Asian Journal of Multidisciplinary Studies
Vol. 4, No. 2, (2021)
ISSN 2651-6691 (Print)
ISSN 2651-6705 (Online)

NoSQL: An Optimization Approach in a Stand-Alone System

***Reynan Anislag*[1]*, Jay C. Rico*[2]**
[1]*University of the Philippines-Diliman*
[2]*Pangasinan State University*

Abstract *- With some limiting factors of SQL databases, the NoSQL has been studied and is now a popular database that is being used by the largest companies in the world like Facebook and Google to work with the unstructured big data that they collect and manage every day. In a stand-alone application like the payroll system, the use of SQL databases also becomes a struggle especially when the collected data are growing rapidly. The normalization or query optimization has a little advantage while NoSQL solved this problem in a much better performance. In the new model, the NoSQL (MongoDB) database was designed to work with the transactional data, specifically during the read phase which must be taken from the transactional table. On the other hand, SQL (MySQL) database still works on the other tables that have stable attributes. The test was performed by using a stand-alone system over the usual application of NoSQL databases like the online distributed systems. The combined SQL and NoSQL databases conveyed significant results than that of the SQL database alone in terms of performance. It is evident that NoSQL works well with SQL. NoSQL would have been one of the best options to optimize the performance of the system. This is not only perfectly working with unstructured data in distributed systems but also bears a big advantage when applied to stand-alone systems especially when it collects and manages large volume of data.*

Keywords *- NoSQL and SQL databases, MongoDB and MySQL, transactional table and data, optimization, polyglot database structure*

1 Introduction

Database is very important to any computer systems and applications. It houses the data which is very important to keep for future references and use. The organization of data in databases is pivotal since storing and accessing data to and from it will be affected. SQL databases have proven its efficiency for more than decades now. However, NoSQL databases surfaced because of some limitations that SQL cannot handle efficiently.

The Westminn Construction Corporation has been using the Computerized Payroll System since April 2013. Currently, they prepare their payroll for around 500 employees on a weekly basis. Their payroll system has been implemented using MySQL as its database. Unfortunately, their payroll transaction becomes slower as the number of records in the database increases. Relational databases such as MySQL are encountering some limitations due to the unstoppable growth and behavior of data such as the retrieval and management of big databases [1].

According to [2], NoSQL is the new darling of the big data world. It came out to support what SQL databases can't or
barely handle with regards to the current characteristics of data. NoSQL systems store and manage data in ways that allow for high operational speed and great flexibility on the part of the developers [3]. SQL and NoSQL databases have been at issue of which one is the best database to use. As a result, to this discourse,

several applications tried using them both instead because of their unique advantages. Commonly, NoSQL databases have been implemented with systems that have unstructured data and the performance is notable. This type of data has been always seen in the online distributed systems such as Facebook, Google and the like where most of the research have shown the strengths of the NoSQL. In contrary, this paper explained how NoSQL is relevant in optimizing the database performance of the stand-alone applications such as the computerized payroll system of Westminn Construction Corporation. The new model used the NoSQL database to the structured data but with a highly increasing data while implementing SQL database to the other data to speed up the processing time of the payroll transaction. The researchers used MySQL (SQL) and MongoDB (NoSQL) as the databases and used VB.Net as the system's programming language.

2 Review of Related Literature

NoSQL's real motivator is "Big Data" - it is a term that describes the large volume of data both structured and unstructured that inundates a business on a day-to-day basis [4]. NoSQL database is a perfect match with systems that have large volume of data. Due to its schema less structure and its capability to store all the data in one place using the document-based databases such as MongoDB, the access of data becomes seamless and fast.

MongoDB is mostly closed to the relational use cases scenario. It is perfectly aligned for reliability, durability and read-oriented use cases among other popular NoSQL databases. NoSQL databases are commonly divided into read and write optimization categories. MongoDB takes advantage on an optimized read side which is the problem with relational databases because data loading and organization become its bottleneck [5]. MongoDB support is not a big problem anymore since there are many research communities now arising for those who are interested with the NoSQL databases. [6] used the write-side and read-side databases in his application. The write-side database used the relational SQL database because of its high support to data integrity while in the read-side database, the NoSQL database was used because of its support to speed and scalability. The write-side performs writing on the SQL and also feeding the data to the NoSQL that is ready to be displayed on the user. This is what he called the CQRS architecture using a polyglot database structure which is very essential to businesses that require more reads than writes.

The Sage Group has also already integrated MongoDB into their latest release of Enterprise Resource Planning solutions with a more fulfilling customer experience. It makes the ERP solutions adaptive to the changes and encourages innovation in the future for more competitive software products [7].

This research had focused on the real application of NoSQL specifically in the online and distributed systems with unstructured data where in fact, NoSQL could be also a good choice to be implemented in stand-alone systems with highly increasing data. But since NoSQL has its own limitations as well, combining it to SQL database is a better choice.

3 Methodology

3.1 The Existing Model

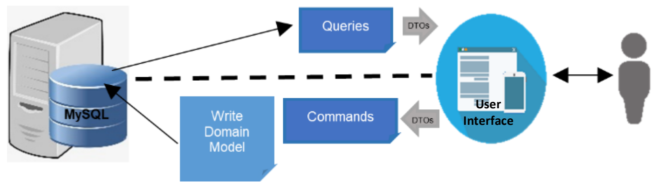


Figure 1*. It shows the existing flow of a process when the user interacts with the Transaction module of the application.*

The life cycle of a process in the existing system demonstrates the usual flow of a query.

As displayed in Figure 1, the process starts when a user makes a request through the User Interface. Doing so, the application creates a Data Transfer Object (DTO). This is when the end user hits the Transaction button on the application. Consequently, the following events are executed.

First, the SQL query is dynamically built on the application itself. Its dynamic behavior depends on the passed parameters. On the existing setup, the default parameters' intention is to get the current transactions that are created on a per day basis. The created DTO serves as the parameter of the request. The application then manipulates the DTO to form an SQL command. This command together with the DTO constitutes the Write Domain Model.

After producing the said model, the application sends it to MySQL to fetch the filtered records according to the set parameters. [8] is responsible for passing and writing the said command on MySQL. The application waits while the query is being executed on the database. The existing codes showed that waiting is synchronous. It must get a response from MySQL first before it continues to execute the remaining lines of code. Additionally, the application used the default 30 second timeout before it issues a timeout error.

Once the application already gets a response from MySQL, each record in the result set is iterated. Each iteration means an insert to the list view control on the application. This process shows how the application encapsulates the resulting query back into a DTO so that it can be translated properly to the User Interface. At the same time, the user can understand the results on his end.

3.2 The New Model

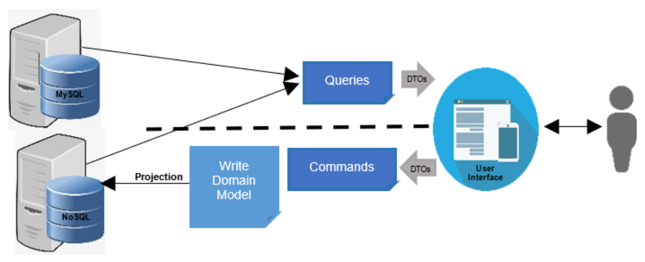


***Figure 2**. It shows the new flow of a process using the proposed optimization technique.*

In the new model, the life cycle of a process as shown in Figure 2 is the same except that the Write Domain Model is now passed to a new database, MongoDB. Unlike the existing model, this new model has two data resources involved. MySQL which contains the master data and MongoDB which contains the transactional data are utilized on this proposed optimization technique. Series of existing events are revised which considers the overhead in fetching the isolated transactional data from an additional database resource.

The user makes a request through the User Interface once the *Transaction* button is clicked. The application still generates a dynamic SQL query. This time, the query statement does not include the transactional table since transactional data is already transferred to MongoDB.

The request is translated into a DTO to be used in executing a command. This serves as a parameter to the generated query statement. The Write Domain Model created from here using the command is written and sent by the application to NoSQL where all transactional requests are saved.

As soon as the request gets executed, an additional event that is to fetch the current transactions generated for the day from MongoDB is executed. This behavior is the counterpart of the default parameters' intention that is discussed in the existing model. MongoDB's result set is saved in a collection which will be used later. This is to make sure that we only have a single database hit for this purpose.

At this point in time, we already have the result sets of both MySQL and MongoDB. MySQL holds the master data (employee list) while MongoDB holds the transactions (computed salary) created for the day. Iteration on the master data comes next. Each iteration means searching data from the said MongoDB generated collection. Whether a matching data is found or not, it also means an insert to the listview control on the application. When there is a matching data, the last few columns of the list view control are filled with the data from the collection. Otherwise, the application will leave these columns empty.

In a nutshell, the application takes good care in joining the data. And the joined data is formed to match the existing DTO so that this proposed optimization still matches the existing User Interface where end users interact with the system.

3.3 The Existing Database Query

```
SELECT oel.ID_NUMBER,
    oel.LAST_NAME,
    oel.FIRST_NAME,
    ojl.JOB_DESCRIPTION,
    oec.RATE_PER_DAY,
    otl.NUM_OF_DAYS,
    otl.NUM_OF_OT_HOURS,
    otl.SALARY
FROM  OIC_EMPLOYEE_LIST oel
    INNER JOIN OIC_EMPLOYEE_CROSSREF
AS oec
        ON      oec.EMPLOYEE_ID      =
oel.ID_NUMBER
         AND oec.EMP_STAT = '1'
    INNER JOIN OIC_SITE_LIST AS osl
        ON osl.SITE_ID = oec.SITE_ID
    INNER JOIN OIC_JOB_LIST AS ojl
        ON ojl.JOB_CODE = oec.JOB_CODE
    LEFT                              JOIN
OIC_TRANSACTION_LIST_COPY otl
        ON      otl.EMPLOYEE_ID      =
oel.ID_NUMBER
         AND   CAST(otl.TRANS_DATE   AS
date) = CAST(NOW() AS date)
WHERE  oel.ID_NUMBER LIKE '%%'
    AND oel.LAST_NAME LIKE '%%'
    AND osl.SITE_DESC LIKE '%'
ORDER  BY oel.ID_NUMBER
LIMIT  500
```

The existing database query has been the subject for optimization. Upon evaluation, the application always considers five tables in generating the query. These tables are the following:

- **OIC_EMPLOYEE_LIST** – Main table. This contains the list of employee details.
- **OIC_EMPLOYEE_CROSSREF** – It is an extension of the main table that contains the payroll related fields. These fields are referencing to other tables like site and group number IDs.
- **OIC_SITE_LIST** – It is the source of site names, and it is also used for filtering sites.
- **OIC_JOB_LIST** – It is the source of job descriptions, and it is also used for filtering jobs.
- **OIC_TRANSACTION_LIST** – It is the source of transactions including its historical data. It is filtered to current day on the query.

OIC_EMPLOYEE_CROSSREF, *OIC_SITE_LIST* and *OIC_TRANSACTION_LIST* have one-to-one relationship to *OIC_EMPLOYEE_LIST* . Thus, these are [9] INNER JOIN'ed to the main table using their primary keys such as *EMPLOYEE_ID*, *SITE_ID* and *JOB_ID* respectively. Aside from filtering the primary key, *OIC_EMPLOYEE_CROSSREF* is added with an additional condition that is to get active employees only.

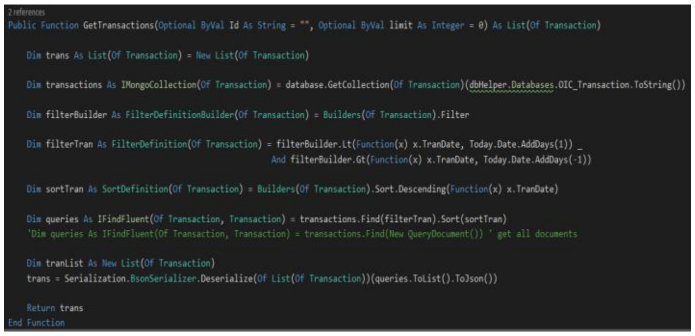
Moreover, *OIC_TRANSACTION_LIST* is [10] LEFT JOIN'ed to the main table using the *EMPLOYEE_ID* as the primary key. Since this contained the transactions created for the day including the historical data or transactions created previously, the filter must include an additional condition which is to get the current transaction out from the transactions created for an employee since the start of his stay in the company. The attachment of this transactional table to the main table provides an expensive process by querying from a large table on an employee-based filtering.

3.4 The New Database Query

```
SELECT oel.ID_NUMBER,
    oel.LAST_NAME,
    oel.FIRST_NAME,
    ojl.JOB_DESCRIPTION,
    oec.RATE_PER_DAY
FROM   OIC_EMPLOYEE_LIST oel
    INNER JOIN OIC_EMPLOYEE_CROSSREF
AS oec
            ON         oec.EMPLOYEE_ID         =
oel.ID_NUMBER
                AND oec.EMP_STAT = '1'
    INNER JOIN OIC_SITE_LIST AS osl
            ON osl.SITE_ID = oec.SITE_ID
    INNER JOIN OIC_JOB_LIST AS ojl
            ON ojl.JOB_CODE = oec.JOB_CODE
WHERE  oel.ID_NUMBER LIKE '%%'
    AND oel.LAST_NAME LIKE '%'
    AND osl.SITE_DESC LIKE '%'
ORDER  BY oel.ID_NUMBER
LIMIT  500
```

The existing database query has been revised to remove the connection of *OIC_TRANSACTION_LIST* table to other tables. This means that the non-transactional tables are kept as they are. Same primary keys are still used as reference keys to the main table.

The assumption of having the new database query is that, the historical data from the transactional table which is *OIC_TRANSACTION_LIST* is already exported to MongoDB. Furthermore, the system has to issue a query to this database aside from executing a separate MySQL query in order to get the transactional details on a per employee basis. *(See Figure 3).* It may sound expensive also but MongoDB by its nature, will take care of the performance in doing so.



```
2 references
Public Function GetTransactions(Optional ByVal Id As String = "", Optional ByVal limit As Integer = 0) As List(Of Transaction)

    Dim trans As List(Of Transaction) = New List(Of Transaction)

    Dim transactions As IMongoCollection(Of Transaction) = database.GetCollection(Of Transaction)(dbHelper.Databases.OIC_Transaction.ToString())

    Dim filterBuilder As FilterDefinitionBuilder(Of Transaction) = Builders(Of Transaction).Filter

    Dim filterTran As FilterDefinition(Of Transaction) = filterBuilder.Lt(Function(x) x.TranDate, Today.Date.AddDays(1)) _
                                                         And filterBuilder.Gt(Function(x) x.TranDate, Today.Date.AddDays(-1))

    Dim sortTran As SortDefinition(Of Transaction) = Builders(Of Transaction).Sort.Descending(Function(x) x.TranDate)

    Dim queries As IFindFluent(Of Transaction, Transaction) = transactions.Find(filterTran).Sort(sortTran)
    'Dim queries As IFindFluent(Of Transaction, Transaction) = transactions.Find(New QueryDocument()) ' get all documents

    Dim tranList As New List(Of Transaction)
    trans = Serialization.BsonSerializer.Deserialize(Of List(Of Transaction))(queries.ToList().ToJson())

    Return trans
End Function
```

Figure 3*. It shows the codes on how the transactions are fetched from MongoDB. The results are put in a collection list or a hash table afterwards.*

Once the application issues a query to get transaction details for an employee, it will create a [11] query document, a JSON formatted query whose *Sort* and *Filter* operations are defined first. These are then attached to a *Find* operation. The Find, Filter and Sort correspond to SELECT, WHERE and ORDER BY clauses in MySQL respectively. This query document then gets executed in MongoDB.

4 Results and Discussions

During the data gathering phase, researchers had extracted the current records of the company's database. It was found out that based on the last database extraction that contains six-month worth of data, 58% of the company's MySQL database comprises the transactional data which is rapidly growing. This is because the company had been processing the payroll every week. Weekly processing means a production of a record for every employee. Around 500 records are created every week for the transactional table while other tables are dependent to the incoming and outgoing employees to and from the company. The transactional table has the following attributes:

TRANS_ID	EMPLOYEE_ID	NUM_OF_DAYS	NUM_OF_OT_HOURS	SALARY	TRANS_DATE

These data are structured. However, the problem is that the transactional table always becomes almost full even in a short span of time. Approximately, there are currently 500 employees which corresponds to 500 payroll transactional data per week. On the following week, another 500 records will be created which are stored on the same table which means that there is a total of 1000 records available in the database. This follows that the number of records in the transactional table is growing by: $\sum_{i=1}^{n} x_i$ where x=500 and i=1 corresponds to a week. This is voluminous which makes the read of the data from the database slow.

The current model uses a query statement that is joining every related table. During the analysis phase, the researchers realized that the transactional table has data that is quite huge and grows at a great rate. What makes it poor is that the transactional table is used as a sub table in the query statement. An additional performance overhead on filtering from the transactional table before it joins with the main table makes it expensive on the entirety of the query execution.

The new model minimized the performance overhead by removing the transactional table from the existing query statement. A significant increase in the performance is noticed during the query execution. Since transactional table is exported to MongoDB, a procedure in getting the transactional data is added to the existing flow. Performance wise, this procedure adds up performance overhead but the summation of performance in using both databases is still faster than the existing scenario which is using MySQL alone. The application writes a BSON (Binary JSON) query document and passed it to MongoDB. And by MongoDB's NoSQL nature, the execution is quite impressive. Moreover, the application takes only a single database hit. The result of the query document is saved on the memory through collection document so that during iteration of master data, reconciliation of its transactional records would only access the memory which is all we know that it is faster than performing another MongoDB hit for every master datum. It only requires linear time in accessing the transactional data since we already have a collection or a hash table.

4.1 Comparative Results of the Response Time using MySQL and MySQL+NoSQL Databases

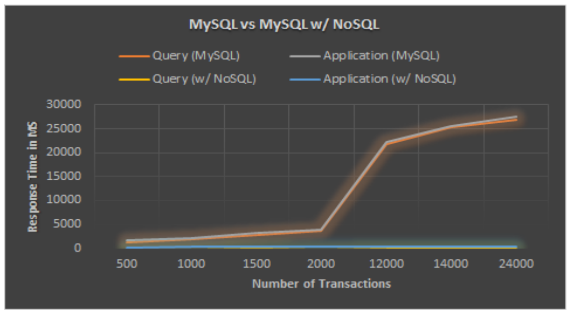


Figure 4*. It shows the comparison of response times generated from executing the query and the application of both implementations. One is using MySQL database alone and the other one is using both MySQL and NoSQL databases.*

The existing application that harnessed MySQL database alone depicted an expensive database utilization. As shown, both query and application performance took at least a second in performing a task. As the number of records increases, the execution time increases significantly. This means that as the payroll is performed over the year, end users will suffer much on this behavior.

In the proposed optimization technique, a remarkable enhancement on the performance was recorded. According to the figure, the time frame and the number of transactions do not necessarily affect the response time. All rows

Compared to the existing implementation, the row with the least number of transactions recorded more than a second response time while the average response time of the proposed implementation still runs in less than half a second. This only means that the optimization was effective as the performance remained consistent across the given number of transactions. Unlike the existing one, the increase of response time is very much noticeable as the number of records increases. This registered an average of 12 second response time.

5 Conclusion

Even in the stand-alone systems, a great growth rate of data in SQL databases becomes a bottleneck in the performance of the system. This means that SQL alone could not do the work well done anymore. This has been an observation from the payroll system in which we thought ~~it~~ is a small system but rather becomes bigger in terms of the transactional data it generates and collects. As a solution, the researchers considered separating the transactional records from MySQL which is causing the application to perform very slow. Additionally, a NoSQL MongoDB database, a free and open-source cross-platform document-oriented NoSQL database has been selected to be used as a database for the registered less than half a second response time. transactional records of the system because of its high scalability and a best fit for Big Data related projects. A comparative analysis of the response time of data that are produced with the existing implementation that used MySQL alone and with the new implementation that used both MySQL and NoSQL databases shown from the results that the idea on a polyglot (integrating different databases into one system) type of a database structure as an optimization approach to stand-alone system is highly recommended. This affirmed that the use of NoSQL and MySQL databases combined in a stand-alone system was also benefitted and it is not only to be used with the unstructured data and online distributed systems.

References

[1.] Shetty, Deepika V. and Chidimar, Sana J. (2016). ***Comparative Study of SQL and NoSQL Databases to evaluate their suitability for Big Data Application.*** International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online) Vol. 4, Issue 2, pp: (314-318). www.researchpublish.com

[2.] Bhatia, Richa (2017). ***NoSQL vs. SQL - Which Database Type if Better for Big Data Applications***. Analytics India Magazine. https://analyticsindiamag.com/nosql-vs-sql-database-type-better-big-data-applications/

[3.] Yegulalp, Serdar (2017). ***What is NoSQL? NoSQL databases explained.*** InfoWorld.com. https://www.infoworld.com/article/3240644/nosql/what-is-nosql-nosql-databases-explained.html

[4.] Ashwinii, Amit (2017). ***Should You Use NoSQL or SQL Db or Both***. https://www.cognitiveclouds.com/insights/should-you-use-a-nosql-db-sql-database-or-both/

[5.] Lourenco, Joao Ricardo et. al. (2015). ***NoSQL Databases: A Software Engineering Perspective.*** Advances in Intelligent Systems and Computing 353:741-750 DOI 10.1007/978-3-319-16486-1_73

[6.] Smith, Jon P. (2017). ***EF Core – Combining SQL and NoSQL databases for better performance***. https://www.thereformedprogrammer.net/ef-core-combining-sql-and-nosql-databases-for-better-performance/

[7.] www.mongodb.com. ***Sage Upgrades Sage ERP X3 Experience with MongoDB***. https://www.mongodb.com/press/sage-upgrades-sage-erp-x3-experience-mongodb

[8.] Microsoft Developer Network. ***OdbcDataAdapter Class.*** https://msdn.microsoft.com/en-us/library/system.data.odbc.odbcdataadapter(v=vs.110).asx

[9.] w3resource. ***What is INNER JOIN in MySQL?*** https://www.w3resource.com/mysql/advance-query-in-mysql/inner-join-with-multiple-tables.php

[10.] w3resource. ***What is LEFT JOIN in MySQL?*** https://www.w3resource.com/mysql/advance-query-in-mysql/left-join.php

[11.] mongoDB|Documentation. ***Query Documents*** https://docs.mongodb.com/manual/tutorial/query-documents/