Project 2: Performance Tuning

T-508-GAG2 Performance of Database Systems

Instructors: Björn Þór Jónsson, Grímur Tómas Tómasson

Teaching assistant: Patrekur Patreksson

March 27th 2019

Group name: Unicorns

Names:

Daníel Örn Melstað (danielm15@ru.is)

Halldór Bjarni Stefánsson (halldors17@ru.is)

Hera Hjaltadóttir (hera16@ru.is)

Martha Guðrún Bjarnadóttir (marthab16@ru.is)

Víðir Snær Svanbjörnsson (vidir17@ru.is)

Table of contents

Abstract

Optimization on database (DB) performance is important for customers and users all around the world, it can save a lot of time and a lot of disk space. This report discusses DB performance before and after tuning with measurements on a bank DB by using Microsoft SQL server Management studio 2017. If a DB is performing poorly it can most likely be tuned with a few approaches, in this report we'll show how we achieved better performance by tuning the DB.

1. Introduction

This report provides an overview of the methods and results of tuning on an unoptimized relational database (DB). The goal of DB tuning is to reduce storage space and workload processing time without sacrificing the integrity of the results returned from queries or the consistency of the data itself.

Our project is to improve the performance of the database with four different queries provided by our teachers using the optimization techniques we have learned so far. These four queries use various DB operations in different ways and were intentionally very inefficient so we could fully grasp the effects of optimizing them.

DB tuners make judgments on how the DB system should perform based on how it's used [1]. A tuner must think about how the system will be used with a global perspective. If many users connect globally to a DB system it must perform on demand without users waiting impatiently for a request to be processed.

The experiment was performed by following certain steps to optimize the DB and specific workload queries. In the beginning, data types were optimized to consume less disk space.

The DB size was reduced to ⅓ of its original size giving less execution time for queries. After tuning the data types the SQL queries that were given were optimized to perform faster by implementing shortcuts to retrieve data faster. Clustered indices were made to work well with the queries and were added on the biggest tables that showed incredible performance increases in regards to the workload.

By comparing the performance before and after DB tuning on a specific laptop the results were positive to our expectations. Overall the performance depended on implementation of queries, indexing wisely and hardware stats.

The experiments were done for us to get a deeper understanding of DB tuning in SQL server and see differences in CPU time, wall clock time, reads and writes on the DB.

This report is structured as follows. Section 2 covers the different methods used to tune databases, Section 3 describes the technical environment in which all measurements were made, along with a description of the database and the workload. We then explain changes made to the database and workload during our tuning effort. Section 4 describes the results of our tuning on each query. Section 5 is a discussion of our results, followed by a conclusion in section 6.

2. Background

Most systems and software today rely on some type of DB to keep track of data. In the architecture of these systems the DB is frequently the most integral part and as solutions become more global the demand for efficient, large scale DB's is higher than ever.

With the amount of data being collected the size of these DB´s can quickly become enormous which as a result can make simple queries take hours to run and the need to buy more storage increases. As developers and users demand responsiveness, this delay is unacceptable. To optimize this process, we employed a handful of different methods in DB tuning and applied them to our DB.

2.1 Database tuning

Optimizing data types

Data is stored in a table as a data type. In our case the data is stored in types that occupy much more space than is actually needed.

When we observed the data types we were focusing on the right size of each data type for the data. We saw data types that were too big for the data needed so clearly there was a chance to change them and shrink the size of the database. By shrinking the DB with the right size of each data type the performance increased.

Having a data type that uses for example 4 bytes but the analysis shows that the actual data uses at most 1 byte then we can reduce it to a 1 byte data type which saves 75% of the space used for a given column. On a small scale this change is fairly insignificant but on a large scale it can be drastic.

Optimizing queries

There are many things to consider when tuning queries to decrease their running time. One thing that can largely affect the running time is the order in which we narrow down our selection. Narrowing our selection as much as possible early in the query means less work is needed to do most operations later in the query, such as table scans and joins.

Queries that always perform a full table scan for every execution can take exponential time with an increase of amount of data in the DB. That's why selecting a specific range for example can limit the amount of searching through a table.

Another factor that can hugely affect the running time of queries is using explicit inner joins rather than implicit joins with WHERE clauses. Since using these clauses creates a cartesian join, it can easily generate a massive amount of rows that need to be searched through to find rows that match our clause. INNER JOIN instead only generates the minimum number of rows needed to satisfy the clause. [5]

Indexing

When using the right indices we focused on the bigger tables. indices in SQL server uses B-trees to store clustered nodes. When using a clustered index there comes an organized B-tree where each id is a node pointing to the actual value in a table. Putting the right index can speed up execution time of queries that require scanning because the list is now organized. In the experiment measurements were done before and after to see if the indices had any effect on performance which it did [7].

Partitioning

When partitioning it can be good to split big tables into smaller tables so queries don't have to scan through a lot of data, but this partitioning is often tailored to suit certain workloads.

3. Methods

All experiments were tested on laptop computers from each individual in the group. The experiments were performed and measured finally on a single laptop computer. The same DB was used as in project 1 and shrinked to about ⅓ of its original size.

3.1 Hardware

All of the tests were run on a Acer Nitro AN515-51 laptop, the specs for this computer are the following:
CPU: intel core i5 7200U 2.5GHz, 2701Mhz, 2Core(s), 4 Logical Processor(s)
RAM: 8 GB DDR4.
Storage: INTEL SSD MODEL SSDPEKKW256G7 with 237 GB storage.

3.2 Software environment

The experiment was performed with the following technical environment:

Server:	Microsoft SQL Server Enterprise 2017
Program:	Microsoft SQL Server Management Studio
OS:	64-bit Windows 10

3.3 The database

The SQL database provided from the teachers, called *innheimta*, is a large database with the same data distribution as a real bank database. Before any modifications it takes up a total of 32.2 GB of disk drive space, most of it taken by the largest three tables: *hreyfing*, *krafa* and *greidandi*, which are about 19 GB, 7.9 GB and 5.7 GB respectively.

3.4 The SQL queries

In this experiment the provided queries were four to be optimized:

- **Update.sql** - Updates all rows in *krafa* that have *samningur* with *vidskiptamadur_id* = 281
- **Insert.sql** - Inserts 1000 rows of test data into *krafa*
- **Aggregate_query.sq**l - Aggregate is a query that calculates the sum of all paid invoices for each bank branch and area code.
- **Non_aggregate_query.sql** - Selects invoices from *krafa* with many joins to other tables and has three different scenarios and four different customer ids for each scenario. Each scenario fetches different invoices based on the status or the due-date of the invoice for example.

3.5 Measurements

First the baseline performance of the original database was measured by using the unoptimized queries. This measurement was made so we could clearly see how much our changes were actually improving the performance of the database.

Each query was measured 5 times for each significant change made, the highest and lowest time discarded and the average of the remaining three calculated. For the timing process a timing script was provided that conveniently measures the speed and number of I/Os of the scripts and inserts them into an easily readable table. All of the measurements were made using the final version of the timing script provided.

For this experiment the following variables were measured:

- Reads
- Writes
- CPU time
- Wall clock time (duration)

3.6 Tuning phases

For the experiment the tuning methods are split into two different phases.

Phase 1 should consist of:

- ➢ Rewriting the queries
- ➢ Creating, deleting, editing and rebuilding indices
- ➢ Gathering statistics
- ➢ Changing column data types

Phase 2 should consist of all the methods used in phase 1, along with the following:

- ➢ Denormalize the database, as long as it does not affect any systems already using it
- ➢ Create Indexed Views
- ➢ Create additional tables
- ➢ Create triggers
- ➢ Partition tables
- ➢ Create views

3.6.1 Phase 1

Optimization for phase 1 was done in three steps in the following order:

Changing data types

Data types were observed in Microsoft SQL Management studio 17 and changed in phase 1. By looking at highest int value and need of char space from the 3 largest tables in the database each data type was written down with information on why it should be changed and the estimated reduction per row in bytes. The 3 biggest tables in the database were *hreyfing*, *krafa* and *greidandi*. After shrinking the database we didn’t see any reason to spend extra time shrinking smaller tables for the possibility of a few extra MB of storage space.

Changing queries

Rewriting the queries was done by limiting the WHERE clauses and optimizing JOIN clauses with a method from the IBM website [4]. The unoptimized non-aggregate and update queries contained a lot of WHERE and LEFT OUTER JOIN clauses that could mostly be replaced by INNER JOIN clauses.

For the aggregate query the biggest change was replacing the abundance of WHERE clauses that used AND operators with narrowed down joins and avoid using the HAVING clause.

The insert query contained an abundance of statements that retrieved the minimum value by using SELECT TOP (1) clauses alongside ORDER BY clauses. This was highly inefficient as it required sorting and all these statements were replaced with SELECT MIN() clauses. All casts were also replaced with absolute values and the data types for the variables being inserted were optimized.

Creating indices

We created a clustered index for the two biggest tables *hreyfing* and *krafa*. To select the columns for the clustered indices we tried a few different columns that we assumed would fit well and selected the best one based on performance. For the table *hreyfing* the column selected was *krafa_id* and for the table *krafa* it was the column *gjalddagi*. Nonclustered indices were created on *krafa, hreyfing, vidskiptamadur* and *greidandi.* We made our column selection for these indices by observing the execution plans of the workloads, the different joins needed by the workloads and the tuning advisor provided by SQL Server Management Studio. These observations provided us with some options and the best ones in relation to performance were selected.

3.6.2 Phase 2

In this phase we partitioned the biggest table called *hreyfing*. It was partitioned into two tables called *hreyfingMain* and *hreyfingDetails*. The five columns used in the workload were put into *hreyfingMain* and the rest of the 52 columns went into *hreyfingDetails*. After doing this the queries didn't have to scan through the entirety of the table which reduced the execution time for some workloads. We didn't create triggers because it didn't seem necessary because we could try creating views and indexed views. Creating indexed views on the table krafa didn't show a positive effect in performance. We also tried denormalizing the DB by adding columns from the table *heimilisfang* and *postnumer* to the table *greidandi* but it didn't improve the performance.

4. Results

The results are divided into two phases, phase 1 and phase 2. The queries were run 5 times, the top and bottom result discarded, and the average of the remaining measurements calculated. This was done with the altered database and tuned queries. After fixing the data types the whole DB shrank from 32.2 GB to 9.23 GB, or by 71.4%.

4.1. Phase 1 measurements

The measurements below show four different stages in the process. "Baseline" shows the original performance before the database and queries were changed. "Data types" shows the performance after the data types of the database were optimized. "Queries" shows the performance after the queries were optimized. "indices" shows the performance after clustered and nonclustered indices were created.

All measurements before and after the changes were compared in regards to rows and data returned to ensure consistency between changes.

4.1.1 Aggregate workload

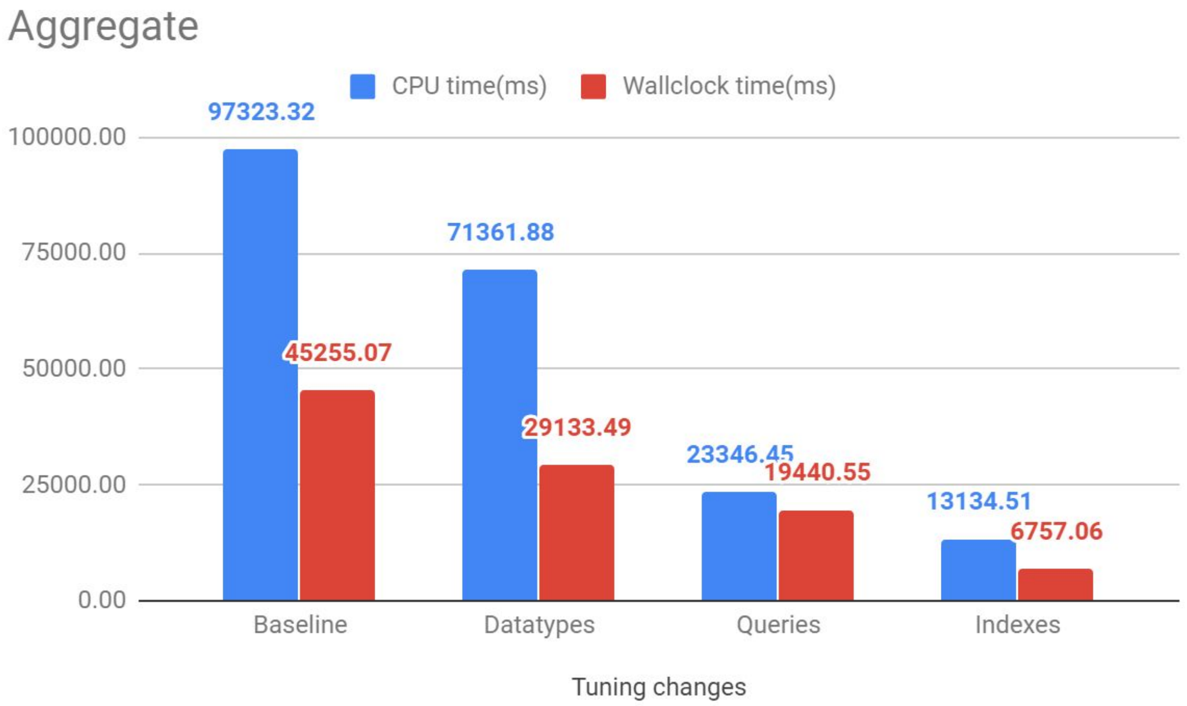


Figure 1: Phase 1 tuning on aggregate workload.

By looking at the wallclock time reduction we can see a significant increase in performance. The duration of the aggregate query was reduced to 15% of its original time. As shown in figure 1 there was no dominant tuning change but each change showed a steady reduction in wallclock time.

4.1.2 Insert workload

The total performance of the insert query increased mostly after the data types were changed and the database was shrunk. As shown in figure 2 the performance also increased after tuning the query.

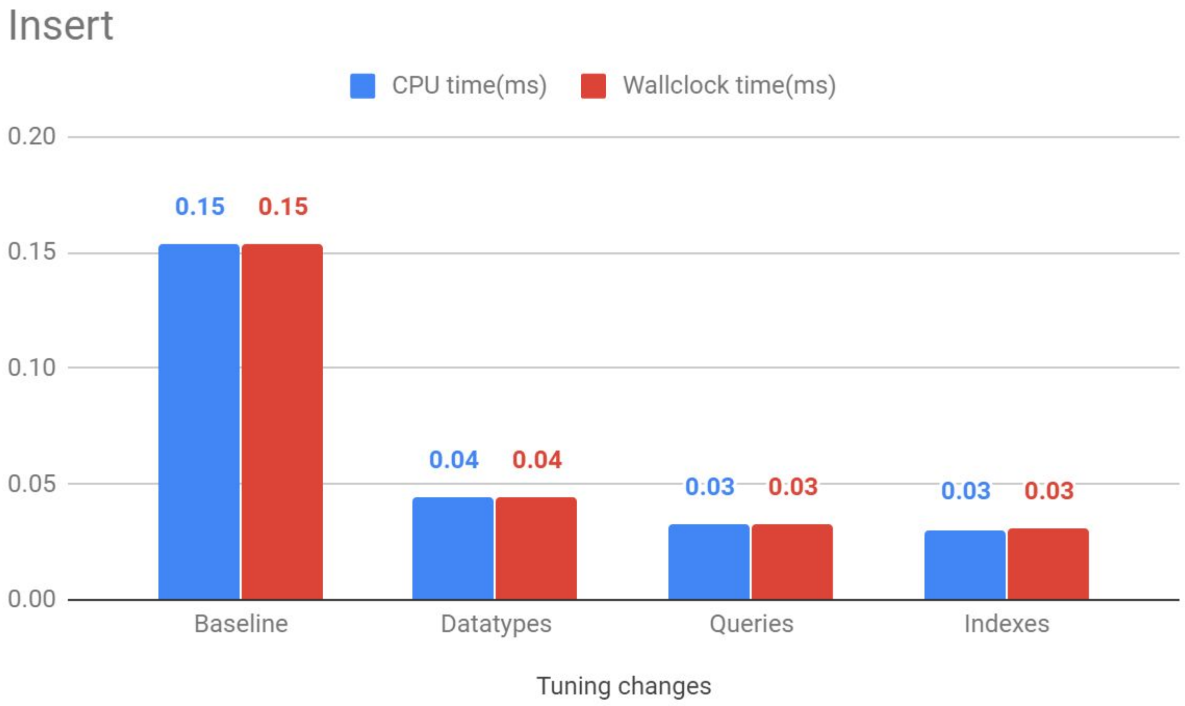


Figure 2: Phase 1 tuning on insert workload.

By looking at the wallclock time reduction we can see an increase in performance. The duration of the insert query was reduced to 20% of its original time. As shown in figure 2 there was a big reduction after changing the data types and a slight reduction in time after the changes in queries but no change after adding indices.

4.1.3 Update workload

A similar behavior to the insert was shown in the tuning of the update query. As shown in the update graph the greatest effect on performance was changing the data types and shrinking the database.

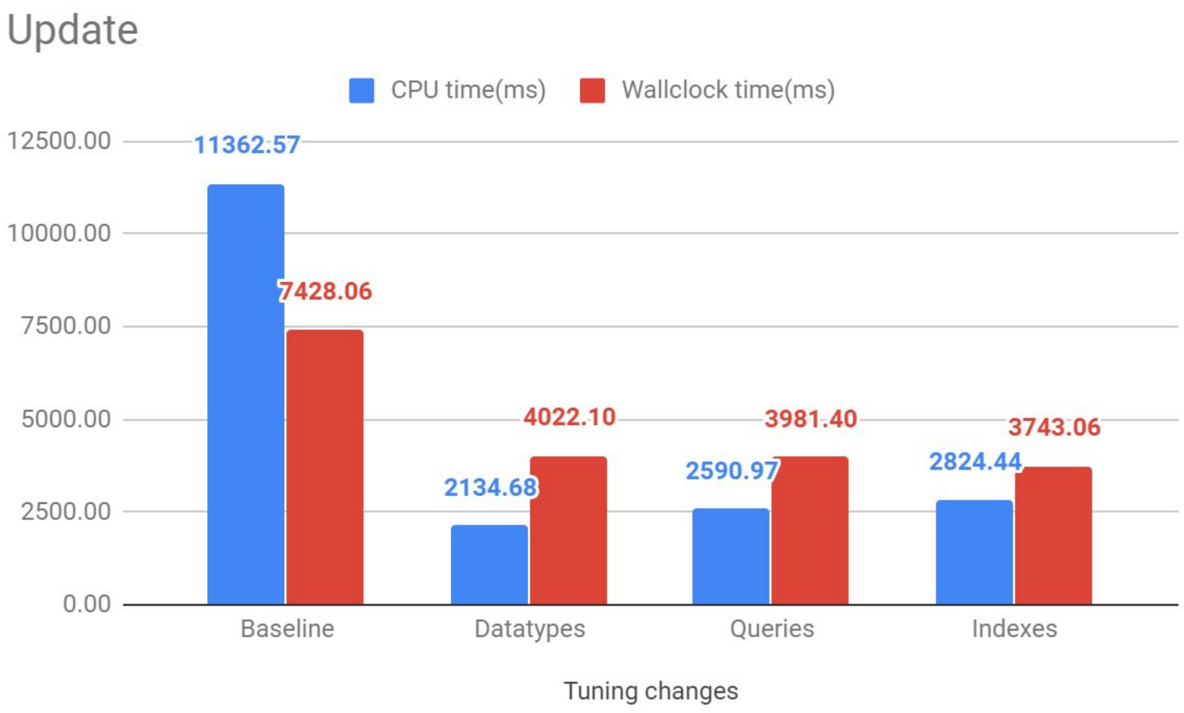


Figure 3: Phase 1 tuning on update workload..

By looking at the wallclock time reduction we can see an increase in performance. The duration of the update query was reduced to 49.6% of its original time. As shown in figure 3 there was a big reduction after changing the data types and a very little reduction in time after the changes in queries and adding indices made almost no difference.

4.1.4 Non-aggregate workload

Scenario 1 returns claim statements of paid invoices with due dates in the year 2008.

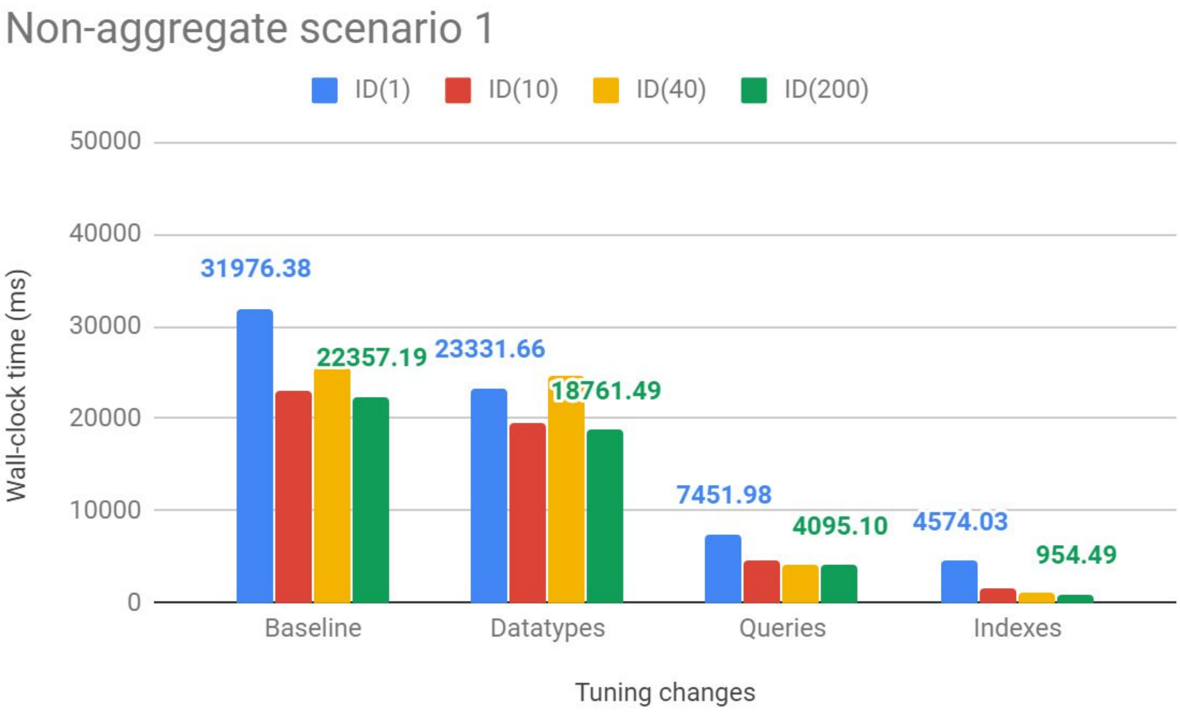


Figure 4: Phase 1 tuning for scenario 1 of non-aggregate workload.

By comparing the baseline and index measurements for all ID's in scenario 1 the query execution time was on average reduced to 7.9% of its baseline time.

Scenario 2 returns claim statements for a specific customer based on id that have due dates between 2010 and 2013.

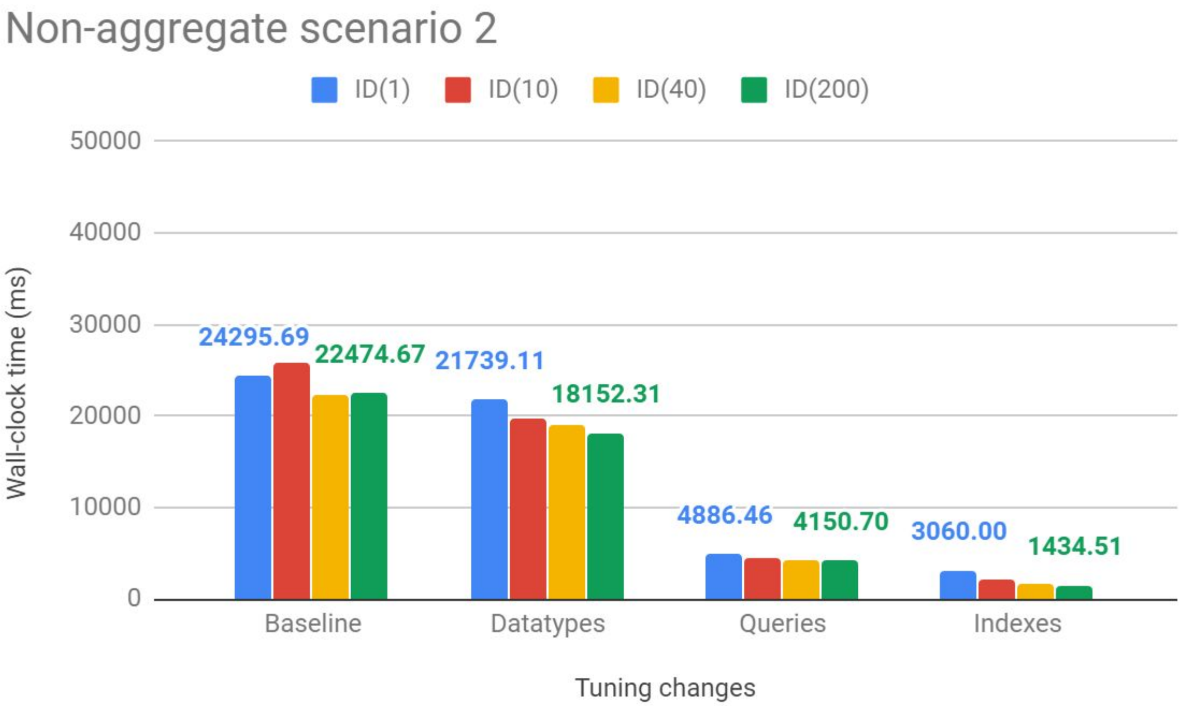


Figure 5: Phase 1 tuning for scenario 2 of non-aggregate workload.

Similar to scenario 1, scenario 2 had on average a reduction in query execution time to 8.7% of its baseline time.

Scenario 3 returns claim statements where invoices are unpaid and the invoice exists in *hreyfing* as a payment.

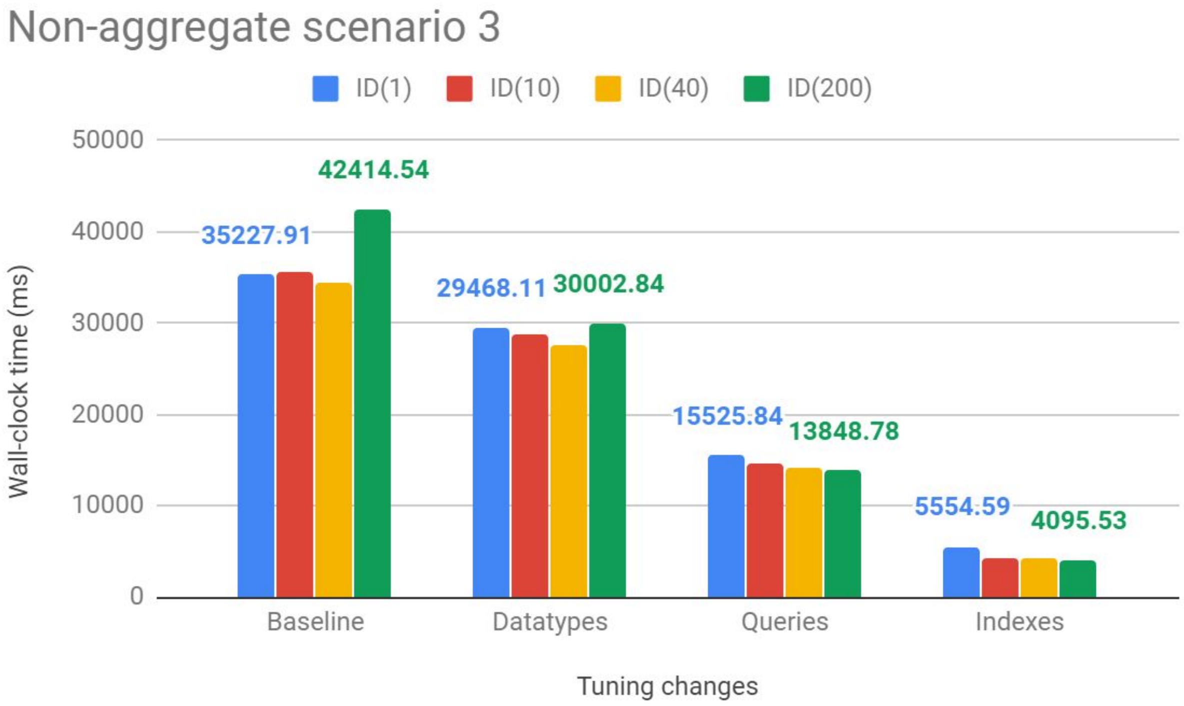


Figure 6: Phase 1 tuning for scenario 3 of non-aggregate workload.

In scenario 3 the query execution time was on average reduced to 12.4% of its baseline time.

4.2 Phase 2

In the graphs below we can see differences after partitioning the *hreyfing* table.

4.2.1 Aggregate workload

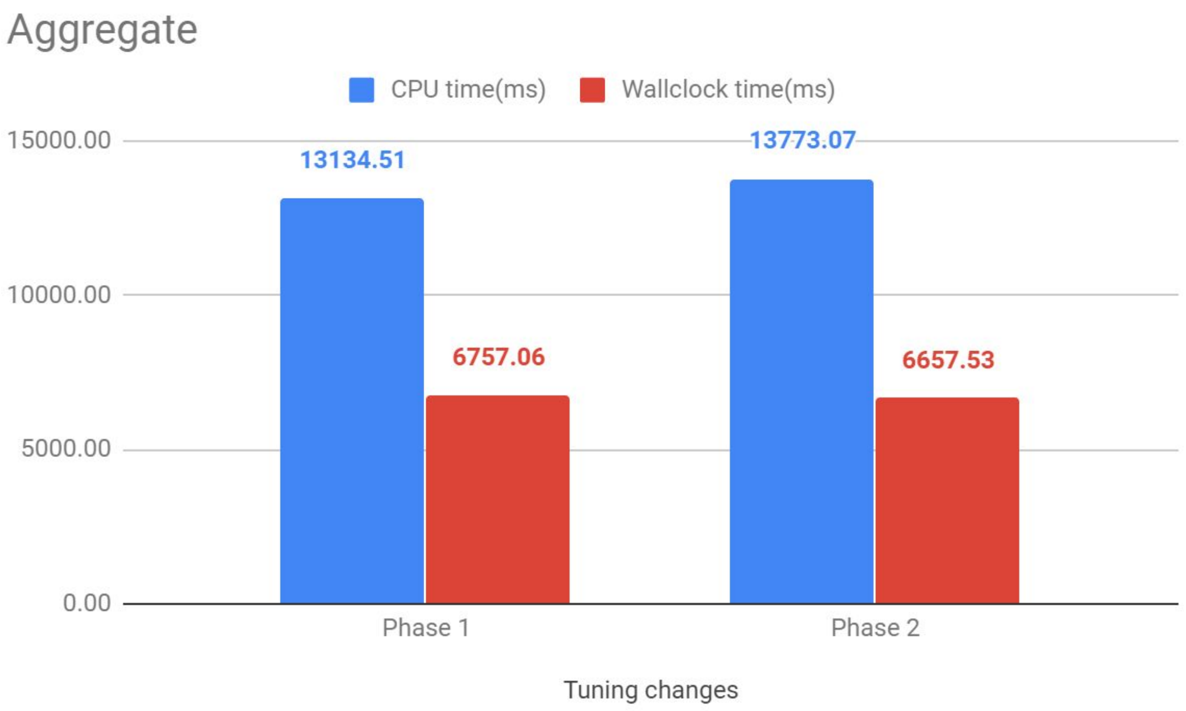


Figure 7: Aggregate tuning for phase 1 and phase 2.

The aggregate query in phase 2 showed a query execution time reduction to 98.5% of phase 1.

4.2.2 Insert workload

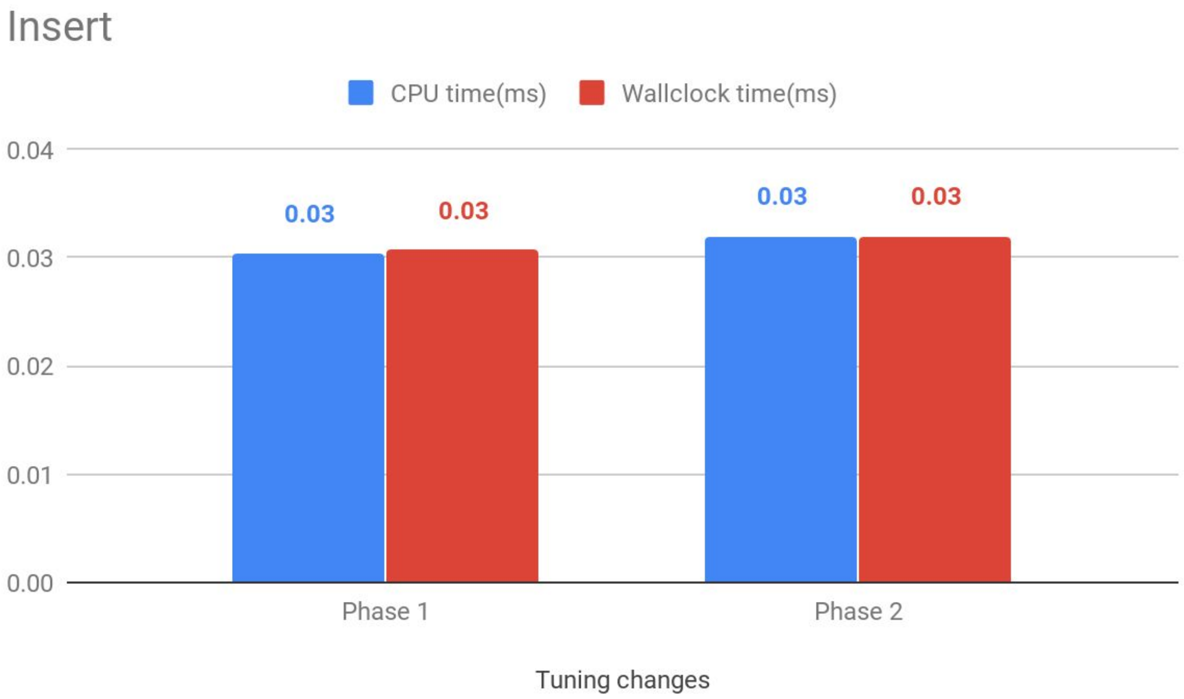


Figure 8: Insert tuning for phase 1 and phase 2.

No significant difference in execution time was observed for the insert query between phases.

4.2.3 Update workload

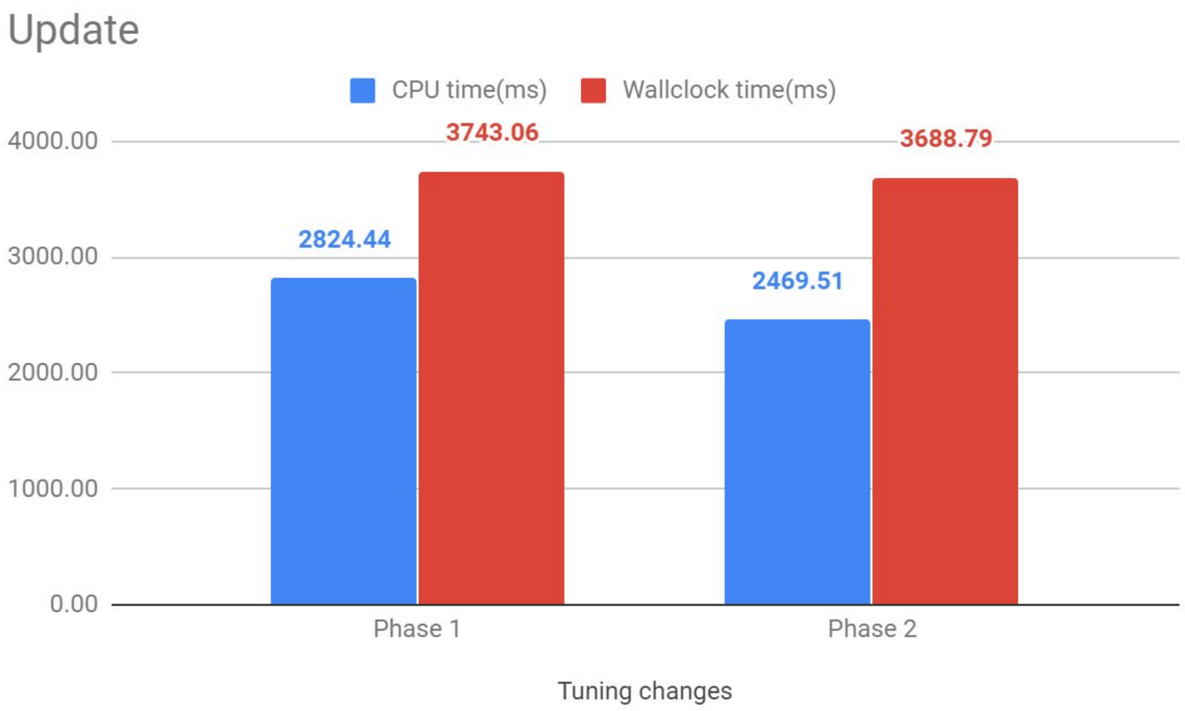


Figure 9: Update tuning for phase 1 and phase 2.

The update query in phase 2 showed a reduction in execution time to about 98.5% of phase 1.

4.2.4 Non-aggregate workload

Scenario 1 shows minimal changes after partitioning was performed with a slight increase in execution time for ID's 1 and 200 but a slight reduction in execution time for ID's 10 and 40.

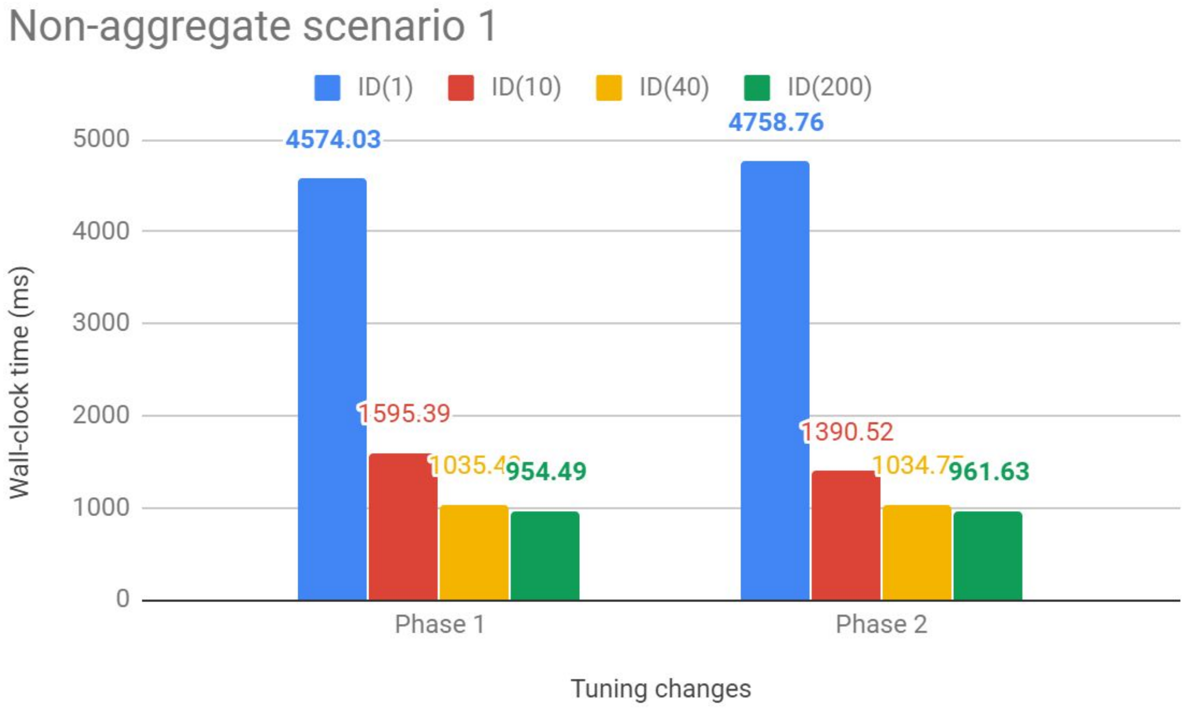


Figure 10: non-aggregate tuning for phase 1 and phase 2 scenario 1.

By calculating the difference between the averages of all ID's between phase 1 and phase 2 the partitioning shows an overall 0.02% decrease in wallclock time for scenario 1 in phase 2.

Scenario 2 had very similar results to phase 1 after partitioning but showed a slight reduction in execution time for ID's 1 and 200.

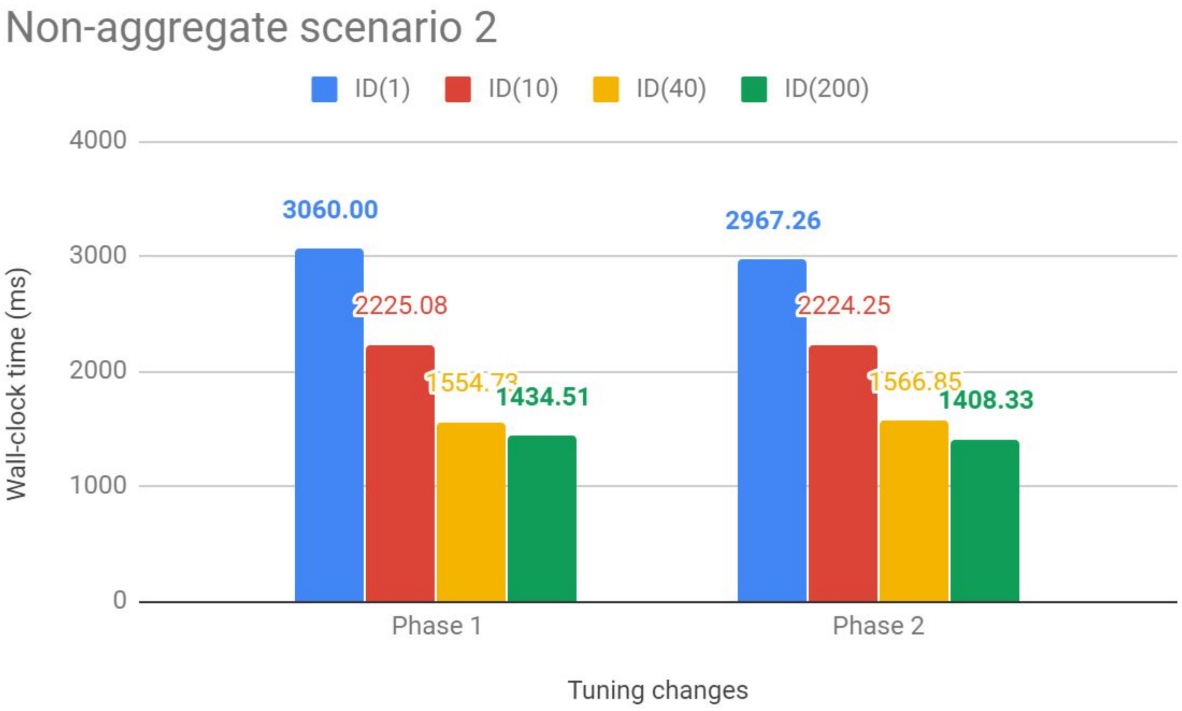


Figure 11: non-aggregate tuning for phase 1 and phase 2 scenario 2.

By calculating the difference between the averages of all ID's between phase 1 and phase 2 the partitioning shows an overall 1.32% decrease in wallclock time for scenario 2 in phase 2.

Scenario 3 after partitioning shows a noticeable decrease in execution time for ID 1 and a slight decrease for ID 10. However it shows an increase in execution time for ID's 40 and 200.

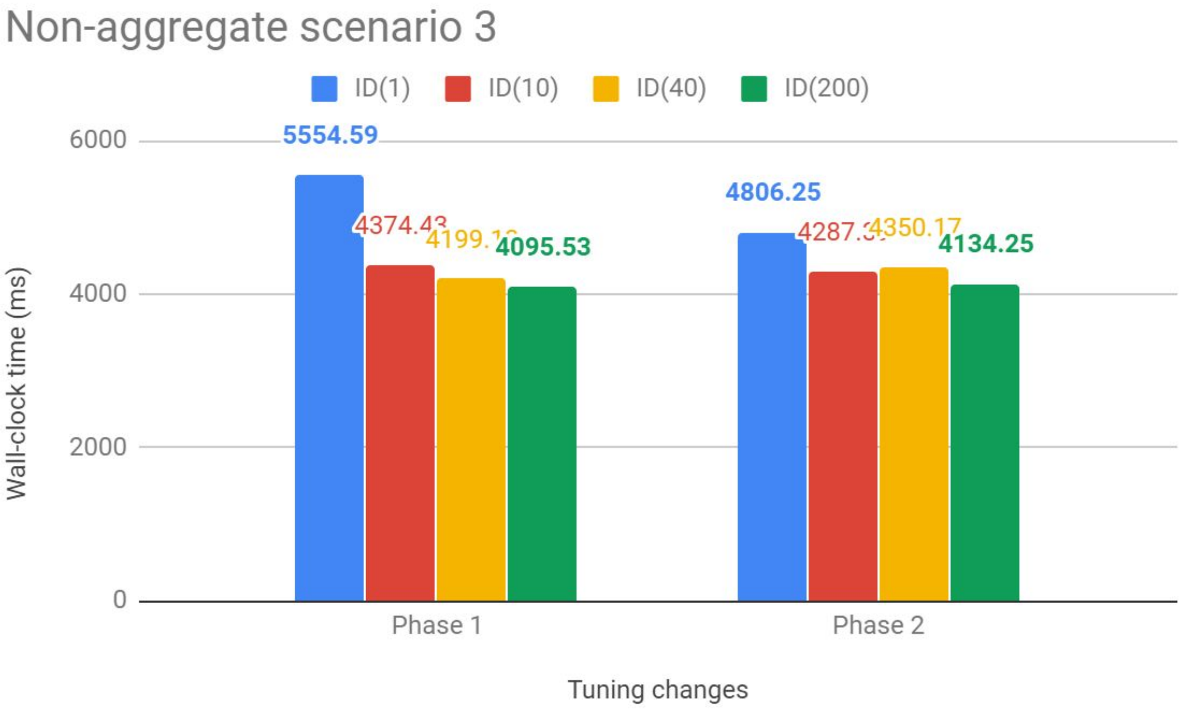


Figure 12: non-aggregate tuning for phase 1 and phase 2 scenario 3.

By calculating the difference between the averages of all ID's between phase 1 and phase 2 the partitioning shows an overall 3.6% decrease in wallclock time for scenario 3 in phase 2.

5. Discussion

Overall this experiment took a long time to perform, we were not very satisfied with the results from phase 2 and tried adding indexed views but our measurements didn’t show positive results. We were satisfied with the results from phase 1 and put our results into more visual statistical outputs.

When we reduced the size of the original database by changing the data types we got an incredible performance boost. Our guess is that the performance boost might have been because of the SQL servers buffer manager [6].

In our experience measuring the time taken for queries is a helpful tool to monitor the reduction in run time after optimization.

In this project we learned a lot about optimizing a DB on a laptop computer without thinking about hardware specs and their influence on performance. This of course plays its role in the world of DB systems and is an interesting factor used to optimize DB systems.

6. Conclusion

We tuned the database and query performance with time and space utilization in mind. The database (DB) size was reduced to 28.6% of its original size after optimizing the data types.
Four queries were optimized and all showed improvements after phase 1 but after phase 2 there was little change. In phase 1 we tested the improvements in three stages and overall the results showed 50-90% reduction in time from the baseline.

Data types of columns were changed to the smallest options that could fit the data to shrink the DB. Queries were optimized, which boosted performance. Clustered indices were added for even more performance except on insert and update queries.

Logical reads showed much better memory utilization after optimization so we can see that the queries don’t scan through as much data as before.

7. References

[1] Dennis Shasha and Philippe Bonnet, *Database Management Systems*, New york, America: Morgan Kaufmann Publishers, 2003, ch. 1, sec. 1.3, pp. 7.

[2] Paul Tero “Speeding up your website’s database”, *Smashing magazine*, March, 2011. Available: https://www.smashingmagazine.com/2011/03/speeding-up-your-websites-database/ Accessed: 27.February, 2019.

[3] Dennis Shasha and Philippe Bonnet, *Database Management Systems*, New york, America: Morgan Kaufmann Publishers, 2003, ch. 1, sec. 1.3, pp. 2.

[4] IBM “Techniques for improving the performance of SQL queries under workspace in the Data Service Layer”, *IBM*.https://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.developer.doc/refs/rsdperformanceworkspaces.htm Accessed: 27. February, 2019.

[5] “8 Ways to Fine-tune your SQL Queries (for production databases)”, *Sisense*, June, 2017. Available: https://www.sisense.com/blog/8-ways-fine-tune-sql-queries-production-databases/ Accessed: 19. March, 2019

[6] “Memory management Architecture Guide”, *Microsoft*, January, 2019. Available: https://docs.microsoft.com/en-us/sql/relational-databases/memory-management-architecture-guide?view=sql-server-2017

[7] “SQL Server Index Design Guide”, *Microsoft*, June, 2017. Available: https://docs.microsoft.com/en-us/sql/2014-toc/sql-server-index-design-guide?view=sql-server-2014

Appendix

The following are all the measurements from our experiments. "Baseline" represents the unchanged database with unchanged queries. Each value is an average of 5 measurements where the top and bottom measurements are excluded from the average.

Aggregate					
Tuning change	**CPU time(ms)**	**Wallclock time(ms)**	**Logical Reads**	**Physical reads**	**Logical writes**
Baseline	97323.32	45255.07	18311082	3466773	70
Datatypes	71361.88	29133.49	63226323	1050901	15
Queries	23346.45	19440.55	6521009.333	1090968	25
Indices	13134.51	6757.06	5533599.667	97740.66667	4
Phase 2	13773.07	6657.53	5564121.00	126550.33	5.00

Insert					
Tuning change	**CPU time(ms)**	**Wallclock time(ms)**	**Logical Reads**	**Physical reads**	**Logical writes**
Baseline	0.15	0.15	16.33333333	0	0
Datatypes	0.04	0.04	6	0	0
Queries	0.03	0.03	4.666666667	0	0
Indices	0.03	0.03	13	0	0
Phase 2	0.03	0.03	13.00	0.00	0.00

Update					
Tuning change	**CPU time(ms)**	**Wallclock time(ms)**	**Logical Reads**	**Physical reads**	**Logical writes**
Baseline	11362.57	7428.06	994734.3333	993581.3333	541
Datatypes	2134.68	4022.10	340721	340021	272
Queries	2590.97	3981.40	340721	339961.3333	271.6666667
Indices	2824.44	3743.06	344384.6667	339078	0
Phase 2	2469.51	3688.79	344408.00	339036.00	0.00

Non-aggregate scenario 1 id 1

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	19141.68	31976.38	2343083.333	2029268.333	0
Datatypes	13465.08	23331.66	993454	784446	0
Queries	9412.24	7451.98	2952325.667	414294	944.6666667
Indices	7036.85	4574.03	2750823.333	97661.66667	923.3333333
Phase 2	6978.09	4758.76	2751903.67	97700.00	925.67

Non-aggregate scenario 1 id 10

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	15762.49	23049.10	2040693.667	1994418	0
Datatypes	11316.18	19447.94	810331	756023	0
Queries	5374.71	4625.88	1978042.667	352516.6667	117
Indices	3958.79	1595.39	1695119.333	48121	113
Phase 2	3629.14	1390.52	1695528.00	48115.00	118.33

Non-aggregate scenario 1 id 40

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	17816.66	25508.14	2002048	1993675.333	0
Datatypes	13329.91	24764.87	786414	752617	0
Queries	4702.08	4207.51	1845830	343631.6667	6
Indices	3137.66	1035.49	1553022.667	41807	5.666666667
Phase 2	3085.52	1034.75	1552481.67	41807.00	6.00

Non-aggregate scenario 1 id 200

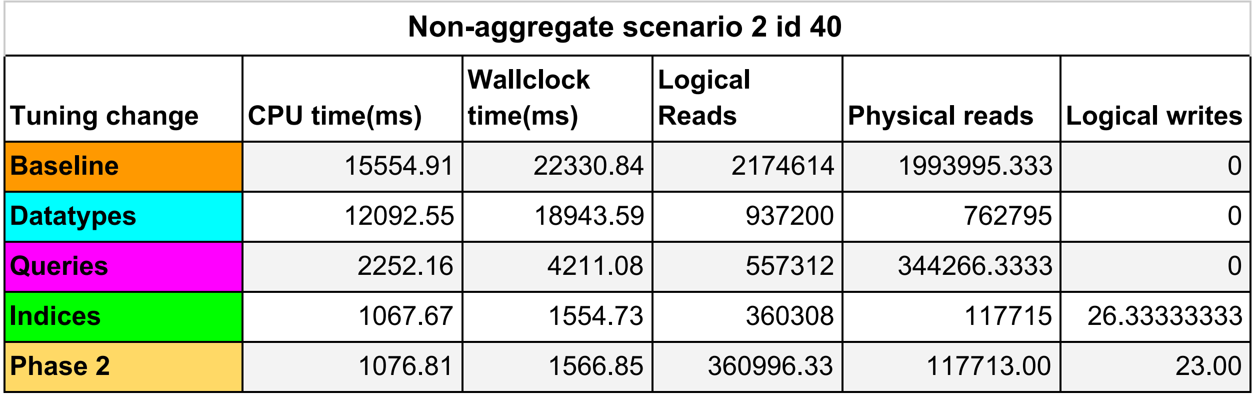
Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	13750.92	22357.19	1995126.667	1993682.333	0
Datatypes	8717.06	18761.49	782485	770600	0
Queries	4669.32	4095.10	1820624.667	340710.3333	0
Indices	3027.98	954.49	1524695	39967	0
Phase 2	3017.73	961.63	1524694.00	39967.00	0.00

Non-aggregate scenario 2 id 1

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	19178.41	24295.69	2287576.333	2009886.667	0
Datatypes	11669.30	21739.11	1027813	777163	0
Queries	4127.28	4886.46	1109870	392566.3333	0
Indices	3283.63	3060.00	1236607.333	157691.6667	352.3333333
Phase 2	3197.14	2967.26	1239983.67	157675.33	348.67

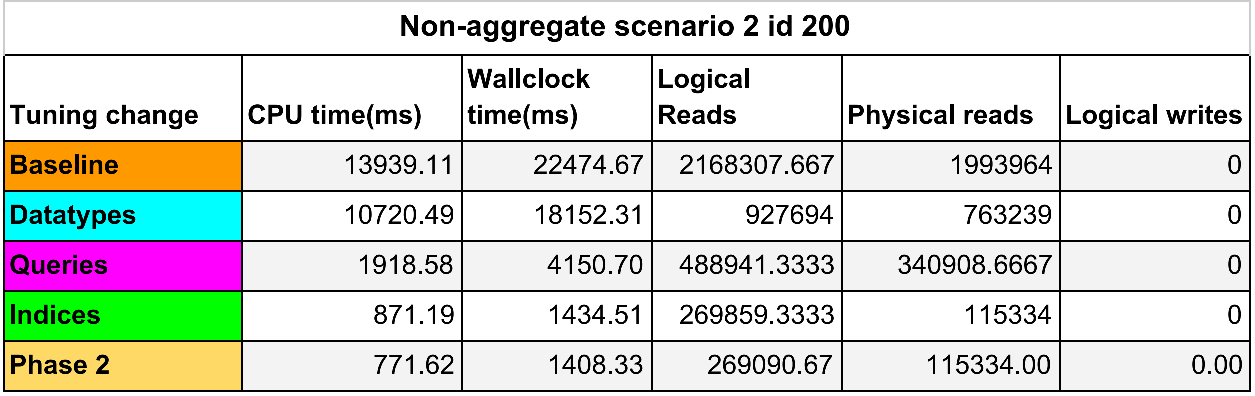
Non-aggregate scenario 2 id 10

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	18665.58	25791.90	2243637.667	1999318.667	0
Datatypes	13369.92	19808.73	980311	771541	0
Queries	3063.53	4519.14	879840.6667	352374.3333	0
Indices	2254.27	2225.08	779795.3333	124227	225
Phase 2	2094.00	2224.25	778710.33	124221.33	226.33



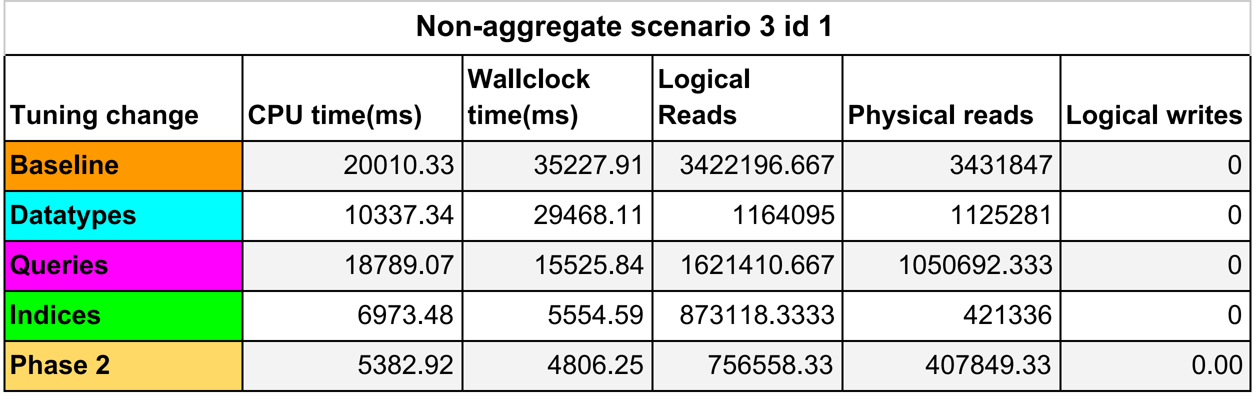
Non-aggregate scenario 2 id 40

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	15554.91	22330.84	2174614	1993995.333	0
Datatypes	12092.55	18943.59	937200	762795	0
Queries	2252.16	4211.08	557312	344266.3333	0
Indices	1067.67	1554.73	360308	117715	26.33333333
Phase 2	1076.81	1566.85	360996.33	117713.00	23.00



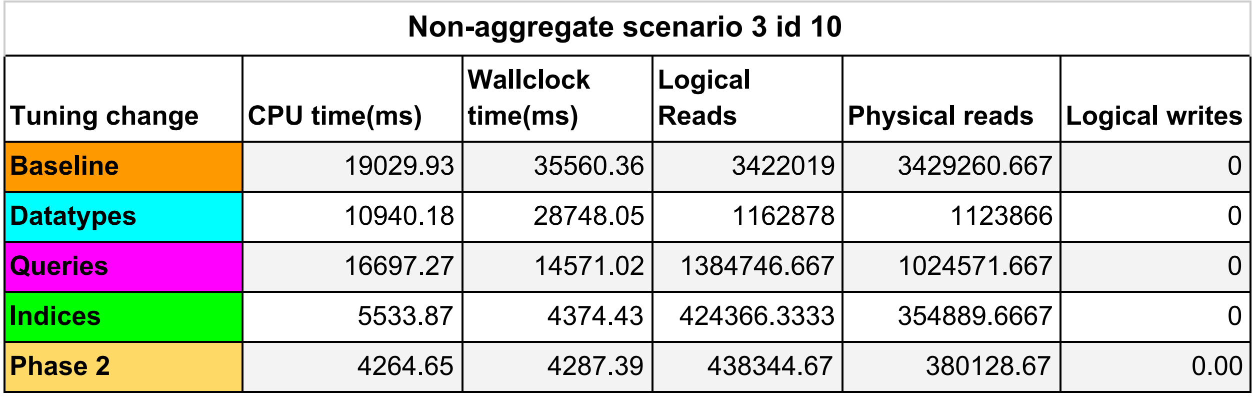
Non-aggregate scenario 2 id 200

Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	13939.11	22474.67	2168307.667	1993964	0
Datatypes	10720.49	18152.31	927694	763239	0
Queries	1918.58	4150.70	488941.3333	340908.6667	0
Indices	871.19	1434.51	269859.3333	115334	0
Phase 2	771.62	1408.33	269090.67	115334.00	0.00

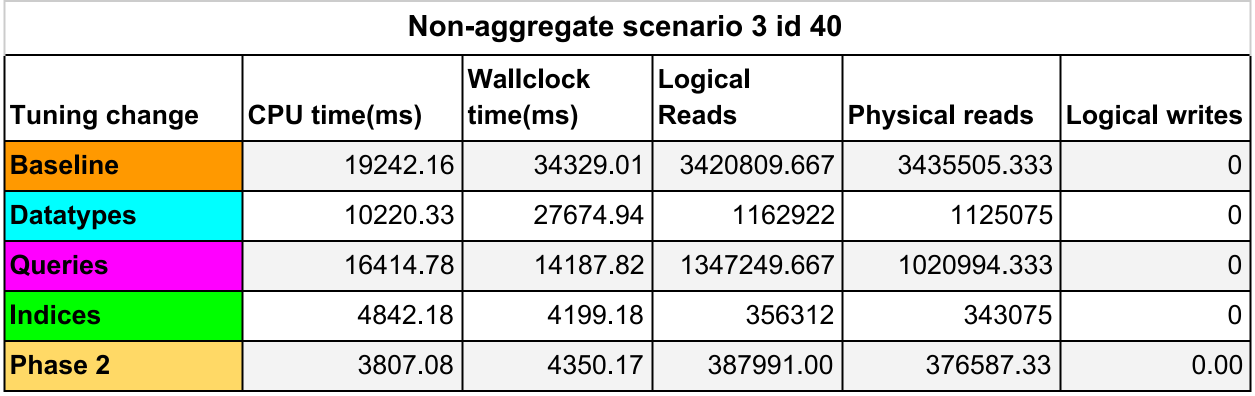


Non-aggregate scenario 3 id 1

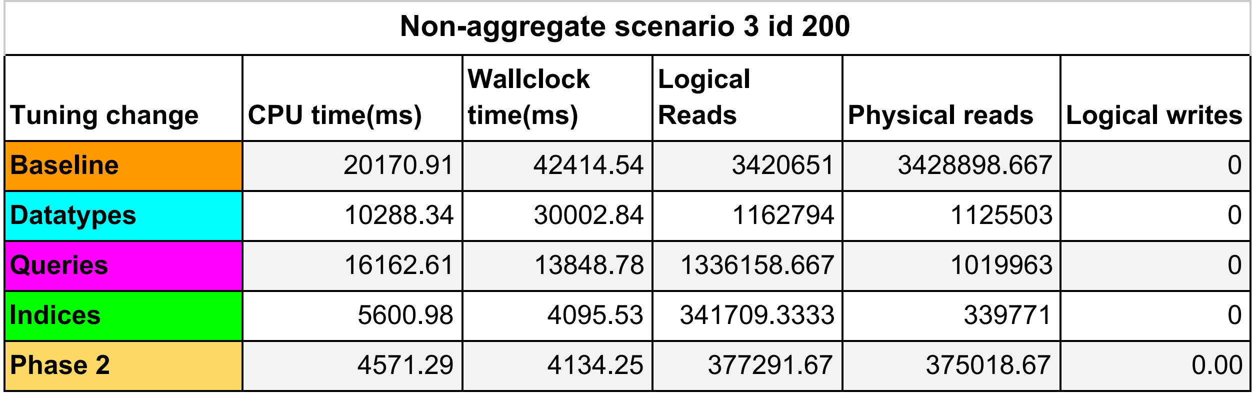
Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	20010.33	35227.91	3422196.667	3431847	0
Datatypes	10337.34	29468.11	1164095	1125281	0
Queries	18789.07	15525.84	1621410.667	1050692.333	0
Indices	6973.48	5554.59	873118.3333	421336	0
Phase 2	5382.92	4806.25	756558.33	407849.33	0.00



Non-aggregate scenario 3 id 10					
Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	19029.93	35560.36	3422019	3429260.667	0
Datatypes	10940.18	28748.05	1162878	1123866	0
Queries	16697.27	14571.02	1384746.667	1024571.667	0
Indices	5533.87	4374.43	424366.3333	354889.6667	0
Phase 2	4264.65	4287.39	438344.67	380128.67	0.00



Non-aggregate scenario 3 id 40					
Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	19242.16	34329.01	3420809.667	3435505.333	0
Datatypes	10220.33	27674.94	1162922	1125075	0
Queries	16414.78	14187.82	1347249.667	1020994.333	0
Indices	4842.18	4199.18	356312	343075	0
Phase 2	3807.08	4350.17	387991.00	376587.33	0.00



Non-aggregate scenario 3 id 200					
Tuning change	CPU time(ms)	Wallclock time(ms)	Logical Reads	Physical reads	Logical writes
Baseline	20170.91	42414.54	3420651	3428898.667	0
Datatypes	10288.34	30002.84	1162794	1125503	0
Queries	16162.61	13848.78	1336158.667	1019963	0
Indices	5600.98	4095.53	341709.3333	339771	0
Phase 2	4571.29	4134.25	377291.67	375018.67	0.00