http://doi.org/10.35784/iapgos.2056

OPTIMIZATION IN VERY LARGE DATABASES BY PARTITIONING TABLES

Piotr Bednarczuk
University of Economics and Innovation in Lublin, Institute of Computer Science, Lublin, Poland

Abstract: *Very large databases like data warehouse slow down over time. This is usually due to a large daily increase in the data in the individual tables, counted in millions of records per day. How do we make sure our queries do not slow down over time? Table partitioning comes in handy, and, when used correctly, can ensure the smooth operation of very large databases with billions of records, even after several years.*

Keywords: partitioning, data warehouse optimization, billions of records, AdventureWorksDW

OPTYMALIZACJA W BARDZO DUŻYCH BAZACH DANYCH POPRZEZ PARTYCJONOWANIE TABEL

Streszczenie: *Bardzo duże bazy danych typu hurtownie danych z czasem zwalniają. Przyczyną zazwyczaj jest duży dzienny przyrost danych w pojedynczych tabelach liczony w milionach rekordów. Co sprawić aby z czasem nasze zapytania nie działały wolniej. Z pomocą przychodzi partycjonowanie tabel, które użyte w prawidłowy sposób może zapewnić sprawne działanie bardzo dużych bazy danych z miliardami rekordów nawet po kilku latach.*

Słowa kluczowe: partycjonowanie, hurtownie danych, miliardy rekordów, AdventureWorksDW

Introduction

Optimizing very large databases is never a simple matter. By the term “large database”, I mean databases with tables that contain hundreds of millions of records, or even billions of records. Such large numbers of records in single tables are often found in data warehouses or BIG DATA databases [1][3]. The administrators of such databases often face a dilemma as to which solution to apply; which solution will bring the best results both when it comes to the look-up time of such large tables, while maintaining acceptable writing times to these tables. This directly translates into the speed of reporting operations in a given system, which is important for the managerial staff using these reports. The expected reporting times in commercial solutions are a few seconds, and often a database response time extending to minutes is unacceptable. In addition, the question arises: how do you make sure that as the data in the database increases over time, the reports execute just as quickly as when the system was implemented?

One of the solutions proposed by database providers is table partitioning [7]. The following article presents a practical solution to the use of table partitioning even with a billion records. The essence of the problem was included in the research, where it was shown that when using this type of solution, the size of the tables does not affect the execution times of queries.

1. Table partitioning

Table partitioning, in a nutshell, is the division of tables into arbitrary parts constituting some separated ranges of data, e.g.: monthly, quarterly or annually. This division takes place in the database files, but the operator sees them as one object; one table. Although it is always possible read the contents of each partition by using the appropriate tags, a standard select from the partitioned table will return the result in the form of one set of records [2][6].

As the database is in use over a long period of time, the number of records increases up to a number (hundreds of millions, billions) which directly affects the time of performing queries on this database. Partitioning is mainly done to minimize the impact of the database size on the query speed. This is achieved by executing the query only on the records in selected partitions. In this way, regardless of the increase in the number of records in the table, the number of records that the query will be performed on is almost always significantly smaller than the total size of the table because it will usually relate only to a limited number of partitions, e.g. the last 3 days stored in one partition. This is the main advantage and benefit of data partitioning.

2. Technical implementation of partitioning

To start the implementation of partitioning in our database, it must first create a partitioning function [4]. For this purpose, it is worth doing a short analysis of what periods of data we usually perform data analysis on for reporting purposes. In my experience, there are three such most common reporting periods: yearly, quarterly, and monthly, and in 80% of cases, it is monthly. This is why the monthly partitioning function was selected for the research, as can be seen in listing 1.

Listing 1. A monthly partitioning function

```
create partition function PartitionFunctionByMonth (int)
as range right
for values(
   20050101
    ,20050201
    ,20050301
    ,20050401
    ,...
);
```

It should be noted that the example uses the integer type and earlier it was mentioned that the partitioning function will be based on date, monthly, quarterly or annual ranges, so the DateTime type should be used. This is one more conscious optimization procedure used in data warehouses [8]. In queries of tables with billions of records, it matters what data type the WHERE condition will be based on. With the integer type, the condition on the field on which the partition is based will run faster, hence the change of the data type from DateTime to integer.

Based on the partitioning function, we create a schema based on the clustered index. An example of creating a schema based on a partitioning function is shown in listing 2.

Listing 2. Partitioning scheme based on the monthly partitioning function.

```
create partition scheme PartitionSchemaByMonth
as partition PartitionFunctionByMonth
all to ([PRIMARY])
```

Now tables can be created where we will generate millions or even billions of records. So, for the purposes of this research, a partitioned table was created (FactSalesPartitioned) along with an identical table without partitions (FactSales). Exactly the same data will be inserted into both tables and the execution times will be measured on the same SELECT query run against both tables. The reading times given from these tables will be recorded, with an additional 100 million records added to each table each time.

The structure of the partitioned table is shown in listing 3.

Listing 3. Partitioned table structure; the unpartitioned table will have the same structure.

```
CREATE TABLE [dbo].[FactSalesPartitioned](
    [SalesKey] [bigint] IDENTITY(1,1) NOT NULL,
    [DateKey] [int] NOT NULL,
    [EmployeeKey] [int] NOT NULL,
    [CustomerKey] [int] NOT NULL,
    [ProductKey] [int] NOT NULL,
    [SalesValue] dec(5,2) NOT NULL
    )
```

The next step is to use the partitioning scheme when creating the clustered index on the partitioned table (FactSalesPartitioned). It should be noted that the creation of this index is best done immediately after creating the table, because creating it on a table containing data may take a long time. An example of an index definition on a partitioned table is shown in listing 4.

Listing 4. Index using a partitioning scheme.

```
create unique clustered index [PartitionedIndexReport] on
[dbo].[FactSalesPartitioned]([SalesKey],[DateKey])
on PartitionSchemaByMonth ([DateKey])
```

The same clustered index, but without the partitioning scheme, was created on the second, unpartitioned table (FactSales). The index definition on the table without partitioning is shown in listing 5.

Listing 5. Definition of the Index on the table without partitioning

```
create unique clustered index [IndexReport]
on [dbo].[FactSales]( [SalesKey],[DateKey])
```

The last step is to use the partitions correctly in SELECT queries. The query normally does not differ from the query on a table without partitions. This is important because you do not need to change existing report queries. It is only necessary for the condition in the WHERE clause to be based on the column on which the partition index was built, which in the example is the DateKey field. Everything takes place in the database engine, specifically in the query optimizer which only reads data from partitions falling within the date range limited by the WHERE condition [1][4]. Listing 6 shows the use of partitions in the SELECT query.

Listing 6. A select query on a partitioned table is no different from a query on a table with no partitions.

```
select EmployeeKey, SUM(SalesValue)
from [dbo].[FactSales] --unpartitioned table
where DateKey between 20070925 and 20070927
group by EmployeeKey

select EmployeeKey, SUM(SalesValue)
from [dbo].[FactSalesPartitioned] -- partitioned table
where DateKey between 20070925 and 20070927
group by EmployeeKey
```

In these queries, the WHERE condition must be based on a field in the DateKey partitioning scheme, otherwise the partitioning function will not work. The partitioning function is based on the integer type and dates written as numbers, e.g., September 27, 2009 will be 20090927.

3. Comparative partitioning research

A training database – the AdventureWorksDW2017 data warehouse – was used to test partitioning compared to a solution without using partitioning. This database contains sample data very similar to the data in real systems of this type. The amount of data in the training database is, however, insufficient to carry out measurements because it contains only tens of thousands of records in individual tables. In real systems, the normal situation is an amount of data reaching millions or even billions of records in a single table. To reflect the real operation of a data warehouse system as much as possible, new records were generated based on a combination of data from the original dimensions of sellers, products and customers: DimEmployee, DimProduct and DimCustomer for existing dates in the DimDate dimension. The data in the FactSales and FactSalesPartitioned tables were generating evenly, exactly 1 million records for each day as follows. From the dimension tables, records were randomly selected in quantities:

- 100 products from the DimProduct product catalog, which contains over 600 items,
- 1,000 clients from the DimCustomer client file, which contains over 18,000 items,
- 10 salespeople from the DimEmployee employee file, which contains nearly 300 items.

and their combinations were inserted into the Fact tables. In this way, an additional 1 million daily data records were obtained ($100 \times 1000 \times 10 = 1{,}000{,}000$).

The measurements were performed starting from 1 million records, to 10 million and later every 100 million up to 1 billion records. Like this for each significant number of records: 1, 10, 100, 200 ... 1,000 million, measurements were taken of the amount of time the SELECT command took to reading data from the last 3 months, i.e. 3 million records.

These queries are presented in listing 6, while the measurement results themselves are presented in Table 1 in the Optimization results section.

As the study methodology required the records to always come from one partition, at the 700 million point, the measurement was performed at exactly 703 million, because this number fell on a partition boundary and crossing this would affect the query times, invalidating the testing process.

The total number of generated records was 1 billion. The insertion time of each of 100 million records was about 15 minutes.

4. Measurement of query execution times

The measurements concerned the execution times of the SELECT queries from listing 6, always for the last 3 million records added to the tables. The time was measured simply with a precision in milliseconds. The timing script is shown in listing 7.

Listing 7. Measuring query execution times

```
declare @startTime datetime
declare @stopTime datetime
select @startTime = getdate()
...
    -- SELECT from partitioned and unpartitioned tables
...

      select @stopTime = getdate()
select datediff(ms,@startTime,@stopTime)/1000.00
```

It is worth noting that in this study, accurate time measurement is not important, because it is mainly about checking whether the query execution time on a partitioned table will remain at the same level while the time of performing the same query on a table without a partition will increase. It does not matter if it is one second, two or five seconds. It is only important that the query execution time on a partitioned table is independent of the data increment.

5. Optimization results

During the measurements, attempts were made to maintain constant measurement conditions:

- the same table structures – FactSales, FactSalesPartitioned,
- always the same number of records in the select – 3 million,
- the same definition of a clustered index based on the SalesKey and DateKey columns,
- exactly the same data inserted into both tables.

The results presented in Table 1 clearly indicate that the select query times from the partitioned table are relatively constant and fluctuate around 2 s. It is different for the table without partitioning, where this time always increases with the increase of data. I remind you that the tested select always operated on the same, invariable number of 3 million records.

This is very well illustrated in figure 1 where the orange line marks the time to execute the SELECT query on the table with partitioning. It is clear that this line is flat; it remains constant even for a billion records. The blue line represents the execution time of the same SELECT query on a table without partitions, and is noticeably different. As the records in the table increase, this time also increases.

This is better illustrated on a graph with a logarithmic scale (figure 2), where there is an upward trend for a successive orders of magnitude of the number of records in the table. Execution time increases to an unacceptable level and is almost two orders of magnitude worse results for a billion records.

Table 1. Times of select query performed on tables with and without partitioning

Number of records in millions	The execution time of SELECT from the table without partitioning	The execution time of SELECT from the table with partitioning
	s	s
1	0.24	0.25
10	1.85	1.88
100	8.53	0.89
200	17.18	1.11
300	23.45	1.93
400	33.89	1.83
500	44.11	1.76
600	55.77	1.95
703	66.62	1.88
800	77.88	1.85
900	93.03	2.01
1000	108.98	1.93

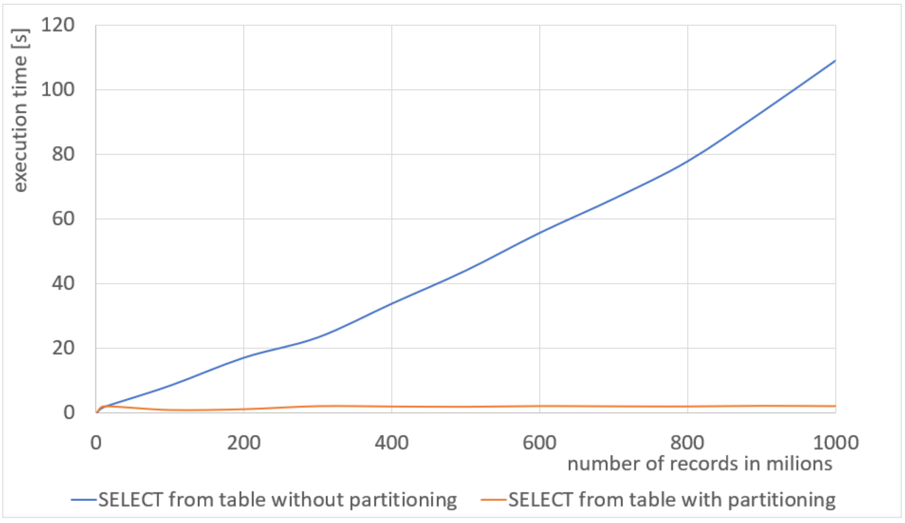

Fig. 1. Execution times of SELECT query on a table with and without partitioning in the range from 1 million to 1 billion records in the table

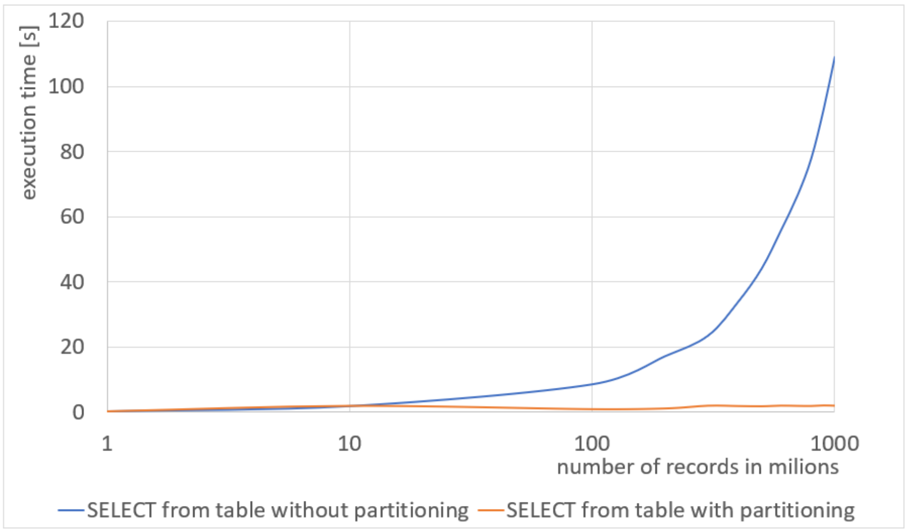

Fig. 2. Execution times of SELECT query on a table with and without partitioning, on a logarithmic scale

6. Query execution plan analysis

Let us take a look at our queries from listing 6, in terms of which of their parts significantly affect the time it takes to complete the query.

Listing 8. Selected query analyzed for the query execution plan

```
select EmployeeKey, SUM(SalesValue)
from [dbo].[FactSalesPartitioned] – partitioned table
where DateKey between 20070925 and 20070927
      group by EmployeeKey
```

This is undoubtedly a grouping of data and an aggregation function of SUM, as well as the select for records meeting the WHERE condition by the way of which an index scan is done. The rest of the query has nearly zero cost of execution, as shown in figure 3.

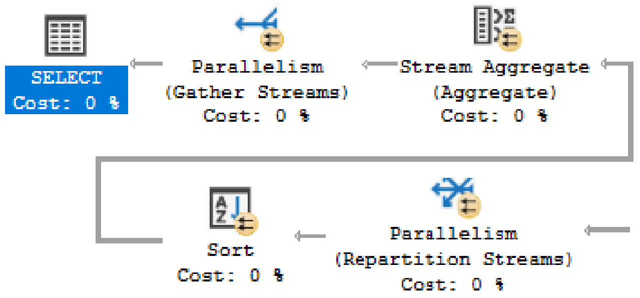

Fig. 3. Part of the execution plan with zero cost, not relevant for further analysis

We will focus on analyzing only the components of the query whose query cost is significant – greater than zero. In our query, this will be the aggregation when grouping and an index scan under the WHERE condition. In figure 4, the query plans are compared for 300 and 400 million records.

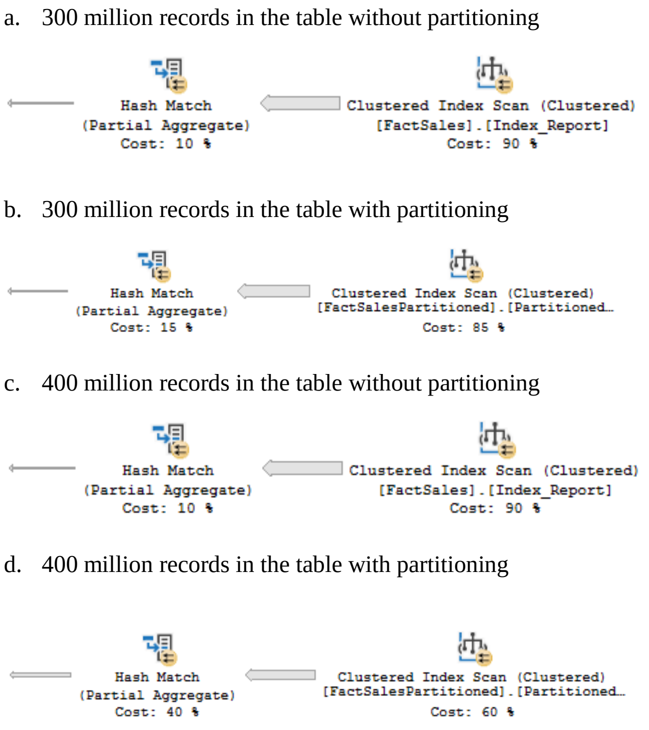
a. 300 million records in the table without partitioning
Hash Match
(Partial Aggregate)
Cost: 10 %
Clustered Index Scan (Clustered)
[FactSales].[Index_Report]
Cost: 90 %
b. 300 million records in the table with partitioning
Hash Match
(Partial Aggregate)
Cost: 15 %
Clustered Index Scan (Clustered)
[FactSalesPartitioned].[Partitioned…
Cost: 85 %
c. 400 million records in the table without partitioning
Hash Match
(Partial Aggregate)
Cost: 10 %
Clustered Index Scan (Clustered)
[FactSales].[Index_Report]
Cost: 90 %
d. 400 million records in the table with partitioning
Hash Match
(Partial Aggregate)
Cost: 40 %
Clustered Index Scan (Clustered)
[FactSalesPartitioned].[Partitioned…
Cost: 60 %

Fig. 4. Part of the query execution plan with a significant share of the cost of the query, for different numbers of records (300 and 400 million) in tables with and without partitioning

As can be seen in Figure 4, the select query execution plan from a table without partitioning is almost constant for different numbers of records in the table and is distributed between the aggregate (10%) and the index scan (90%).

The situation is different for the partitioned table. In this case, while maintaining the same query time, its cost is distributed between the aggregate and the scan, and decreases for the scan (85% → 60%) in favor of the cost of the aggregate (15% → 40%) which increases as the number of records increases in the partitioned table. This means that with an increasing number of records in the partitioned table, index scanning takes place after the partition to which the WHERE condition applies, so has a smaller and smaller share in the cost of the entire query.

7. Conclusions

The measurements clearly indicate the effectiveness of the use of table partitioning when optimizing databases in terms of SQL query times. The most important advantage of this solution is the independence of query performance from the number of records in the table. This is confirmed by the results of measurements of query execution times and analysis of query execution plans. This is crucial in databases with a very large number of records with a large daily increase in data, such as data warehouses. A common problem on systems with a very large database is the slowdown of reports over time. At the beginning, the system works quickly and efficiently, but it slows down over time and after a few years without implementing appropriate solutions, such a system may stop responding within an acceptable time and the reporting process will take minutes or even hours. Thanks to partitioning, we can achieve the same system performance at the beginning, right after starting and after a few years of its operation.

References

[1] Chodkowski A.: Partycjonowanie tabel a wydajność zapytań w SQL Server, seequality.net, 2017, [https://pl.seequality.net/partycjonowanie-tabel-wydajnosc-zapytan-sqlserver/].
[2] Kumar A., Jitendra Singh Yadav: A Review on Partitioning Techniques in Database International Journal of Computer Science and Mobile Computing 3(5), 2014, 342–347.
[3] Matalqa S., Mustafa S.: The effect of horizontal database table partitioning on query performance. The International Arab Journal of Information Technology 13(1A), 2016, 184–189.
[4] Microsoft documentation, Partycjonowanie danych poziomych, pionowych i funkcjonalnych, [https://docs.microsoft.com/pl-pl/azure/architecture/best-practices/data-partitioning].
[5] Qi W., Song J., Bao Y.: NPA: Near-uniform range partition approach for increased partitioning in large database. 2nd IEEE International Conference on Information Management and Engineering – Chengdu, 2010, 101–106, [http://doi.org/10.1109/ICIME.2010.5477529].
[6] Song J., Bao Y.: NPA: Increased Partitioning Approach for Massive Data in Real-Time Data Warehouse. 2nd International Conference on Information Technology Convergence and Services – Cebu, 2010, 1–6, [http://doi.org/10.1109/ITCS.2010.5581277].
[7] Watson H.: Recent Developments in Data Warehousing. Communications of the Association for Information Systems 8, [http://doi.org/10.17705/1CAIS.00801].
[8] Zheng K., Gu D., Fang F., Zhang M., Zheng K., Li Q.: Data storage optimization strategy in distributed column-oriented database by considering spatial adjacency. Cluster Computing 20(4), 2017, 2833–2844, [http://doi.org/10.1007/s10586-017-1081-3].

Ph.D. Eng. Piotr Bednarczuk
e-mail: Piotr.Bednarczuk@wsei.lublin.pl

He is a doctor in the Institute of Computer Science at the University of Economics and Innovation in Lublin. Studied and defended his PhD thesis at Lublin University of Technology. He supports his scientific knowledge with professional practice gained in a leading IT company, where he has been working for over 15 years, currently as the head of the database solutions department in the mobile systems department. His research area focuses on the software engineering web database systems, mobile-device systems and databases and data warehouses.

https://orcid.org/0000-0003-1933-7183

otrzymano/received: 30.06.2020 *przyjęto do druku/accepted: 15.09.2020*