

Increasing performance in large capacity databases

Eliona SKENDERAJ

EUROPEAN UNIVERSITY OF TIRANA

Abstract

The daily activity of companies is closely related to data. Databases are growing and demand for data per unit of time is increasing and high performance is required to meet this demand. Big data is a collection of large volumes of complex data that exceeds the processing capacity of traditional database architecture. They are also so difficult to administer and organize, and for this reason, the best aim is to find the best and most effective solution. The purpose of this work is to identify the main performance issues that appear while working with databases with large capacities and to propose solutions for such cases. DBMS systems have implemented techniques and tools that monitor data activity and help to improve performance. One of the main directions that focuses on the work of the topic is that of writing requests “query design” in such a way that regardless of the size of the data required, these requests are not delayed and fail. The goal is that, regardless of the proposed writing method, it provides recommendations that will be practically applied in a data warehouse system. In this way, the benefit in performance will be compared. In the end of this paper, it was possible to find very effective and clear solutions.

Key words: Database, SQL, query, performance, large capacities

Introduction

Nowadays data is everywhere. We are generating a large amount of data. The growth of data is surprising in how deeply it affects businesses. For years, companies have been using their transactional data to make informed business decisions. Decrease in the cost of storage and computing power has made companies interested in storing user-generated content in social networks, e-mail, sensors, photos, and incoming message servers that can be used for information. useful. Traditional database management systems, such as relational databases, proved to be good for structured data, but in case of semi-structured and unstructured data it breaks down. However, data comes from different data sources in different formats, and most of this data is unstructured or semi-structured. Additionally, database systems are also pushed to their storage capacity limit. As a result, organizations are struggling to extract useful information from the unpredictable explosion of data captured from inside and outside their organization. This explosion of data is referred to as “big data”.

Big data is a collection of large volume of complex data that exceeds the processing capacity of traditional database and data warehouse technologies do not support billions of rows of data and cannot effectively store unstructured and semi-structured data. From 2005 to 2023, the amount of information created and copied in the world will increase by a factor of 300, from 130 exabytes to 40,000 exabytes (more than 5200 gigabytes for every man, woman and child) (IDC Digital Universe, 2020). Big Data technologies describe a new generation of technologies and architectures to extract economic value from very large volumes of wide variety of data, enabling high speed capture, discovery and analysis. The McKinsey Global institute estimates that data is growing at 40% for year and this percentage will grow further. To address this challenge, we must choose an alternative way to process data.

Google was the first that included MapReduce structure computing, The Google File System (GFS), and closed distributed services. Amazon created a new moment in the big data storage space. Over the past few years resource tools and technologies including Hadoop, HBase, MongoDB, Cassandra, Storm and many other projects have been added to the big data space.

Description of the problem

The fast growth and the complexity of data in various fields, such as scientific research, business analysis and online platforms, has presented an

important challenge in the efficient management and analysis of large-scale data. To address this challenge, there is a need to explore ways to improve the performance of data processing and analysis on big data platforms. The problem that we are facing is to improve the performance of data operations on large datasets. Traditional methods and tools often struggle to handle the volume, velocity and type of big data, leading to problems such as slow query speed, high latency, and dissatisfaction. As a result, organizations and researchers face different types of problems such as problem in finding valid notes and taking informed decisions. This problem requires the identification of innovative techniques and strategies that can optimize the performance of big data processing. Solutions may include advances in hardware infrastructure, parallel computing, data sharing, indexing, and algorithmic optimizations. Aim is to develop approaches that can effectively handle large data sets and improve the speed and efficiency enabling users to obtain timely and accurate results from their data analyses. The challenge to improve performance in big data processing has consequences in different sectors. Improving efficiency in data operations can empower business to get.

The aim of the work

The aim of this study is to propose techniques that increase the performance of data warehouse systems.

If we look at the challenge of how to improve performance this is one of the most important things in big data and that has important consequences in various sectors. Improving efficiency in data operations can empower businesses to make important decisions for their success.

Objective

The objective of this paper is to demonstrate that:

- The large size of database affects the efficiency of database.
- The large size of the database has a positive effect on increasing the speed and the performance of databases.
- Improved performance of big data databases brings relief to complex statistic reports that produce complex analysis for businesses with high data flows.
- There is a fair relationship between performance increases in large databases with advances in technology.

Hypothesis

The use of clustered index and non-clustered index increases the performance of data warehouse systems.

Performance analysis of indexes in MSSQL Server

In this Fraction we will show the basics of MSSQL Server for using pages, B-tree, clustered and non-clustered index. To show all of this we will use tempdb database and create an index on the table.

Use tempdb Go

-- B+ Trees në SQL Server

CREATE TABLE Indexing (ID INT IDENTITY (1,1), Name CHAR (4000), Company CHAR(4000), Pay INT) .

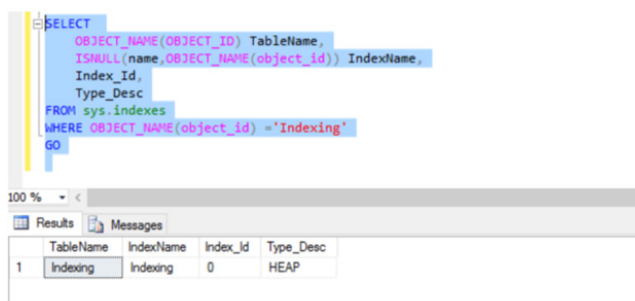
This table has 4 columns defined, ID INT as a column and two data char 4000 with column names “Name” and “Company” in this particular table. This means that we have one row, which is close to 8000 bytes. This ensures that each row is in one page and a page cannot contain more than one row. This script is executed and the table is created.

Definitions

```
SELECT OBJECT_NAME(object_id) TableName, ISNULL(name,OBJECT_NAME(object_id)) IndexName, Index_Id, Type_desc FROM sys.indexes WHERE OBJECT_NAME(object_id) = 'Indexing' GO
```

Let’s go ahead and use sys.indexes DMV. This tells us that the index of the table is Heap.

FIGURE 1. SQL Server: Index Heap



As we can see in the figure above, INT ID is zero. As we defined earlier if INT ID is zero it means it is a heap.

To continue with the demonstration, we need to establish a condition: SET NOCOUNT ON. No-count on is used to stop displaying messages for rows affected by changes and this comes as part of the SQL Server Management studio output.

Let's add some values to the table.

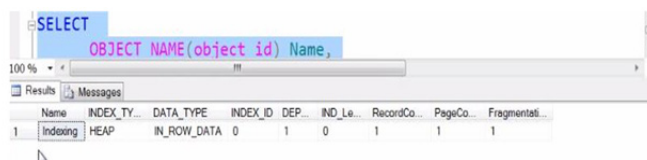
```
INSERT INTO indexing VALUES ('Eliona', Information technology ,10000)
```

Nodes will be created in this table. Now we use status check DMV which gives us the opportunity to see object, type of indexes, levels, if it is root, intermediate nodes and leaf node levels. We will also look at page count and fragmentation if it is needed and this is given by the DMV of DB index physical status.

- Status check (DMV of DB index physical stats)

```
SELECT
OBJECT_NAME(object_id) Name,
Index_type_desc AS INDEX_TYPE,
Alloc_unit_desc as DATA_TYPE,
Index_id as INDEX_ID,
Index_depth AS DEPTH,
Index_level AS IND_Level,
Record_count AS RecordCount,
Page_count AS PageCount,
Fragment_count AS Fragmentation FROM sys.dm_db_index_physical_stats(DB_ID(), OBJECT_ID('Indexing'), NULL , NULL , 'DETAILED');
GO
```

FIGURE 2. SQL Server: Indexing Types and Levels (DMV)



	Name	INDEX_TY...	DATA_TYPE	INDEX_ID	DEP...	IND_Le...	RecordCo...	PageCo...	Fragmentati...
1	Indexing	HEAP	IN_ROW_DATA	0	1	0	1	1	1

In this case you are seeing that the indexing of the table is heap and contains in it 'in row data', which means it can fit within a page that's why it's inside the row. It has a depth level equal to one and a record count equal to one. Later we will go ahead and add two more rows to particular table.

```
INSERT INTO Indexing Values
```

```
('Arild','Manolia',15000),
```

```
('Ardit','NetTrade',13000)
```

```
GO
```

After adding the rows again, let's check the status using the same DMV.

FIGURE 3. SQL Server: Status check

Name	INDEX_TY...	DATA_TYPE	INDEX_ID	DEP...	IND_Le...	RecordCo...	PageCo...	Fragmentati...	
1	Indexing	HEAP	IN_ROW_DATA	0	1	0	3	3	3

We get the same data again, but the number of records has changed.

```
INSERT INTO Indexing  
VALUES  
(‘Albano’,‘NetTrade’,11000)  
GO 100
```

FIGURE 4. SQL Server: DMV with 100 additional records

Name	INDEX_TY...	DATA_TYPE	INDEX_ID	DEP...	IND_Le...	RecordCo...	PageCo...	Fragmentati...	
1	Indexing	HEAP	IN_ROW_DATA	0	1	0	103	103	5

Now we need to create a clustered index to not have again INDEX_ID equal to zero.

```
- clustered Index  
CREATE clustered INDEX CI_IndexingID ON Indexing (ID) GO
```

FIGURE 5. SQL Server: Creating the cluster index

```
SELECT  
OBJECT_NAME(object_id) Name,
```

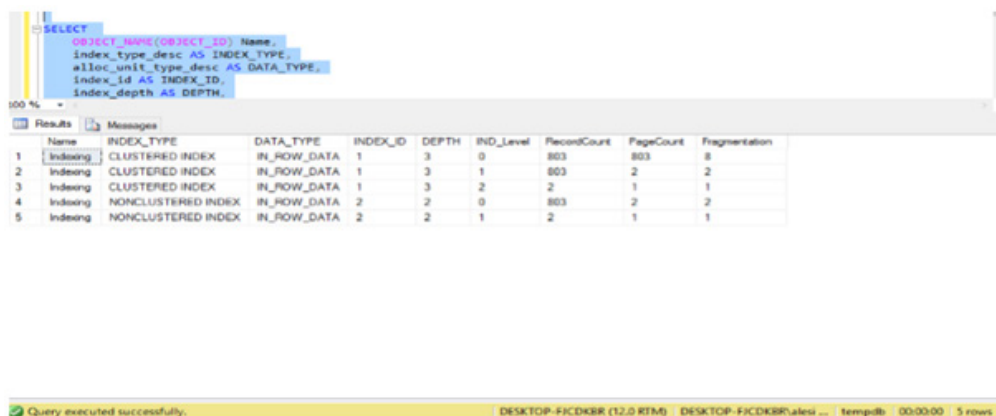
Name	INDEX_TYPE	DATA_TYPE	INDEX_ID	DEP...	IND_Le...	RecordCo...	PageCo...	Fragmentati...	
1	Indexing	CLUSTERED INDEX	IN_ROW_DATA	1	2	0	103	103	1
2	Indexing	CLUSTERED INDEX	IN_ROW_DATA	1	2	1	103	1	1

What we notice is that in our table we have created an index of type clustered index and it is defined in row data. In it we have index_ID equal to a Depth_level equal to two. At IND_level equal to zero we have 103 records and the number of pages is 103. To manage these 103 records we have another page, which has these 103 records but on a single page. Intermediate root nodes and the root node mentioned earlier is exactly what is being shown. This tells us so clearly the structure of B-tree.

Performance in Non-clustered Index

Non-clustered index is a special type of index that rearranges the way data is recorded in the table. Therefore, the table can only have one cluster index.

FIGURE 6. Table with one cluster index



```
SELECT
OBJECT_NAME(OBJECT_ID) Name,
index_type_desc AS INDEX_TYPE,
alloc_unit_type_desc AS DATA_TYPE,
index_id AS INDEX_ID,
index_depth AS DEPTH.
```

	Name	INDEX_TYPE	DATA_TYPE	INDEX_ID	DEPTH	IND_Level	RecordCount	PageCount	Fragmentation
1	Indexing	CLUSTERED INDEX	IN_ROW_DATA	1	3	0	803	803	8
2	Indexing	CLUSTERED INDEX	IN_ROW_DATA	1	3	1	803	2	2
3	Indexing	CLUSTERED INDEX	IN_ROW_DATA	1	3	2	2	1	1
4	Indexing	NONCLUSTERED INDEX	IN_ROW_DATA	2	2	0	803	2	2
5	Indexing	NONCLUSTERED INDEX	IN_ROW_DATA	2	2	1	2	1	1

Query executed successfully. DESKTOP-FICDK8R (12.0 RTM) DESKTOP-FICDK8R\atlas... tempdb 00:00:00 5 rows

The DMV status check comes up with the non-clustered index in columns 4 and 5. It is interesting to note that it is slightly different from the clustered index. You can see that the 803 records are stored on two pages, and we have one more root page which stores two pages which store the 803 records on the page. Non-clustered index stores only the key and not the data.

Conclusions

The increase of performance in databases with large capacities contains inside many directions to make possible the increase in performance. Here we can include hardware architecture construction, database architecture construction, improvement technique from the smallest data to table level till up to database level. The main purpose of this topic was to propose different ways of writing the index, different techniques and developing controls, which when applied to a data warehouse system bring an increase in performance. The first concept was that of clustered index and non-clustered index. Clustered index means that a table is physically stored according to the order of the specified index. While the concept of non-clustered index is something different. Non-clustered index is a logical index that tries to group according to a logical relation. The difference between them is

that clustered index can be created just one time and non-clustered index can be created so many times. At the end we conclude that: the indexes are absolutely necessary for modeling and speeding up a database because of their architecture in tree-form. The second concept was that of index performance maintenance. This concept is related to index duplication, how it could be detected, and what disadvantages it brought us. What this duplicated index brings is the anomalies through which the database system would go, bringing a big loss in SQL Server, problems in processing, causing poor database performance. The difficulty of maintaining them is that SQL Server itself cannot prevent the re-creation of such an index because you can create one with a different name.

References

- Bernhardt, L.V. (2015). *Data, data everywhere: Bringing All the Data Together for Continuous School, second edition Improvement*. Routledge.
- Carter, P. (2011). *Big data analytics: future architectures skills and roadmaps for the cio*. International Data Corporation.
- Criticism, N. (2003). *Data are Everywhere. Narrative Analysis: Studying the Development of Individuals in Society*, p. 63.
- Dumbill, E. (2012). *Planning for Big Data*. O'Reilly Media, Inc.
- Han, J., Haihong, E., Le, G., Du, J. (2011). Survey on NoSQL database.6th International Conference on Pervasive Computing and Applications (ICPCA), pp. 363–366.
- Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and A. H. Byers, H.A. (2011). *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute, pp. 1–137.
- Marz, N., & Warren, J. (2015). *Big Data Principles and best practices of scalable realtime data systems*. First edition, Manning Publications.
- Tripp, L.T. (2023). *SQL Server: Indexing for Performance*. Available at: <https://www.pluralsight.com/courses/sqlserver-indexing-for-performance>
- Randal, P. (2023). *SQL Server: Index Fragmentation Internals, Analysis, and Solutions*. Available at: <https://www.pluralsight.com/courses/sqlserver-index-fragmentation-internals-analysis-solutions>
- Shashanka, M. & Giering, M. (2009). Mining Retail Transaction Data for Targeting Customers with Headroom-A Case Study. *Artificial Intelligence Applications and Innovations III*, pp. 347–355.

