operate to retrieve data from the database and are not indicated in the writing process. For more details about code implementation, the CQRS pattern can be seen online.[7] Also, the CQRS architecture with two databases uses the same technology as the CRUD pattern. However, the CQRS pattern with one database performs both reading and writing operations in the same database. On the other hand, the CQRS pattern that operates with two databases writes data in one database and reads from another database.

The functionalization CQRS pattern uses two databases where one database operates to create, update and delete and also could be called the primary database, and the other database is used to perform reading data which could be called a secondary database.

The primary relational database synchronizes data with the secondary relational database in real-time. The synchronization between two databases is achieved by using triggers. Triggers are responsible for the synchronization of data from the primary database to the secondary database in real time.

This test scenario aims to perform a comparison between CRUD patterns and CQRS patterns.

The testing of these patterns is conducted using the 'Apache JMeter' application. Firstly, we have created a thread group of users whose duty is to execute the group of parallel users as shown in Fig 3. This thread group was configured for our requirements. We created an HTTP Request and an HTTP Post and added 'View Result Table' to the executed results.

Fig 2 - Root API of Applications

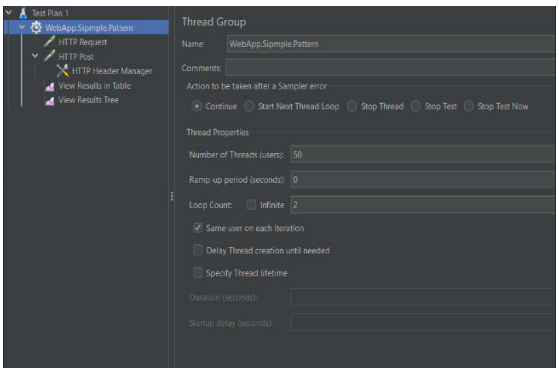


Fig 3- Apache JMeter- Test Plan Scenario

B. Architecture Testing

Testing is conducted on the product table that is created in the RDBMS. The testing is carried out on two groups of data: one with 30k data rows and the other with 100k data rows. Testing groups of users are used for testing. The first group contains 10 users, the second group contains 50 users, and the last group contains 200 users. Each user group performs 2 operations, adding new data and retrieving all products, as shown in Fig 3.

C. Quality Of Test

The infrastructure used for the test is a laptop with the following specifications: Core i7, 2.80GHz (8 CPUs), and 16GB of RAM. Before testing patterns from every user group, we ensured the eco-functioning of processes and the health of the laptop. Additionally, before executing every group, IIS and MSSQL services were started. During the testing, the user groups monitored data processing using SQL Profiler and verified the expected data to be added and retrieved.

Section IV presents the results of this case scenario. Additionally, it will be shown the another extended possible scenario and discussed future work.

IV. Current and Future Work

The results of the tests for this case are presented in the charts. The charts presented in "Fig 4" and "Fig 5" are the average taken from Table I and Table II.

Table I and Table II present average results of three groups of users who performed three different patterns of implementation.

Fig 4- Rez. testing the http get with 30,000 rows.

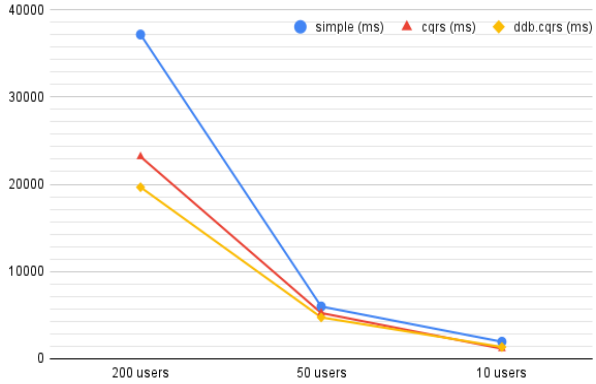
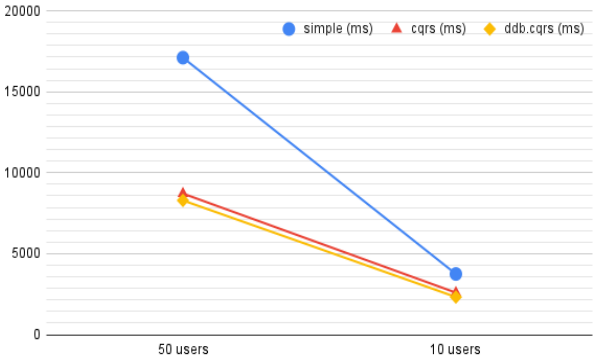


Fig 5 - Rez. testing the http get with 100,000 rows.



Each column in Tables I and II presents the results of three different application implementations: a simple CRUD implementation (labeled 'Simple(ms)'), a CQRS implementation (labeled 'CQRS(ms)'), and a distributed database CQRS implementation (labeled 'ddb.CQRS(ms)'). These applications performed operations for writing and retrieving data. Table I presents the results for parallel user groups of 10, 50, and 200, while Table II presents the results for parallel user groups of 10 and 50.

Based on test results which were presented in Table I and Table II, it appears that there are differences in SIMPLE CRUD performance. It performs well over 30k product rows for 10 and 50 parallel users, and over 100k product rows for 10 parallel users. However, performance is weak over 30k for 200 parallel users and over 100k for 50 parallel users.

Based on the results of Table I, which represents data from "Fig 4", it seems that the CQRS pattern does not have a performance impact when reading 30k product rows for a group of 10 and 50 parallel users. But the impact CQRS patter can be seen in the group of 200 parallel users. Also, based on "Fig 5", which is derived from Table II, it seems that CQRS pattern has improved performance in reading 100k product rows for the group of 50 parallel users.

Based on "Fig 4" and "Fig 5", it seems that the performance of using CQRS increases significantly as the number of rows in the database increases. Furthermore, when the number of rows exceeds 30.000 in the group of 200 parallel users, the performance of CQRS with a single database appears to be lower compared to distributed databases CQRS with two databases.

The CQRS could advance if:

- CQRS could have improved performance and other advantages if, in a distributed database environment, the CQRS implementation separates the read database onto a different machine.
- If this case study were extended to the level of table design, the table could be extended to join with the Products table. This would allow for data retrieval from multiple tables. To optimize CQRS with distributed databases, data could be written in a denormalized way.[5]
- Another optimization of CQRS with distributed databases could be achieved through horizontal scalability, which would involve having more than two read databases.
- Optimization of CQRS with distributed databases could be improved by combining it with other database systems. For example, CQRS could use NoSQL for managing the read database process.[6]

TABLE I. THE RETRIEVAL OF DATA FROM A PRODUCT TABLE CONTAINING OVER 30,000 ROWS

Reading data by users	***Simple*** *(in millisecond)*	***CQRS*** *(in millisecond)*	***DDB.CQRS*** *(in millisecond)*
10 users	3749.91	2596.21	2326.91
50 users	17106.48	8716.16	8292.16
200 users	28404.41	17952.1	17828.1

TABLE II. TESTING THE RETRIEVAL OF DATA FROM A PRODUCT TABLE CONTAINING OVER 100,000 ROWS

Reading data by users	***Simple*** *(in millisecond)*	***CQRS*** *(in millisecond)*	***DDB.CQRS*** *(in millisecond)*
10 users	3749.91	2596.21	2326.91
50 users	17106.48	8716.16	8292.16

V. CONCLUSION

In this paper is presented an importance of performance databases in SQL relation. This is a comparison between the CQRS (Command and Query Responsibility Segregation) pattern and the traditional CRUD (Create, Read, Update, and Delete) pattern. Two different CQRS solutions are presented, one with write and read operations performed in the same database and the other with read and write operations performed in different databases.

CQRS could play a significant role in improving performance in reading data in SQL relational databases. CQRS pattern could be an unavoidable solution in cases when the amount of data in relational databases is growing significantly,

and there is a frequent reading of that data. However, as this paper suggests, in some cases, CQRS may not have an impact on loading data, and traditional patterns may be sufficient.

The CQRS pattern could potentially slow down the writing process on the database, especially if the synchronization is in real-time.

REFERENCES

[1] CQRS pattern - Azure Architecture Center | Microsoft Learn." https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs. Accessed 18 Mar. 2023.

[2] Rajković, Petar, Dragan Janković, and Aleksandar Milenković. "Using cqrs pattern for improving performances in medical information systems." Proc. of the 6th Balkan Conference in Informatics. 2013.

[3] "CQRS - Martin Fowler." 14 Jul. 2011, https://martinfowler.com/bliki/CQRS.html. Accessed 21 Mar. 2023.

[4] Betts, Dominic, et al. "Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure." (2013), pp. 223-234.

[5] CQRS Journey, http://msdn.microsoft.com/enus/library/jj554200.aspx

[6] "Combining SQL and NoSQL databases for better performance." 24 Oct. 2017, https://www.thereformedprogrammer.net/ef-core-combining-sql-and-nosql-databases-for-better-performance/. Accessed 5 Apr. 2023.

[7] V. Sinani. 2023. Repository of CQRS product case. https://github.com/valmirsinani/CQRS-Pattern. Accessed 5 Apr. 2023.

[8] "Apache JMeter - Apache JMeter™ - The Apache Software Foundation!." https://jmeter.apache.org/. Accessed 5 Apr. 2023.